SmartCropCNNLite: Real-life Tomato Crop Disease Detection using deep learning and Smart devices

By

Chandrashekhar Purandare

DISSERTATION

Presented to the Swiss School of Business and Management Geneva

In Partial Fulfillment

Of the Requirements

For the Degree

DOCTOR OF BUSINESS ADMINISTRATION

SWISS SCHOOL OF BUSINESS AND MANAGEMENT GENEVA

June 2024

SmartCropCNNLite: Real-life Tomato Crop Disease Detection using deep learning and Smart devices

by

Chandrashekhar Purandare

APPROVED BY

dr. LJILJANA KUKEC, Ph.D.

Dissertation chair

RECEIVED/APPROVED BY:

Admissions Director

ACKNOWLEDGEMENTS

This research study is conducted as a part of Swiss School of Business and Management (SSBM) Geneva, Doctor of Business Administration program. I would like to thank all the SSBM distinguished faculties and professors whose guidance played a very important role to set the foundation of this research work. A special thanks to Dr. Mario Silic, who has provided all the necessary guidance and support to complete this research work. As a thesis mentor, reviewed the research work and thesis deliverables regularly and provided valuable feedback to improve the overall quality of this research work and thesis report. This research would have not been possible without the availability of the required plant image dataset. Hence, I would like to thank the authors and creators of the PlantVillage and PlantDoc datasets.

ABSTRACT

SMARTCROPCNNLITE: REAL-LIFE TOMATO CROP DISEASE DETECTION USING DEEP LEARNING AND SMART DEVICES

Chandrashekhar Purandare
2024

Dissertation Chair: <Chair's Name>
Co-Chair: <If applicable. Co-Chair's Name>

Today agriculture industry is going through a fundamental transformation to adopt "High-Tech Farming" and "Precision Agriculture" leveraging sophisticated technologies such as robotics, IoT sensors, Computer Vision, GPS, and aerial images to improve crop yield. However, it is projected that the world population will reach 9.9 billion by 2050 from the current 8.1 billion population in the year 2024. Currently, nearly 800 million people suffer from hunger worldwide and 8% of the overall world population is expected to be undernourished by 2030. The agriculture industry is continuously challenged by climate changes and the draining of natural resources. Crop yield security is threatened by various diseases, which are caused by fungal, viral, and bacterial organisms. This research focuses on the detection of six types of adverse diseases that are commonly impacting Tomato crops using Computer Vision and Deep Learning classification methods. The research takes advantage of the proven benchmark performance of simple and lighter Convolution Neural Network (CNN) models with Transfer Learning. To suggest appropriate treatment, this research further predicts the degree of disease spread using Machine Learning and Deep Learning techniques. The trained CNN model deployed onto Android smartphones can detect tomato crop diseases in the actual farm (plantation) field with acceptable performance. The solution implemented as a part of this research can allow small and mid-scale farmers to detect the plant diseases along with disease severity to control disease spread. This research has the potential to enhance tomato crop yield per hectare, directly influencing global GDP and making a valuable contribution to the overall agricultural economy.

Keywords: AI in agriculture, Computer Vision, Deep Learning and CNN, Real-time Tomato Crop Disease Identification, Disease Severity Detection, Android Mobile App

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI: Artificial Intelligence

ANN: Artificial Neural Network

AR: Augmented Reality

AWS: Amazon Web Services

BN: Batch Normalization

BOVW: Bag-of-Visual-Words

BPMN : Back Propagation Neural Networks

CNN: Convolutional Neural Network

CPU: Central Processing Unit

CV: Computer Vision

DL: Deep Learning

FAO: Food and Agriculture Organization

FPGA: Field Programmable Gate Arrays

GCP: Google Computing Platform

GDP: Gross Domestic Product

GPRS: General Packet Radio Service

GPU: Graphics Processing Unit

HD: High Definition

HSV: Hue, Saturation, Value (Image processing parameters)

IoT: Internet of Things

K-NN: K-Nearest Neighbour

ML: Machine Learning

MLP: Multilayer Perceptron

NABARD: National Bank for Agriculture and Rural Development

OS: Operating System

PD: PlantDoc (Original) Dataset

PD-BAL: PlantDoc Balanced Dataset

PVD: PlantVillage (Original) Dataset

PVD-BAL: PlantVillage Balanced Dataset

RAM: Random-Access Memory

ReLU: Rectified Linear Units

SOA: State of the Art

SVM: Support Vector Machine

TYLCV: Tomato Yellow Leaf Curl Virus

UAV: Unmanned Aerial Vehicle (popularly known as Drone)

UN: United Nations

CHAPTER I:

INTRODUCTION

1.1 Background of the study

Currently, the agriculture industry is going through a fundamental transformation to adopt "High-Tech Farming" and "Precision Agriculture" leveraging sophisticated technologies such as robotics, IoT sensors, Computer Vision, GPS, and aerial images to improve crop yield. However, it is projected that the world population will reach 9.8 billion by 2050 from the current 7.8 billion population in the year 2020 (United Nations, 2019). Currently, nearly 800 million people suffer from hunger worldwide and 8% of the overall world population is expected to be undernourished by 2030 (FAO | IFAD | UNICEF | WFP and WHO, 2021). As per the World Bank Group (2020), the agriculture sector plays an important role in economic growth which contributed to 4% of Global GDP in 2018. In the case of some developing countries, agriculture contributes to more than 25% of GDP, and hence, the development of the agriculture sector can be a crucial factor in reducing poverty and improving economic prosperity. Tomato (Lycopersicon esculentum) is one of the most important crops worldwide with many nutritional values and contributing largely to an agricultural economy. This annual growing crop has a short growth period of 90 to 150 days and grows well on a wide range of soils (though well-drained, red loam soil with a 5 to 7 pH range is an ideal one) (Indian Council of Agricultural Research, 2017). India is one of the largest producers of tomatoes. In the year 2021, tomato production in India approximated to over 21 million metric tons ("Statista Research: Production volume of tomatoes in India FY 2015-2021," 2021).

In recent years, the agriculture sector witnessed many technological advancements, and it is heavily relying on precision agriculture and high-tech farming to increase crop yield, given the ever-increasing food demands. Despite increased crop yield, food security is always threatened due to climate changes, reducing natural resources, and the outbreak of various crop diseases (Savary et al., 2012). For traditional plant disease methods, expertise in plant diseases is necessary for individuals to be able to identify the diseased leaves accurately. Furthermore, measurement of the correct degree of disease demands a sophisticated lab infrastructure (Singh et al., 2020). Usually, there is a lack of Plant Pathology experts and Labs in the remote regions of developing countries, which makes it difficult for small-scale farmers to detect the plant disease early and control the potential crop yield loss. In the past, due to a lack of early detection of crop diseases or

misinterpretation of the degree of diseases, heavy losses were caused to the farmers. In absence of these details, fertilizers used in low volume than required will not be able to control disease spread and over usage of fertilizers impact the crop and soil quality (Tm et al., 2018). The frequent changes in climatic conditions coupled with crop sensitivity affected tomato crops during all the growth stages.

Early detection of tomato plant diseases, along with correct measurement of the degree of the disease spread, helps to control the disease, and save huge crop yield losses. The common symptoms of fungal, viral, and bacterial plant diseases are visible in the tomato plant leaves. The large population of small-scale farmer communities can be benefited from an automated plant disease diagnosis system that can recognize these visual changes in the plant foliage. The advances in computer vision and the application of ML & AI in the agriculture field are promising to detect plant diseases. Considering the growth and adoption of smartphones even in developing countries, a computer vision solution leveraging smartphones proposed a cost-effective solution with a wide reach. The previous research work primarily focused on training machine learning and deep learning model keeping the focus on improving the mode accuracy. However, most of these models were trained on image datasets from controlled fields which do not perform well in actual farm fields with real-world images. Further, there were separate models trained for the classification of different tomato crop diseases, and few separate models were trained to predict the degree of disease. Therefore, the primary objective of this research is to develop a combined neural network model to classify six types of common critical diseases with the prediction of the degree of disease. This can act as a decision tool to recommend accurate plant disease treatment. The model built as a part of this research will be deployed using an Android smartphone for real-world tomato crop disease detection.

1.2 Problem Statement

Plant diseases impact global agricultural yield by 20%-30% yearly. According to the National Bank for Agriculture and Rural Development (NABARD) survey of 2013, around 30% of crop yield (an estimated Rs. 60,000 crores) is lost every year in India due to crop diseases. So, there is a direct or indirect impact of such diseases on the Indian economy and population which depends on agriculture. This demands on-time and accurate plant disease detection.

In the traditional method, bare eye observation is carried out with a team of experts for disease identification, which is time-consuming and expensive. To address the inefficiencies of the traditional methods, automated image processing techniques can play a vital role in plant disease identification. It provides a robust solution that can perform automatic plant disease detection without requiring a team of experts. In the field of plant disease detection, digital images captured are being used by plant pathologists to conduct diagnosis, research, etc.

The raw images without any associated metadata result in additional time to perform disease diagnosis and impact the quality of diagnosis. Furthermore, some of the tomato plant diseases show similar symptoms, due to which farmers are unable to decide the correct disease type. In several previous instances, it was noticed farmers were unable to measure accurate disease spread. This potentially leads to incorrect treatment involving either misuse of fertilizers and fungicides or inappropriate quantities. It can lead to severe crop yield and quality losses and can also degrade soil fertility.

1.3 Aim and Objectives

The primary goal of this research is to come up with a practical solution to detect and classify six types of tomato crop diseases in the agricultural field (farm) in real-time. This research is focused on four main objectives considering real-world solution deployment.

- To identify six types of tomato diseases leveraging the deep Convolution Neural Network model.
- To improve the model performance with the Transfer Learning technique on real-world plant images (instead of images from a controlled (lab) environment).
- To identify CNN architecture with fewer parameters with equivalent accuracy, which is suitable for deployment on low compute devices (Smartphones, UAVs, and FPGAs).
- To predict the degree of crop disease spread, enabling decision-making for treatment of the disease.

1.4 Research Questions

This research work addresses three important research questions.

1. What role can a lighter Convolution Neural Network model play in detecting tomato crop diseases in the farm field, using commonly available compute devices such as Smartphones?
2. How is the performance of the lighter CNN model architecture compared with the existing state-of-the-art deep CNN networks?
3. Can Machine Learning techniques enable accurate prediction of the degree of tomato crop disease using extracted features from plant leaves, which will allow us to build a remediation solution?

1.5 Scope of the Study

The scope of this research work is to detect 6 types of commonly observed critical diseases in the Tomato plant using images of leaves. Crop disease symptoms can be detected through monitoring the stem, fruit, and even the root of the plant. However, plant leaves show early symptoms, which helps to decide the appropriate treatment to avoid losses to crop yield due to the spread of the disease. Hence, this research focuses on plant leaves only. Due to the lack of a sufficiently large number of annotated images from the actual plantation field, the research work considers only 6 types of tomato crop diseases for which enough data points are available to train and test neural networks. To detect the degree of disease, the research performs feature extraction and experiments with various types of features to ensure acceptable accuracy for a real-world solution. Hence, the scope is limited to only one or two types of diseases to predict disease spread (other disease types can be considered as the scope of future work). Lastly, the scope of model deployment is limited to the Android platform only (which makes sense considering the Android phone adoption rate among farmers who will be the end-users for the proposed solution) and deployment on iOS can be the future scope of work.

1.6 Significance of the Study

Considerable research work has been conducted in this area using machine learning and deep learning techniques on various crops. However, most of the work is based on image datasets from a controlled (lab) environment and not from an actual field (farm) which does not result in a good performance with real-life model deployment on Smartphones. The previous research work mainly

focused on improving classification model accuracy with a variety of deep learning and machine learning techniques and limited consideration for model deployment in real-world solutions which can help the end-users, i.e., farmers, to control crop yield losses due to widespread diseases. The primary contribution of this research work is an end-to-end real-world solution to – (A) detect 6 types of common tomato plant diseases leveraging real-life images; (B) showcase the potential solution to recommend accurate treatment by predicting the degree of tomato crop disease, and (C) evaluate simple and light CNN models using transfer learning methods to be deployed on smart devices.

1.7 Structure of the Study

The structure of the dissertation is as follows. Chapter 1 provides the background of Tomato Crop Disease Detection research work. The problem statement in Section 1.2 highlights the adverse impact of crop diseases and why timely detection of crop diseases is important to avoid major crop yield losses. The aim and objectives are called out in Section 1.3 which tries to address the research questions mentioned in Section 1.4. The details on the scope of the study along with details on what is not covered as a part of this research work are mentioned in Section 1.5. It shows how this research work intended to contribute to the existing body of knowledge. Section 1.6 highlights the significance of the research work in the field of tomato disease detection.

Chapter 2 starts with an explanation of various plant pathogens and commonly observed diseases that impact tomato plants. Section 2.3 discusses and compares the traditional and modern crop disease detection methods. Further, it provides details on the evolution of the computer vision field. It further provides statistical details on the exponential growth of the Smartphone market share and highlights the role of Android OS in emerging markets. Section 2.4 covers the details of publicly available datasets for training Deep Learning models to detect tomato crop diseases. The important details on previous related research work are covered in section 2.5. The following subsections review and compare various traditional and modern deep learning approaches, data pre-processing techniques, feature engineering aspects, and finally, model training and evaluation processes. Section 2.6 discusses the gaps identified based on the detailed literature review and highlights the importance of this research work to fulfill those research gaps.

Chapter 3 starts with an explanation of the research methodology which provides a structured process for this research work. Section 3.3 provides details on the dataset selection which is then followed by Section 3.4 on data processing techniques. The feature engineering process, which includes both automatic and manual feature extraction, is mentioned in section 3.5 and its subsections. Section 3.6 covers the model training process details for a simple and lighter CNN model which is suitable for deployment on Smartphone devices. A dedicated subsection on CNN with Transfer Learning highlights the challenges in training deep CNN models from scratch. It then explains the transfer learning approach followed by this research work to overcome those challenges. The model evaluation metrics for the image classifier CNN model are discussed along with the reasoning on why those specific evaluation metrics are considered. Finally, Section 3.7 explains the approach for deploying the trained CNN model on Android mobile to predict the tomato crop diseases in actual plantation/ farm fields.

Chapter 4 focuses on important aspects of the research, which include data exploration and analysis, data preparation, model training, and model evaluation. It begins with section 4.2 which contains the first subsection on data loading followed by a subsection on data cleansing and image annotations. The subsequent sections explain how the cropped image dataset and plant severity dataset are derived from the original PlantVillage and PlantDoc datasets. The following section covers image resizing and rescaling aspects to optimize the model training process. A subsection on image augmentation explains how image datasets are augmented using the Keras data processing APIs. Section 4.3 explains the details of high-level EDA for class distribution analysis followed by image EDA details leveraging the “basic-image-eda” tool. The section also provides details of interferences made and arrived at conclusions based on the analysis details. Next, section 4.4 covers the details of class balancing techniques and provides data descriptions for balanced datasets. Section 4.5 contains subsections that cover the CNN model designing and implementation steps in detail. The subsections cover the details on the process for selection of base models, the architecture of the CNN model, and hyperparameters tuning. Section 4.6 covers details of model evaluation and explains the process of final model selection for deployment. Next, section 4.7 covers the details on the creation of the TFLite model using Python APIs and integration of the TFLite model with the Android app. The process of designing and implementing the Android app is covered along with the details on inferencing flow. The final section 4.8 covers the approach for plant disease severity detection using global (color, shape, and texture) and local (ORB keypoint

and feature descriptors) features and by leveraging machine learning classification algorithms including Random Forest and Support Vector Machine. The section also covers the CNN model implementation details for plant disease severity detection.

Chapter 5 discusses the evaluation results obtained from multiple experiments carried out in this research study and the analysis of the results. Section 5.2 initially explains the overall evaluation process followed in this study and the classification metrics being used for the evaluation. The following eight subsections cover the evaluation of multiple CNN models against multiple variations of PlantVillage and PlantDoc datasets, which include original datasets, balanced datasets, and cropped image dataset. Each subsection focuses on individual SOA mobile CNN model architectures including MobileNet V1 & V2, EfficientNet B0, and NasNetMobile. For the MobileNet family of models, a separate subsection provides details of performance monitored with models trained using 100% parameters and 50% parameters (reduced model width). Section 5.2.8 finally compares the performance of the lighter CNN models with deep CNN models trained using ResNet50 V2 model architecture. Section 5.3 provides details of comparative model performance analysis of all CNN models. A table in subsection 5.3.1, which contains the details of CNN models' performance metrics trained with various SOA CNN base models on PlantDoc original dataset, provides a complete comparative analysis view. Section 5.3.2 compares different trained CNN models' sizes which are fine-tuned using the PlantDoc dataset. Next section 5.4 validates the final model performance against the PlantDoc and PlantVillage test datasets. The plant severity detection model details cover two approaches followed in this research study. It first explains the process of manual feature extraction including global (color, shape, and texture) and local (ORB features) image features. Then it evaluates the performance of classical ML models trained using these feature vectors which include Random Forest Classifier and SVM. The next subsection 4.8.2 covers the second approach of leveraging the CNN model to detect plant disease severity. It compares the MobileNet V2 default (100% parameters) model with a reduced model trained using 50% parameters. It also compares the performance of CNN models trained to detect plant disease severity with ML classification models trained in approach 1. The final section discusses the inferencing process flow using mobile phones. It provides the details about inferencing duration using SmartCropCNNLite mobile application.

Chapter 6 summarizes the entire research study process and explains important conclusions drawn from the analysis of the results obtained. Subsection 6.3 covers the details of the contribution of this study to the existing body of knowledge. Lastly, section 6.4 discusses the future recommendations which show the way for improving the model performance and enhancing the solution scope to Tomato Plant Disease Treatment Recommender System.

CHAPTER II:

LITERATURE REVIEW

2.1 Introduction

This chapter will contain the details of an existing literature review conducted as a part of this research work and summarise the analysis of referenced research papers, journals, and web articles. The section will begin with details on common plant pathogens and commonly observed Tomato plant diseases which are covered in this research work. It will be followed by an overview of various approaches and methods used for crop disease detection and how those evolved due to technological advancements. The next subsection will provide details about publicly available datasets in the field of plant pathology. The importance of real-world image datasets is then compared with a dataset from a controlled field (lab) environment. The next subsection will discuss the role of machine learning in plant pathology and various techniques used for feature extraction along with commonly used machine learning algorithms in this field. The following subsections will cover how the increased computational power and the availability of large datasets are helping to design and implement deep artificial neural networks (ANNs) with multiple layers. The following subsections will provide details on the role of computer vision and deep learning models such as CNN in the agriculture sector. It will be followed by a comparative analysis of various CNN architectures used for crop disease detection considering deployment on smartphones. The following subsection will discuss the various feature extraction algorithms, which extract features such as color, shape, and texture to measure the degree of disease speared. The last subsection will cover reference details on the deployment of a trained CNN model onto the Android mobile devices for detecting tomato crop diseases in the actual field.

2.2 Plant Pathogens and Tomato Plant Diseases

Plant diseases prevent the natural growth of plants, causing them not to perform to their maximum potential. There are mainly two types of plant diseases – (1) abiotic or non-infectious diseases and (2) biotic or infectious diseases ("Plant Disease: Pathogens and Cycles | CropWatch," 2021). Abiotic plant diseases are caused by non-living agents and conditions which are external to the plant. These diseases do not spread from one plant to another plant and are usually caused due to nutritional deficiencies, soil compaction, salt injury, ice, and sun scorch. Biotic or infectious diseases, which are also known as plant pathogens after affecting the plant are caused by living

organisms. Plant pathogens affect plant tissues of the various parts of the plant which include leaves, fruit, stems, crowns, roots, and vascular tissues. The common types of plant pathogens are fungi and fungal-like organisms, bacteria, phytoplasmas, viruses and viroids, nematodes, and parasitic higher plants.

Tomato crop diseases impact yield adversely and affect crop quality. Commonly observed diseases include wilts, leaf spots/blights, fruit spots, and rots. Most of these germs are fungi, with bacterial and viral infections whose symptoms vary according to the diseases (Janakiraman Aarthi, 2019). Fungal diseases are a type of plant pathogen that is very common in vegetables which impacts plant cells. The infected soil, seed, crop, and weeds are the main source of fungal diseases. It is spread by the wind and water and through the movement of contaminated soil, animals, workers, machinery, and tools. Early & Late Blight, Leaf Mold, and Septoria Leaf Spot are common tomato plant disease examples that are caused by fungus. In viral disease, a virus enters the plant through a lesion that affects the natural growth of the plant, causes mechanical injury to the plant or leaf, or causes the plant infections. Tomato Yellow Leaf Curl Virus (TYLCV) is an example of a viral disease. There are some definitive symptoms visible in the tomato plant leaves, which can help in the identification of the target pathogen. This research work focuses on the identification of six types of critical tomato diseases for which sufficient images from the farm field are available (or absence thereof).

Types of Tomato Diseases (covered in this research)

- Early Blight (*Alternaria solani*) is the most common disease caused by the fungi *Alternaria tomatophila* and *Alternaria solani* (Kluepfel et al., 2020). Early symptoms of the disease are small, brown lesions, mainly on the older foliage. Spots grow bigger and concentric rings in a bull's-eye pattern can be spotted in the middle of the infected regions. It affects the leaves, fruits, and stems of the tomato plant.
- Late Blight is a critical disease. The fungus *Phytophthora infestans* causes it (Kluepfel et al., 2020). It affects almost every part of the plant. Young leaf lesions are tiny and look like dark, water-soaked spots which can quickly turn into a white mold, appearing at the edges of the affected region, on the bottom layers of the leaves.
- Bacterial Spot occurs due to Xanthomonas Vesicatoria, which infects the plant, including leaves, stems, and fruits (Kluepfel et al., 2020). Several small, angular to irregularly shaped,

water-soaked spots on the leaves and slightly raised, scabby spots on the fruit are the common symptoms.

- Tomato Yellow Leaf Curl Virus (TYLCV) transmitted by whiteflies is immensely damaging to fruit yield. The symptoms are upward twine of leaves, yellowish (chlorotic) borders of the leaves, small-sized leaves, plant stunting, and flower drop (Kluepfel et al., 2020).
- Leaf Mold is caused by the fungus *Passalora fulva* which impacts older leaves of the plant close to the ground which lacks fresh air movement, resulting in increased humidity (Kluepfel et al., 2020). Initially, light green/ yellowish spots are seen on the upper surface of a leaf, which increases and turns into peculiar yellow.
- Septoria Leaf Spot is a rapidly spreading disease that adversely impacts foliage, petioles, and stems of the tomato plant. The fungus *Septoria lycopersici* is the cause of this disease (Kluepfel et al., 2020). Normally, this infection can be seen at the lower leaves near the soil, once plants start getting fruits. Several circular spots with dark edges covering a beige-colored center appear on the old leaves. Minute black spore-producing bodies are visible in the middle of the spots.

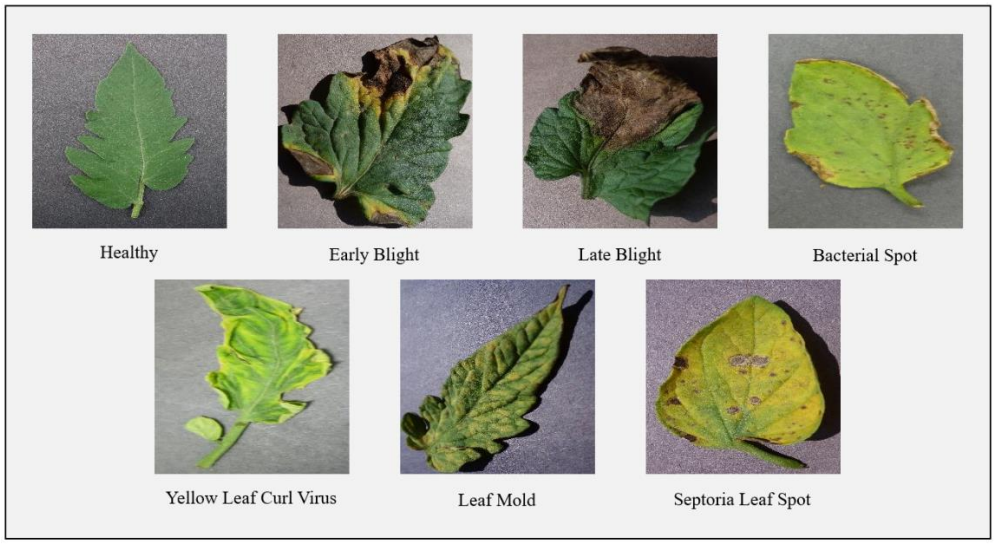


Figure 1 – Healthy and diseased tomato leaf samples (extracted from PlantVillage image dataset)

2.3 Crop Disease Detection Methods

The crop disease detection methods evolved from traditional manual and expert-based systems to more automated systems using Machine Learning, Deep Learning, IoT, and Computer Vision technologies in the last few decades.

2.3.1 Traditional methods and their limitations

To control crop yield losses due to plant diseases, a timely and careful diagnosis is required. There are many instances in the past when late identification of crop diseases resulted in difficulties to control the spread of diseases, which also increased the financial burden on the agricultural industry, especially small-scale farmers. The common methods used for crop disease detection include human experts performing visual analysis of affected areas of plants, evaluating morphology features using the microscopic method, as well as molecular, serological, and microbiological diagnostic techniques (Mahlein, 2016). The traditional methods involving human experts to identify crop diseases pose a lot of challenges due to a lack of a sufficient number of plant pathologists and their accessibility in remote locations. This clearly indicates a very high demand for a fully automated Computer-based expert system to detect crop diseases, which can be easily accessible even to small-scale farmers (who are the largest segment of the agriculture economy). By combining an automated computer-based crop detection system with aerial, satellite, and drone images, it is possible to cover large farm/ plantation fields which is difficult with the current manual verification process. This can support precision agriculture adoption by enabling identification of specific disease-affected farm areas, which can be focused to control the spread of plant diseases. This ultimately helps in better utilization of time, efforts, and financial resources to improve the overall crop yields via targeted crop disease detection and pest control management.

2.3.2 Computer Vision Evolution

Computer Vision (CV) evolved as a subset of Artificial Intelligence (AI) in the late 1960s, where researchers and scientists tried to mimic the human visionary system. It was believed that processing data from digital images to achieve a high-level understanding of it and unraveling symbolic data from the image data was an easy task – namely the "visual input" problem (Richard Szeliski, 2011). The complexity of transforming retina input into symbolic information was grasped by the researchers shortly afterward. With the evolution of the CV field in the 80s, researchers changed their focus on using mathematical models and techniques to analyze images,

like edge and contour detection. During this time, researchers noticed correlations between various algorithms in Computer Vision, and algorithms were unified to a higher extent. In the later 2000s, the domain of Machine Learning (ML) models for visual recognition emerged, and is currently dominating the field of CV.

The amount of labeled data, sophisticated algorithms, and the level of computational power are increasing day by day, enabling these ML models to categorize objects without the supervision of human beings. The MNIST dataset was commonly used to evaluate the performance of ML models for visual recognition. Currently, the most used algorithm for object detection in CV is CNN, which has surpassed human-level performance on image classification (He et al., 2015). The characteristics of Artificial Neural Networks (ANN) computations are like the characteristics of real-time graphics computations in video game rendering, where operations such as matrix multiplication and division are executed per pixel in parallel. Around 2005, researchers realized the potential advantages of performing ANN computations on Graphics Processing Unit (GPU) rather than Central Processing Unit (CPU), which resulted in higher performance and quicker computations. This allowed researchers to increase the number of layers in neural networks to create deeper ANNs and use more data while maintaining a reasonable execution time.

Image classification and image understanding are the most prominent subfields of computer vision and will evolve further in upcoming years due to computer hardware commoditization and robust software library support for these tasks. Today many personal laptops and desktops have GPU hardware acceleration along with multi-core CPUs, which can execute complex computational operations in considerably less time. There are many high-level software libraries such as Keras (with support of TensorFlow and Theano as backend), PyTorch (based on the Torch library), MXNet (specializes in distributed, multi-machine learning), and OpenCV (provides real-time image processing capabilities). Most of these libraries are written in C/C++ with Python bindings which are performance oriented. Also, many cloud providers (AWS, Azure, GCP, FloydHub, Google Collab) today provide GPU compute resources along with all required machine learning and deep learning libraries pre-configured with the “Pay-as-you-go” model which allows to quickly get started with minimal upfront CAPEX requirement.

This research work explores the probability of using a CV to find cost-efficient and scalable solutions for tomato plant disease detection. Training a large neural network may take a lot of time, but once the models are trained, they can perform image classification instantly, making them an ideal solution for consumer applications on smartphones. Computer Vision paved way for solving current agricultural problems using recent advancements in the field of Deep Learning. This allows agricultural improvement resulting in automated disease control and increased crop yields. Hence, this research leverage Computer Vision and Deep Convolutional Neural Network for automated tomato plant disease detection.

2.3.3 Smartphone current and future trends

As of date, the number of smartphone users surpasses the 3 billion marks, and it is predicted to further increase by several hundred million in the coming years. China, India, and the United States are leading this race with large numbers of smartphone users crossing the 100 million user mark. Out of the 7.7 billion worldwide population, there are 3.2 billion smartphone users globally indicating penetration of smartphone usage of 41.5% ("Statista | Smartphone penetration worldwide," 2024). Nowadays, high-resolution display screens, high-performance processors, and HD cameras are commonly available with most smartphones enabling building computer vision applications and AR apps. Android phones hold a 72.92% mobile market share keeping a worldwide leadership position in October 2020 (O'Dea, 2022). In developing countries (especially rural areas) Android phone adoption over iPhone (with iOS) is considerably higher due to cost consideration, which makes Android-based smartphones a choice of platform for this research work.

2.4 Plant Disease Detection Datasets

Real-world solution for plant disease detection needs to be validated with a quality dataset with appropriate volume. There are very few large public image datasets are available for plant disease detection. In the past, most of the research work gathered the plant images using digital and mobile cameras and annotated the images using either human experts or automated image labeling tools. However, those datasets are proprietary and not available in the public domain. In the next subsections, these public image datasets are analyzed for their suitability for this research study.

2.4.1 PlantVillage Dataset

The PlantVillage project carried out by Mohanty et al. (2016) resulted in the creation of a large plant leaves dataset of 14 crop species. The dataset contains 38 classes that span 26 plant diseases. The researchers trained a deep convolutional neural network to identify 14 crop species and 26 diseases using AlexNet (Krizhevsky et al., 2012) and GoogLeNet (Szegedy et al., 2015) architectures. The research work carried out involves training with 60 experimental configurations by combining CNN model architectures, which include AlexNet and GoogLeNet; two types of training mechanisms which are leveraging transfer learning and training CNN model from scratch; three types of dataset types which include color, greyscale, and leaf segmented; and five different choices of training and testing set distribution. The model trained using GoogLeNet architecture on color images dataset with 80%-20% train-test split proved to be the best performing model on PlantVillage dataset. The trained CNN model achieved an accuracy of 99.35% on a hold-out test set, which demonstrates the use of the PlantVillage dataset for crop disease detection.

The PlantVillage dataset encompasses photos captured under controlled settings which filtered the background noise. This dataset is not quite effective in the real world for plant images containing several leaves with varying background conditions and different types of lighting conditions.

2.4.2 PlantDoc Dataset

Singh et al., (2020) as a part of their research, curated real-world images of infected and non-infected crops downloaded from Google Images and Ecosia to create a PlantDoc dataset. The dataset contains 2,598 images spanning 27 classes covering 13 plant species. These images are captured from the actual field (plantation) instead of a controlled (lab) environment (which was the case of the PlantVillage dataset (Mohanty et al., 2016)) which impacts the model efficacy in real-world images classification scenarios. To prove the impact of the PlantDoc real-world dataset, the researchers trained 3 CNN models using the dataset and compared the results with PlantVillage Dataset. The research claimed that using the PlantDoc images dataset increased the classification accuracy by 31%. The researchers also created another cropped image dataset (Cropped-PlantDoc) using the bounding box information. This dataset contains only images of leaves like the PlantVillage Dataset; however, the images are low-quality with varying backgrounds which is the usual scenario in a farm field. The CNN models are trained using VGG 16, InceptionV3, and InceptionResNet V2 architectures and compared the classification accuracy and F1-Score. The

analysis confirms that when the CNN model is trained using images from a controlled field such as the Plant Village dataset and then tested with real-world images, the model performance is not up to the mark. To improve the CNN model performance, training needs to happen with real-world images from the actual farm/plantation field. The research work also included leaf detection using PlantDoc Dataset by leveraging the MobileNet model and an efficient R-CNN model based on InceptionResnetV2 architecture.

Though, the PlantDoc dataset has been curated by experts and images are labeled following appropriate process and guidelines, still, there are chances of some images classified wrongly due to the absence of extensive domain expertise. Furthermore, some of the classes do not contain a sufficiently large number of images which are required for accurate model training for disease detection. The PlantDoc dataset's usefulness can be further enhanced by applying proper image segmentation techniques to take out the leaf from the images to overcome the background noise in real-world plant images.

2.5 Related Research

Based on the existing literature review, it is evident that Crop Disease Detection is an area of interest of the research community for the past few decades. There are two main approaches for automated crop disease detection using Computer Vision, which include the traditional machine learning approach and modern deep learning approach, which are reviewed and compared in the next subsections.

2.5.1 Traditional ML/ DL modelling

The first traditional approach followed is based on image processing techniques such as segmentation, clustering, and filtering. It involves manual feature extraction using Python libraries such as OpenCV to find interesting regions to detect the changes in plant leaves due to disease. Once the various features of interest are extracted, including color, shape, and texture, the traditional ML models, such as K-Nearest Neighbour, SVM, and Logistic Regression, are used to classify the images in various types of disease classes.

In the past research, plant images were segmented into multiple, non-overlapping regions using edge detection techniques, region extraction, and clustering. The image segmentation groups the

different objects of an image with the same color and texture. The image segmentation was performed, leveraging a genetic algorithm, which resulted in the detection of various plant diseases for fruits and vegetable crops in the early stage of disease with less computational resources (Arya et al., 2018; Singh et al., 2015; Singh and Misra, 2017). The research highlighted that genetic algorithm-based segmentation clubbed with Bayes classifier, ANN classifier, Fuzzy Logic, and hybrid algorithms increased classification efficiency to detect plant diseases. Al Bashish et al. (2011) performed image segmentation leveraging K-Means clustering and classification using neural network and detected five types of plant diseases. The research used RGB images of plant leaves to create a color transformation structure that is segmented using K-Means clustering. The texture features were extracted to detect infected parts of the leaf and then passed to the neural network classifier. Though, a model trained using K-means clustering is guaranteed to converge, the solution quality is largely dependent on the choice of the value of K and the initial set of clusters. K-Nearest Neighbor (KNN) classifier was used after image segmentation and color and texture feature extraction for various plant types of disease classification (Hossain et al., 2019; Vaishnnave and Devi, 2019). However, K-Nearest Neighbor (KNN) classifier is computationally expensive and is not suitable for large datasets. The research from Islam et al. (2017); Meena Prakash et al. (2018); Padol and Yadav (2016) leveraged the SVM classifier after segmenting the images using K-Means Clustering. SVM helped to train robust classification models even on datasets with high bias. In the past research work, a common practice adopted was to try out different classifier models such as KNN, SVM, Naive Bayes, and Decision Tree. The overall classification model performance using the traditional machine learning is directly dependent on how well feature engineering techniques were adopted to extract and select features for plant disease detection.

With the advancements in the Deep Learning field to classify plant diseases, multiple researchers leveraged deep neural networks with multiple hidden layers and GPU-based compute for parallel processing. Multilayer Perceptron (MLP) using back-propagation algorithm was used to train the classification models to detect various plant diseases. These deep learning models used similar approaches for image segmentation such as K-means clustering and extracted color, shape, and texture features using HOG, GLCM, SIFT, SURF, and so on. After feature extraction and selection, feature vectors are passed to the Back Propagation Neural Networks (BPNN) classifier instead of traditional Machine Learning classifier models such as SVM, Naive Bias, and KNN (Dhakate and

B, 2015; Ramakrishnan and Sahaya, 2015). BPNN requires more training time as compared to ML models, however, the model accuracy is higher for these models when an appropriate number of layers and neurons are used in hidden layers. Khan and Narvekar (2019) used Multiclass Support Vector Machine (MSVM) classifier and neural networks to classify the tomato plant diseases using extracted hybrid features. The research can be further enhanced by using CNN to extract the features in an automated way and compare the impact on classifier performance. With the objective of detecting and classifying tomato and cotton leaf diseases, Kumari et al. (2019) focused on their research work to perform image segmentation using k-means clustering. They extracted the features such as Contrast, Correlation, Energy, Homogeneity, Mean, Standard Deviation, and Variance to detect tomato and crop diseases. The research used a neural network to classify the images in four categories which include Bacterial Leaf Spot & Target Spot for cotton leaves and Septoria Leaf Spot & Leaf Mold for tomato leaves. This research work is carried out with very limited number of image datasets which includes 20 images for tomato diseased leaves and 20 images for cotton plant spanning across four disease classes. Hence, the higher accuracy numbers need to be investigated for overfitting scenario.

2.5.2 Data Acquisition and Processing

In the past, researchers used multiple methods to acquire an image dataset with enough data points and then annotated the image dataset using manual and automated techniques. The methods include capturing the plant images from mobile or digital cameras and downloading the images from the internet by searching for specific plant disease types. Labeling and cleaning of image datasets for various disease types require domain expertise in Plant Pathology and requires a considerable amount of time and effort. Due to this many research studies in the past relied on the PlantVillage dataset which contains 54,306 images of 14 crops infested with 26 diseases (Mohanty et al., 2016). However, this dataset was curated using images taken from a controlled (lab) environment. After loading the data, multiple pre-processing techniques were used to normalize, center, and standardize the image dataset.

The research work by Wang et al. (2017) used the healthy and black rot apple images from the PlantVillage dataset and botanists further annotated the black rot apple leaf images into four disease severity classes which included healthy, early, middle, and end stages. The images were rescaled

as per the requirements of each CNN architecture for transfer learning and to 256 X 256 pixels for a shallow network. Normalization is applied to achieve standard normal distribution data which improved the model training efficiency. To prevent overfitting, various image augmentation techniques were applied such as flipping, shearing, zooming, and randomly rotating images. In the previous research work from Owomugisha and Mwebaze (2017), National Crops Resources Research Institute (NaCRRI) collected the cassava leaf images using smartphones which were then manually labeled by experts. The dataset included 7,386 cassava plant leaves images which were divided into 5 classes including healthy leaf images and 4 types of diseases. Durmus et al. (2017) used PlantVillage dataset by taking a subset of the only tomato leaves images for the research work to evaluate suitable lighter CNN models for deployment on mobile devices. The data set considered had 10 different types of tomato image classes which included healthy and 9 types of tomato diseases. Sladojevic et al. (2016) collected the images from the internet by searching with specific disease names for different types of plants in different languages. The research work also considered creating a new class with only background images using the Stanford background dataset (Gould et al., 2009) to increase the classifier accuracy. After cropping the images to get the interesting plant leaf region, the researchers resized the resulted image dataset to 256 X 256 pixels using Python script and OpenCV framework. Then images were augmented using simple image rotations and various transformation techniques, such as affine transformation and perspective transformation. The research carried out by Rqjodl et al. (2018) used the 54,306 images from PlantVillage dataset spanning across 38 categories. The images were resized to 64 × 64 pixels for model training and making predictions. The pre-processing was only limited to scaling images to the range of the tank activation function [-1, 1]. The research done by Ramcharan et al. (2017) gathered several cassava genotypes and stages of maturity plant images from experimental fields which contained 2,756 images after data set cleaning. The researchers then manually cropped original images into individual leaflets to build the second dataset with15,000 images of cassava leaflets (containing 2,500 images per class). The images were then labeled into six classes including a healthy class and 5 types of Cassava disease classes. To demonstrate the feasibility of leveraging deep CNN model to classify five tomato plant diseases, Ashqar and Abu-Naser (2018) experimented with 2 types of models built using 9000 images of infected and healthy Tomato leaves collected under controlled conditions from public data set. They resized the images to 150×150 pixels without compromising on image quality.

2.5.3 Important Features and Feature Extraction Techniques

The previous research studies leveraged multiple feature extraction techniques to extract various features such as color, texture, shape, and spot size details. After extracting the features, any redundant features need to be discarded to avoid data redundancy. Only those features need to be selected which contribute to model prediction capability. The previous study has shown that usually hybrid features such as a combination of color, shape, and texture are efficient to detect multiple patterns in the images for plant disease detection.

In the research work done by Owomugisha and Mwebaze (2017) 4 types of cassava plant diseases are classified based on features extracted using the ORB algorithm (Rublee et al., 2011) and color features. As Cassava plant diseases can be noticed easily using shape and color deformations on the plant leaves, the research extracted those representative features from the input images. Intending to leverage Open Source and freely available feature extraction algorithms, this research leveraged the color features and the Oriented FAST and Rotated BRIEF (ORB). To extract color features, the research performed the HSV color transformation of the image and calculated the normalized hue histogram of the image using 50 bins. The ORB features provided the best combinations of two algorithms which included Features from Accelerated Segment Test (FAST) keypoint detector algorithm and Binary Robust Independent Elementary Features (BRIEF) feature descriptor. The ORB algorithm solves the computation of orientations problem of the FAST algorithm and the major drawback of poor performance on rotation with BRIEF. The ORB identifies interest key points in the image. The key points are scattered throughout the image with most of the points centered on the section of the leaf which is damaged. Each point has 32 vectors that describe the key point at that location uniquely. To get a uniform representative feature vector of the image, the research applied the bag-of-visual words technique to form 120 clusters with different key points to represent the image. This forms a dictionary that is used to train uniquely for each disease class. To represent a new image using ORB features, keypoint descriptors were extracted from the image and then mapped to the cluster centers in the dictionary. The research created two extracted datasets containing a 7386×50 dataset representing color hue histograms and a 7386×120 dataset representing the generated ORB feature vectors.

The survey conducted by Khan and Narvekar (2019b) reviewed and compared the various feature extraction techniques. They reviewed Gray Level Co-Occurrence Matrix (GLCM) which includes texture-based features such as entropy, energy, contrast, and homogeneity statistical measures. The research then reviewed color-based features using color histograms which converted images from RGB color spaces to HSV and YCbCr color spaces. To represent local structures and object contours, the research work included the Histograms of Oriented Gradient (HOG) algorithm which is a shape-based feature extraction technique. Finally, the research focused on automated feature extraction using deep learning techniques. The research work also recommended a hybrid feature set for tomato leaf disease detection, which included color information extracted from the color histogram, Local Binary Pattern (LBP) as a texture descriptor, and Zernike Moments (ZM) as a shape descriptor.

2.5.4 CNN Training (from Scratch)

The approach of the deep learning neural networks, to automatically extract various features and then perform image classification, is gaining more popularity due to its high accuracy rate. The most common neural network architectures are Convolutional Neural Networks which are very effective in object detection and classification tasks for image datasets.

The research performed by Sardogan (2018) focused on detecting 5 types of tomato diseases using CNN with the Learning Vector Quantization (LVQ) algorithm wherein they extracted 500 feature vectors from input images. The research used the CNN model for automatic feature extraction and LVQ is used for classifying the tomato leaf images into five classes. LVQ combines competitive learning with supervised learning, resulting in a simple and adaptive model which can classify a fixed number of classes efficiently (Mokbel et al., 2015).

The research work by Wang et al. (2017) focused on diagnosing the apple plant leaf black rot disease severity using a deep convolutional neural network. The networks were trained from scratch by researchers with different numbers of convolutional layers (2, 4, 6, 8, & 10). The network used 32 filters of 3 X 3 size with ReLU activation function in convolution layer followed by 2 X 2 max pooling layer, two fully connected layers, and finally a SoftMax layer. The shallow network model used a dropout ratio of 50% to avoid overfitting the model. The different CNN architectures had shown impressive results in image classification and object detection tasks.

Sladojevic et al. (2016) built a CNN model to detect 13 types of plant diseases. The research leveraged CaffeNet architecture by adjusting the SoftMax layer to support 15 classes (13 plant disease classes and a healthy and background images class each). The model was built by sequencing convolutional layers with ReLU activation function, max pooling layers, fully connected output layers, and finally SoftMax as a classifier. To fine-tune, the model and control overfitting, the research used lower learning rates in later layers. Agarwal (2019) in their research paper proposed a convolution neural network model in which they stacked 3 convolution layers, then 3 max pooling layers providing inputs to 2 completely connected layers to identify a disease for the corn crop. They trained the model on a publicly available PlantVillage Dataset (Mohanty et al., 2016). The research work compared the proposed CNN model with the traditional machine learning models such as Naïve Bayes, Decision Tree, Random Forest, Logistic Regression, k-NN by extracting Hu-moments, Harlick, LBP-histogram, and HSV-histogram features manually.

To demonstrate the feasibility of leveraging the deep CNN model to classify five tomato plant diseases, Ashqar and Abu-Naser (2018) created two types of models. The first model considered images with 3 color channels (RGB) and the second one with a single Gray-Scale channel. They considered four convolutional layers with ReLU activation function followed by max pooling layer in the future extraction layer for both the models. For both types of models, in the flatten layer, two dense layers were added. However, they considered different numbers of hidden units for color (256) and grey-scale (128) models resulting in almost twice the number of trainable parameters for the full-color model. In the classification layer, the model used the SoftMax activation function to classify the final output into 6 different types of classes. The full-color model outperformed in terms of classification accuracy as compared to the grey-scale model with very high accuracy of 99.84%. As the model is trained from scratch by using images from a controlled environment, there is scope for future enhancements to optimize the model training time using transfer learning and validate the model with real-world images.

2.5.5 Transfer Learning

In recent years, multiple research studies used a transfer learning approach to train the CNN models to optimize the training time. For plant disease detection, the transfer learning process provided a huge benefit due to leveraging state-of-the-art CNN models on large image datasets like ImageNet and COCO. The CNN models trained with the transfer learning approach in past studies resulted

in the highest model performance as compared to deep learning or machine learning model trained from scratch.

For transfer learning, research performed by Wang et al. (2017) used VGGNet, Inception-v3, and ResNet50 architectures as base models. The research relied on ImageNet trained weights for training new transfer learning models by freezing initial all layers except the fully connected layers. Further, the top convolutional block for VGG16 and VGG19, the top two inception blocks for Inception-v3, and the top residual block for ResNet50 were unfrozen to train a new fully connected network with a small learning rate. A trained CNN model using the transfer learning approach by Ramcharan et al. (2017) as a part of their research used the existing weights from the ImageNet database to retrain the Inception v3 CNN model to classify the cassava image datasets. During model training, the final layer of the CNN model Inception v3 was retrained for the cassava image datasets with 3 different architectures, which involved using the original inception SoftMax layer, Support Vector Machines (SVM), and K-nearest Neighbor (K-NN). Based on the model validation results, the SVM model had the highest prediction accuracies for four disease classes. The best model resulted in overall 93% validation accuracy on the Cassava images dataset. The results of this study showed that image classification with transfer learning from the CNN Inception v3 model can be used in the field for accurate automated cassava disease detection. The usage of SVM and K-NN in final classification layer also justified that transfer learning can apply traditional machine learning classifiers for retraining the vectors produced by the trained CNN model on new class data. The proposed CNN model with transfer learning, however, needs to be enhanced with larger image dataset to improve model performance and to expand for other crop disease detection.

The research work carried out by Suryawati et al. (2018) compared the performance of the base CNN model, AlexNet, VGGNet, and GoogLeNet architectures. The research analyzed the impact of the depth level of CNN architecture on the model performance. The research leveraged a subset of the publicly available PlantVillage dataset which contained 18,160 tomato leaves images covering 10 different class labels. Before using the images for the training model, those were resized into 64 x64 pixels and RGB values were extracted from the image. The base CNN model had 2 convolutional layers and 2 fully connected layers. Max Pooling was applied to the second convolutional layer and dropout was applied to the first convolutional and fully connected layers. AlexNet CNN architecture proposed by Krizhevsky et al. (2012) used 5 convolutional layers and

3 fully connected layers. AlexNet used the ReLU activation function in each convolutional layer along with max pooling applied in the first, second, and fifth layers. AlexNet used 5 convolutional layers with varying filter sizes of the 11x11 filter, the 5x5 filter, and the 3x3 filter. To overcome the problem of model overfitting, they have applied dropout and data augmentation techniques. Figure 2 – AlexNet Architecture from the AlexNet research paper shows the architecture of the AlexNet.

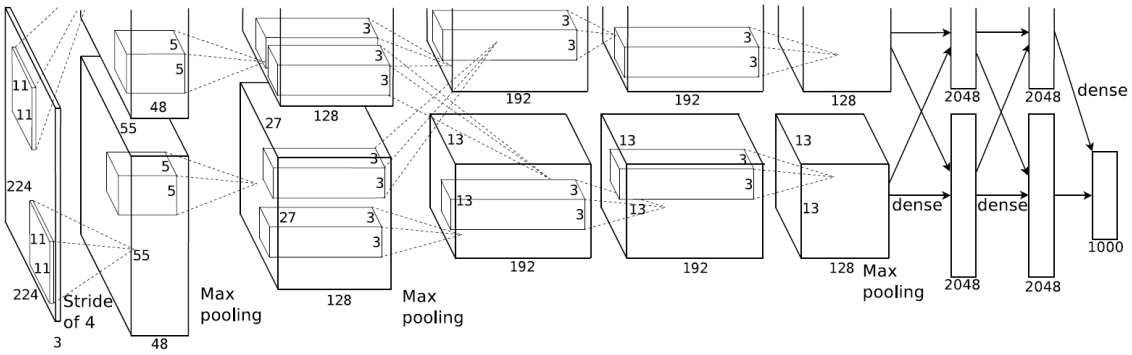


Figure 2 – AlexNet Architecture (Krizhevsky et al., 2012)

The research used the same AlexNet model architecture with the PlantVillage dataset which was originally trained on the ImageNet database as a part of the ILSVRC-2012 competition. Then the study leveraged the VGGNet architecture for training another model which surpassed the performance of AlexNet through increased network depth and reduced filter sizes of 3x3-sized filters as compared to AlexNet filter sizes of 11x11, 5x5, and 3x3. The VGGNet 16 architecture contains a total of 13 convolutional layers with 5 max pooling layers of 2x2 filters with stride 2 placed after convolutional layers processing. It contains three fully connected layers followed by the SoftMax layer as shown in Figure 3 – VGGNet Architecture.

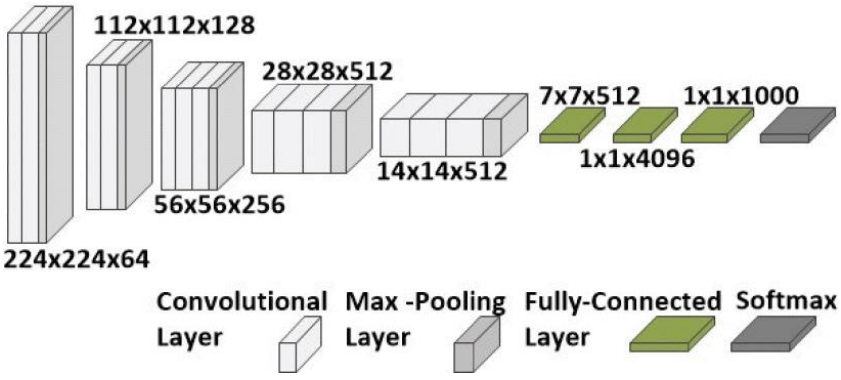
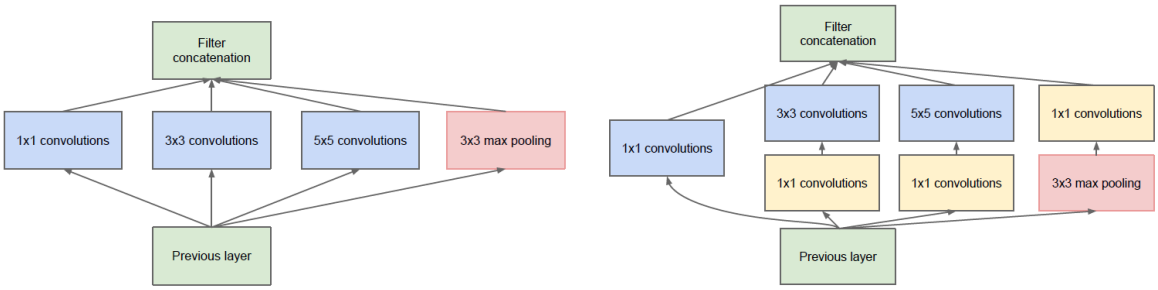


Figure 3 – VGGNet Architecture (Suryawati et al., 2018)

Finally, the researchers tried to build a model for Tomato plant disease detection leveraging GoogLeNet's (Szegedy et al., 2015) model architecture which won the 2014 ILSVRC competition by bringing down the training error rate to 6.67% on the ImageNet dataset. The distinctive factor of GoogLeNet model architecture as compared to previous state-of-the-art CNN architectures such as AlexNet and VGGNet is to explore both depth and width factors of CNN network to learn a better representation of image data. GoogLeNet solved the problem of determining the correct kernel size to perform the convolution process according to the location information (locally or globally) as per the image size variations by introducing the concept of the inception module. The inception module reduced the number of input channels and added the 1x1 convolution layer before the 3x3 and 5x5 convolution layers and after the 3x3 max-pooling layer on each inception module. The inception module supported parallel convolutions by using multiple filters of sizes 1x1, 3x3, and 5x5. GoogLeNet architecture supported low computational devices with the help of a reduced number of network parameters as it reduced the number of parameters from 60 million (AlexNet) to 4 million. Figure 4 – Inception module shows the naive version of the inception module along with the inception module with dimension reduction.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 4 – Inception module (Szegedy et al., 2015)

The research work carried out by Tm et al. (2018) aimed to build a CNN model using a simple approach that leverages minimal computing resources to achieve results that were comparable with the state-of-the-art (SOA) techniques. The researchers experimented with a slight variation of the convolutional neural network model called LeNet to detect and classify 10 tomato leaf diseases. The research used a subset of the Plant Village public dataset which included around 18160 images of tomato leaf diseases spanning 10 different diseases, and healthy leaves. As a part of data pre-processing the images are downsized to 60 X 60 pixels and image augmentation techniques such as random image rotation, horizontal and vertical flipping, and shifting of images with a small

degree was applied. The images were also normalized to bring all the pixel values in the same range by using the mean and the standard deviation (Z-score). As a slight variation to the original LeNet architecture, the model was consisting of an additional block of convolutional, activation, and pooling layers. The filters of size 5X5 were gradually increased in the three convolutional blocks stacked sequentially to compensate for the reduction in the size of the feature maps due to max-pooling layers. The features map was zero-padded to maintain the same image size. Dropout regularization was applied to avoid the overfitting scenario. The researchers used the ReLU activation function in the convolution layer and the SoftMax activation function in the final classification layer to classify the images in 10 different classes.

2.5.6 Light CNN models (for "Low Compute" devices)

For computer vision applications on mobile apps, the most important factors are the smaller size of the model and quick inference time which drive the usability, performance, and future updates over the wire. To that end, some interesting research work has been conducted to implement small but performant neural nets. The two proven approaches for implementing suitable network models for mobile devices and embedded computer vision applications are either to train smaller networks with a lesser number of parameters or to compress already trained large models using various techniques.

In recent years, there is a focus on building deep learning models, which can be deployed on smartphones and used in the plantation field for timely detection of diseases. The research done by Majumder and Shirvaikar (2020) focused on training the transfer learning model leveraging ResNet50 (Yun, 2015) architecture to perform maize crop disease detection in real-time in the field using the smartphone app. They trained their model on a subset of the PlantVillage dataset (Mohanty et al., 2016) and tested it on the Northern Corn Leaf Blight dataset (Hanks et al., 2018), and deployed it on an iOS device for real-time crop disease detection in an actual field. The research leveraged the pre-trained model on the ImageNet dataset (Jia Deng et al., 2009) which is a benchmark computer vision dataset.

The research carried out by Iandola et al. (2016) focused on the design-space exploration of convolutional neural networks using microarchitecture and microarchitecture patterns to compress the CNN model. They have proposed new CNN Architecture called SqueezeNet which has 50

times lesser parameters than AlexNet, however, it still maintains the same accuracy on the ImageNet dataset. Further, it experiments with compressing the SqueezeNet model to less than 0.5 MB size (510 times smaller than AlexNet) which makes it ideal for deployment on devices with limited computing resources such as smartphones and FPGAs. The proposed smaller CNN architecture provides many advantages such as less communication across servers during distributed training, efficiency in providing frequent updates over-the-wire connected devices and suitability of model deployment on FPGAs or devices with low computational hardware. The SqueezeNet model is trained and evaluated with an ImageNet dataset. The research applied three main architectural design strategies which include – (1) replacement of 3X3 filters with 1X1 filters; (2) decrement of the input channel numbers to 3X3 filters; and (3) ensuring large activation maps downsampling. The research introduced the Fire module which is comprised of a "squeeze" convolutional layer with 1 X 1 filters feeding into an "expand" layer which has a mix of 1 X 1 and 3 X 3 convolutional filters. The research work also explored three different types of microarchitectures of SqueezeNet, which included – simple SqueezeNet architecture, SqueezeNet with simple bypass connections between some Fire modules, and SqueezeNet with complex bypass connections between the remaining Fire modules. Both simple and complex bypass microarchitectures showed improvements in accuracy compared to simple vanilla SqueezeNet architecture while simple bypass microarchitecture resulted in the best accuracy. As a future research enhancement, the SqueezeNet model needs to be applied for other problem areas and evaluated against other than the ImageNet data set. To detect tomato plant diseases in farm fields or greenhouses in real-time using robots, Durmus et al. (2017) compared two deep learning model performances with a focus on the suitability of the model to run on mobile devices. The research analyzed the possibility of detecting tomato plant diseases in the field by training the model on limited compute resources using a dataset from a controlled environment. The research work indicated that SqueezeNet is a potential candidate for the mobile deep learning classification because of its lightweight and low computational requirements. The research leveraged SqueezeNet and AlexNet to detect the diseases. AlexNet has 5 convolutional layers. After every convolution layer, there is a ReLU layer. The addition of Normalization layers helped in the generalization of the model. Features at the 5th convolutional layer were fed to a fully connected network after pooling. The fully connected layers were responsible for calculating the class probability. To control overfitting, a dropout layer was added after the fully connected layers. The

SoftMax classifier finally classified the input images into 10 different classes. Another model leveraged SqueezeNet architecture for reducing the model size using 3 design strategies. Those strategies were - reducing filter size, decreasing the input channels, and down sampling late in the network. The proposed neural network started with a convolutional layer, then having eight fire modules, followed by the next convolutional layer, and lastly, the SoftMax classifier.

The research paper on MobileNets (Howard and Wang, 2012) proposed a new model architecture based on depth-wise separable convolutions to evaluate a class of efficient models for embedded vision and mobile applications. The study demonstrated how effective MobileNets are in various use cases (like object detection, fine-grain classification, face attributes, and large-scale geo-localization) and applications. The study also compared the performance of MobileNets with existing state-of-the-art transfer learning algorithms. Research showed a novel way to implement faster and smaller CNN architecture by tunning width and resolution multiplier hyper-parameters with equivalent model accuracy of state-of-the-art deep neural network. The research used various popular image datasets, which included ImageNet, Stanford Dogs, Im2GPS, PlaNet, Yfcc100m, and COCO. The study used depthwise convolutions for the application of a single filter for every input channel (input depth) and then pointwise (simple 1 X 1) convolution for creating a linear combination of the depthwise layer's output. The proposed MobileNets architecture used ReLU and batch norm nonlinearities for both layers and SoftMax in the final output layer. MobileNet has 28 layers considering depthwise and pointwise convolutions as separate layers. The training of the MobileNet models was done in TensorFlow with the help of RMSprop using asynchronous gradient descent, which is like Inception V3. As smaller models have fewer problems of overfitting, the model used less regularization and data augmentation techniques. The research introduced a simple width multiplier to make a network uniformly thin at every layer. Width multiplier effects reduce the number of parameters and the computational cost quadratically. It can be applied to any model structure for defining a new, smaller model having reasonable latency, accuracy, and size trade-off. The 2nd hyper-parameter in the reduction of a neural network's computational cost was the resolution multiplier. The researchers applied this to the input image and the internal representation of every layer is subsequently reduced by the same multiplier. The CNN model based on MobileNet architecture was adapted in research carried by Elhassouny and Smarandache (2019) to diagnose ten types of Tomato leaf diseases. The model implemented based on MobileNet architecture was able to detect the tomato crop diseases using popular mobile phones and

leveraging the mobile phone camera with acceptable performance. The research used a dataset containing 7176 tomato leaves images by resizing those to 224 X 224 pixels to train the model. The model consists of a series of Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batch norm (BN) and ReLU. One layer corresponds to 3 X 3 Depth wise Conv, BN, 1 X 1 Convolution, BN, and ReLU activation function. To get the classification of diseases, the model was ended by average Pooling of pooling layers, fully connected layers, and the SoftMax function with 10 classes. The proposed CNN Model has been tested with many optimization algorithms such as Stochastic gradient descent, adadelta optimizer, adagrad, adagradDA, Momentum, Adam, Ftrl, proximaladagrad, and RMSprop optimizers. Hyperparameter tuning is performed on the model to improve the model performance such as adjustments in learning rate. The accuracy was noticed between 85% to 90% for the various models with different optimizers and learning rates. The model can be further improved with large-quality image data set and newer CNN architectures targeted for low compute devices.

2.5.7 Prediction of Spread of Disease

For crop disease management and control, it is important to know the degree of disease spread to recommend appropriate solutions such as pesticides or rotational crop patterns. There have been instances previously when due to inadequate knowledge or misinterpretation of the intensity of the plant diseases, over-dosage or under-dosage of the pesticide has resulted in severe crop damages. Traditionally, trained experts used to examine plant tissues to detect the plant disease severity manually. This manual assessment method for measuring plant disease severity was expensive and not efficient due to a lack of expert support system which blocked the rapid modernization of the agriculture field to a certain extent (Mutka and Bart, 2015).

The research carried out by Owomugisha and Mwebaze (2017) classified disease severity levels on a scale of 1 to 5, where indicating 1 being a healthy cassava plant and 5 being a critically diseased cassava plant. As a part of the research work, the android application is built based on a client-server architecture that captured the images using smartphones and uploaded them to the server for classification of cassava disease types and severity. The research showed a high accuracy level when the Linear SVC classification model with a cross-validation technique on ORB feature vectors was used to predict the severity of the disease. The deployed model on a smartphone as a part of this research requires a smartphone to have a live internet connection which can be a

constraint sometimes in remote locations. As a future enhancement, there can be a compact version of the model which can work in offline mode to perform preliminary diagnosis and then be backed by further detailed diagnosis once the mobile phone is connected to the internet.

Rqjodl et al. (2018) proposed a Deep Convolutional Generative Adversarial Networks (DCGANs) extract discriminative features automatically from plant images using an end-to-end pipeline to predict the disease severity. DCGANs (Goodfellow et al., 2014) contain two networks - a generator that generates an image from a noise vector, and a discriminator which discriminates between a generated image and a real image. After the network is trained, the generator can generate an image from random noise that looks “real” which the discriminator should detect as “fake.” The research leveraged Adam optimizer with a learning rate of 0.0002. The researchers initialized the weights from a zero-centered Normal distribution with a standard deviation of 0.02. The research compared the quality of the representations learned by DCGANs for supervised tasks and then using the discriminator's convolutional features within all layers. The trained model was able to achieve an accuracy of 89.83%. The model performance can be enhanced using a large image dataset from the real world and trying out different hyperparameter tuning.

2.5.8 Model Evaluation and Testing

The previous research work used different model evaluation techniques. The researchers used various evaluation metrics depending on dataset imbalance percentage and type of classification model such as binary or multi-class classifier. The research work used different data splitting techniques such as hold-out and K-fold cross-validation. The trained models targeted for low compute devices, such as smartphones, were evaluated for latency, size, and power consumption.

The research from Sladojevic et al. (2016) used 10-fold cross-validation to verify model performance. The research work resulted in a precision score of 91% to 98% for separate class tests with an overall training accuracy of 96.3%. Based on the model validation results it was concluded that model fine-tuning did not impact the overall accuracy much, however, the augmentation process helped to improve the model performance a lot. The proposed model can be enhanced using large image data set covering wider land areas wherein images are captured using aerial photographs of farm fields by drones and a CNN model for object detection. The research performed by Sardogan (2018) was able to average an accuracy rate of 86% across 5 classes, which

can be further improved by training a model with different filters and convolutions sizes. The research carried by Agarwal (2019) used various metrics including accuracy, specificity, F1-Score, and Cohen-Kappa to compare the model performances. Though their model performance is not equivalent to the accuracy level of VGG 16, it scores more in terms of overall model size and image classification time needed which are important factors to build lighter models. The proposed model needs to be compressed by maintaining the accuracy and inference time for deployment of a model on devices with lower computational power such as mobile phones. The two-class image classification model trained by Majumder and Shirvaikar (2020) resulted in 0.99 accuracy and F1-Score. However, the model could classify the images only into two classes i.e., Healthy and Northern Corn Leaf Blight, and misclassified the other diseases such as Common Rust disease. The second part of the research focused on model deployment on iOS which though limits the applicability in rural areas of developing countries, as most of the farmers have android phones due to price advantage. Tm et al. (2018) verified various quantitative metrics such as accuracy, precision, recall, and F1-score to validate model performance. The training accuracy of 99.3% and validation accuracy of 94.8% was reported for the model. As a part of future enhancements of the model, different learning rates and optimizers can be experimented along with the newer architectures of CNN to improve the model performance. The research conducted by Wang et al. (2017) resulted in 79.3% test accuracy for the CNN model trained from scratch and 80.0% to 90.4% of test accuracy for models trained using the transfer learning approach (with VGG16 resulting in 90.4% test accuracy). The results of the research work proved that transfer learning helps to overcome the problem of insufficient training data. Durmus et al. (2017) did test of their trained models using the validation set was done on Nvidia Jetson TxI board which is used for robotic systems. Based on the accuracy-test results, the performance level of SqueezeNet was slightly inferior to AlexNet. According to the study, lowered accuracy compared to PlantVillage research work (Mohanty et al., 2016) was mainly caused due to smaller batch size of 20 which was selected due to less RAM size. The proposed model based on SqueezeNet architecture was almost 80 times smaller than AlexNet making it suitable to deploy on the low computational hardware of real-time inferences in the field or greenhouse.

In the research conducted by Suryawati et al. (2018), all the CNN architectures resulted in an accuracy rate above 89%. However, VGGNet outperforms the other CNN models by achieving a

95.24 % accuracy rate with a loss of 0.17. Table 1 shows the comparative analysis of CNN model performance.

Table 1 - Accuracy and Loss Metrics (Suryawati et al., 2018)

Architecture	Accuracy (%)	Loss
Baseline	84.58	0.47
AlexNet	91.52	0.51
GoogleNet	89.68	0.30
VGGNet	**95.24**	**0.17**

The results of the study suggested that CNN with deeper architecture such as VGGNet outperformed CNN base model, AlexNet, and surprising GoogLeNet as well for plant disease classification task with PlantVillage dataset.

2.6 Discussion

This research project plans to leverage the PlantVillage dataset due to its large number of image availability and the PlantDoc dataset to evaluate model performance on real-world images from the actual plantation field. It is evident based on the literature review, that the ORB features provide the best combinations of two algorithms which included Features from Accelerated Segment Test (FAST) keypoint detector algorithm and Binary Robust Independent Elementary Features (BRIEF) feature descriptor. Hence, this research will be extracting Color and ORB features during feature extraction. The study will be using various data pre-processing techniques which include normalization, centering, and standardization of image data. Based on a comparative analysis of traditional ML models and CNN, it justifies the use of CNN with transfer learning to achieve the objective of this study.

In the past research from “The Evolution Of Mobile CNN Architectures | Weights & Biases” (2020), a variety of smaller CNN model architecture performance was compared, that includes EfficientNet-B0, MobileNet, NasNetMobile, GhostNet, MobileNetV2 as shown in Figure 5.

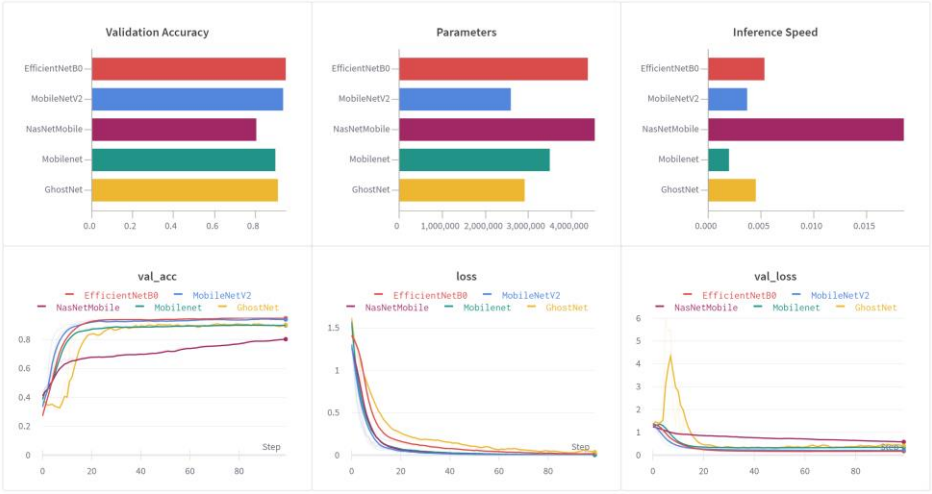


Figure 5 – CNN models analysis ("The Evolution Of Mobile CNN Architectures | Weights & Biases," 2020)

Based on previous CNN model comparative performance analysis, this research work will be leveraging MobileNet V1, V2, EfficientNet-B0, and NasNetMobile as backbone CNN models for Transfer Learning.

2.7 Summary

This chapter provided details on the extensive literature review conducted for this research project. It explained commonly found Tomato plant diseases and various methods to detect those and how those evolved in the past few decades. The chapter also covered details about publicly available image datasets for automated plant disease detection and reviewed their performances with various models. The chapter covered the evolution of Computer Vision and Machine Learning models which have played a crucial role in the agriculture field. The literature review then focused on review and performance analysis of various CNN architectures and transfer learning techniques followed by how models are shrunk in sizes keeping the same accuracy level to make them suitable for deployment on low compute devices such as smartphones. This chapter also contained details about feature extraction techniques that can be leveraged for measuring the degree of disease severity to predict the spread of disease. Finally, the discussion section summarized the significance and performance benchmarks of various research papers and journals reviewed during the literature review.

CHAPTER III:

RESEARCH METHODOLOGY

3.1 Introduction

This chapter will contain the details of the research methodology adopted for this research project. The first subsection will explain the systematic research approach framework leveraged during each stage of research experimentation. The next subsection section will discuss the characteristics of the datasets which will be used for this research work and how a subset of the dataset will be selected for this research study. The subsequent section on data pre-processing will include the details on techniques used for image acquisition, scaling and transformations, data normalization, and various image augmentation techniques which will be applied to handle data imbalances in the original dataset. The feature extraction subsection will explain how the automatic feature extraction will be carried out as a part of convolutional neural network training and required features for measuring the degree of plant disease will be extracted using various feature extraction algorithms.

The model training section will contain the details of the proposed CNN image classification model to detect tomato plant details. It will explain the model architectures to be considered as a base model for this experimentation along with reasoning on why those model architectures are preferred for real-world solution deployment on mobile devices. The section will further cover the details on how the transfer learning techniques will be used in the research work and will provide details on the convolution neural network layers and training parameters. It will be followed by the details on which hyperparameters are tunned as a part of model optimization. Then the subsequent section will explain the model evaluation process and which statistical metrics will be used to evaluate trained model performance. The subsection on predicting the degree of the disease will explain the algorithms to be used for extracting important features such as color, shape, and texture to measure the spread of the disease and how the input images will be further classified based on the degree of disease.

The next section will cover the details on how the model will be deployed on an Android mobile phone for real-time inferencing in the actual tomato plantation field. The last subsection will provide details of research tools and utilities being leveraged for model training and evaluation which includes hardware as well as software resources.

3.2 Research Approach

The research methodology adopted for this study will include the standard machine learning processes. They are data acquisition & processing, data standardization & transformations, handle class imbalance, split the training and test data, model training and evaluation, model optimization using hyperparameter tuning, and model testing. It will be followed by model deployment and inferencing in an actual production environment. The research process will begin with the loading of images from two public datasets (PlantVillage and PlantDoc) during the Image acquisition step and then prepare the data using pre-processing methods as a part of the "Data Processing" stage. After the necessary data pre-processing is done and the dataset is standardized, necessary features to identify plant disease will be extracted automatically using the CNN model and even manually by leveraging different feature extraction techniques. The manually extracted features will help to predict the actual degree of disease. Next, in the "Model Training" stage, the various CNN models will be trained using different CNN architectures suitable for low compute devices such as mobile phones and their performance will be evaluated. Then the trained models will be optimized by performing hyperparameters tuning. In the third and final stage, the best model will be identified, which then will be deployed into the production environment i.e., in Android App for this research and interferences will be carried out in the actual farm field.

These key process steps will be grouped into three stages as shown in Figure 6.

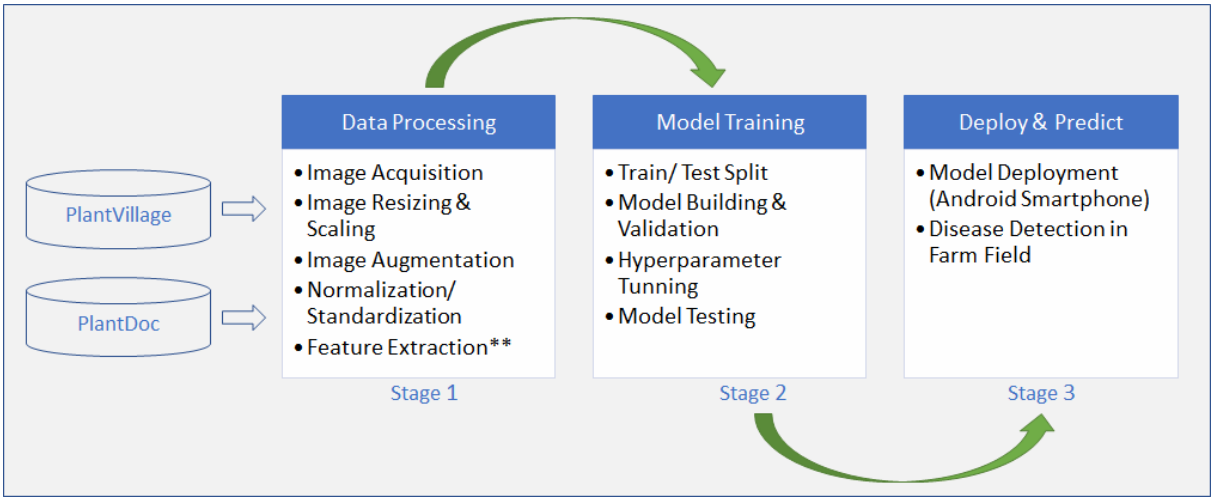


Figure 6 – Research Methodology Process

Feature Extraction** - The manual feature extraction is not required for the CNN model as automatic feature extraction will take place as a part of the learning process. However, the required features will be manually extracted for predicting the degree of disease.

The process flow diagram in Figure 7 explains the exact steps which will be performed in sequence for this research project.

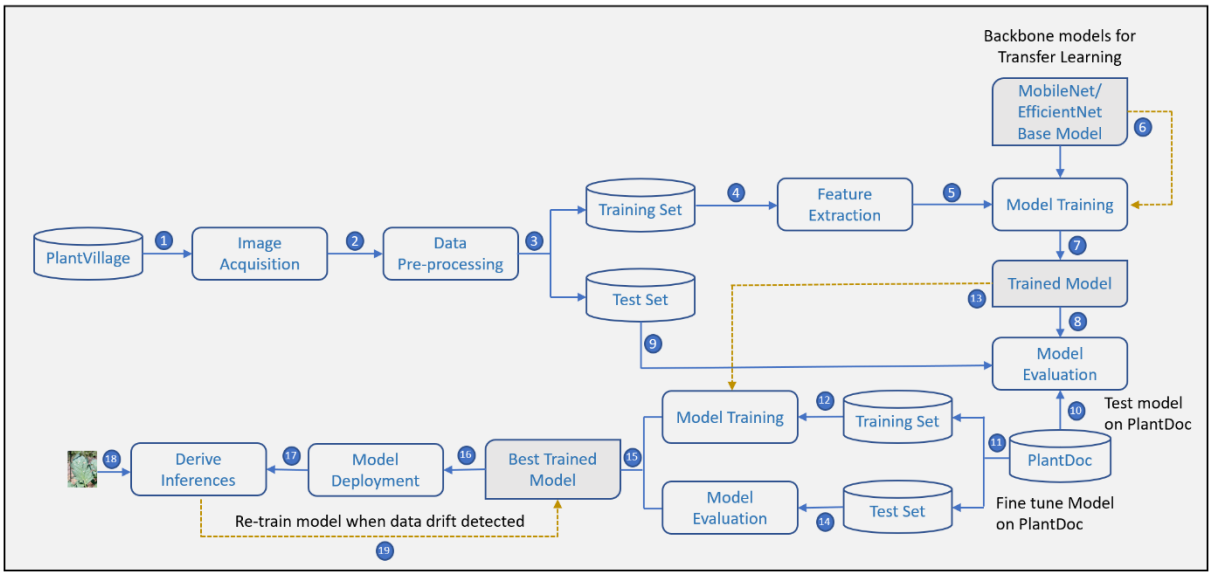


Figure 7 – Process Flow

All the individual steps in the Process Flow are explained in detail in the subsequent sections.

3.3 Dataset Selection

To implement a real-world plant disease detection solution, the model needs to be trained on enough plant leaves images of high quality. A robust CNN model will be trained by taking into consideration the varied background conditions such as different angles, lighting arrangements, and backgrounds. The plant background includes other plants, weeds, flowers, fruits, stems, and human hands. This study will leverage two public plant image datasets including the PlantVillage dataset (Mohanty et al., 2016), which contains the high number of images gathered from a controlled (lab) environment. The second PlantDoc (Singh et al., 2020) dataset contains images from real plantation fields with the presence of different types of background noise. However, there are a smaller number of tomato plant leaf images as compared to the PlantVillage dataset. Initially, the training process will start by using 14,707 Tomato crop color images from the PlantVillage dataset which is distributed across 7 different classes as shown in Table 2 – PlantVillage extracted Dataset Class Distribution.

Table 2 – PlantVillage extracted Dataset Class Distribution

#	Class	Image Count
1	Tomato_Bacterial_Spot	2127
2	Tomato_Early_Blight	1000
3	Tomato_Healthy	1591
4	Tomato_Late_Blight	1909
5	Tomato_Leaf_Mold	952
6	Tomato_Septoria_Leaf_Spot	1771
7	Tomato_Yellow_Leaf_Curl_Virus	5357

The model trained with the PlantVillage dataset will be then used for further classifying the real-world images from the PlantDoc dataset which contains 2,598 images having a total of 27 classes spanning over 13 species. This research will use the subset of the PlantDoc dataset with 7 types of image classes that are mapping with the trained CNN model on the PlantVillage dataset. The PlantDoc dataset structure is organized into training and test datasets as shown in Table 3 – PlantDoc extracted Dataset Class Distribution.

Table 3 – PlantDoc extracted Dataset Class Distribution

#	Class	Train Image Count	Test Image Count
1	Tomato_Bacterial_Spot	86	15
2	Tomato_Early_Blight	76	17
3	Tomato_Healthy	51	17
4	Tomato_Late_Blight	100	19
5	Tomato_Leaf_Mold	78	15
6	Tomato_Septoria_Leaf_Spot	126	20
7	Tomato_Yellow_Leaf_Curl_Virus	68	15

It is expected that the accuracy level will decrease with the PlantDoc dataset due to background noise in the actual field images, such as other leaves, stems, and other background objects. As a next experiment, the PlantDoc dataset will be used for fine-tuning the trained CNN model with transfer learning and validate the model performance.

3.4 Data Processing

PlantVillage and PlantDoc datasets are already annotated and curated by experts and hence don't require any data cleansing and labeling. The research work will load the images and perform the pre-processing as per the details mentioned in the next subsections.

3.4.1 Load Image Dataset

To train the CNN model, image data needs to be either converted to a NumPy array or tensor object. Image data can be loaded manually using the OpenCV library and then converted into pixel arrays. Similarly, output class labels need to be converted to integer values. However, this research will be leveraging Keras's inbuilt *ImageDataGenerator* class library to perform this task. Since there are two datasets with varying sizes, it will be beneficial to use the Keras *ImageDataGenerator* class, which will allow loading a single dataset in batches. The Keras *ImageDataGenerator* class supports progressive image loading. It will load only the required number of images in memory for the current and few next mini batches while training and evaluating a CNN model. The Keras *ImageDataGenerator* provides the *flow_from_directory()* function, which accepts the image dataset folder as an input parameter and then iteratively loads images.

3.4.2 Resize Images

In a convolutional neural network training, resizing images to an appropriate size plays an important role as it will directly impact the model training time and model performance. For different CNN model architectures based on MobileNet, EfficientNet, and NasNetMobile, different common input image resolutions such as 224 X 224 pixels, 128 X 128, and 192 X 192 pixels will be experimented to check which resolution improves model performance. The research will use either the OpenCV library to resize the images to the required resolution or will leverage Keras *ImageDataGenerator* class functions to resize the images.

3.4.3 Normalize, Center, and Standardize Image Dataset

The pixel values in the image range from 0 to 255 which are required to be normalized, centered, and standardized before using for CNN model training. Using larger pixel values will impact the speed of model training due to increased calculation complexities. Therefore, in this study, image pixel values will be normalized between 0 and 1 or -1 and 1 depending on the base model requirements for faster model training. To center images for PlantVillage and PlantDoc dataset, the

first mean pixel value for each of the datasets will be calculated and then it will be subtracted from each image within the dataset. It will result in pixel values distribution being centered on the zero value. The mean pixel value can be calculated for the entire dataset or a mini-batch or even for each of the images. There are two types of centering methods – Global Centering, where the mean pixel value is calculated and subtracted across color channels; and Local Centering, where the centering is performed at per color channel. The image dataset will be standardized to have a zero mean value and standard deviation of one, which follows the Gaussian distribution. The standardized image dataset will improve the training and evaluation process as the dataset will sum to zero and input values will range between -3.0 to 3.0 by following standard normal distribution or empirical rule. The image dataset can be standardized at the image level, mini-batch level, or at the entire dataset level like the centering of images.

There are multiple ways to apply image scaling techniques to normalize, center, and standardize the images. It can be done manually using NumPy and OpenCV libraries. As in this research, images will be loaded using the Keras *ImageDataGenerator* class, it will use the same class even for scaling the images. It will allow the application of the required data scaling just in time while training and evaluating the model. During the instantiation of the *ImageDataGenerator* class, required arguments will be passed for normalizing, centering, and standardizing the image dataset.

3.4.4 Data Augmentation

The subset of the PlantDoc image dataset contains a lower number of images i.e., a total of 690 images and only 631 images as training data. There is even a class imbalance observed for some of the disease categories. For example, Tomato_Yellow_Leaf_Curl_Virus class contains more than double the number of images and the Leaf_Mold class contains a smaller number of images. Usually, a deep neural network model requires a greater number of images to generalize well and avoid overfitting scenarios. This research will be using various image augmentation techniques such as horizontal flipping, rotation, shifting, zoom, and change in brightness. Unlike, image scaling processes, image data augmentation will be only applied to training datasets and not to validation and test datasets. This study will consider the training dataset characteristics (such as class imbalance ratio) and knowledge of the problem domain i.e., plant disease detection to choose appropriate configurations for image data augmentation. In this research, dynamic in-place image

augmentation will be performed using *ImageDataGenerator* class from Keras Deep Learning library.

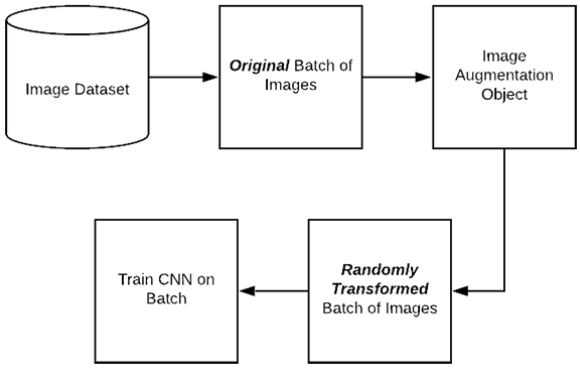


Figure 8 – In-place dynamic data augmentation (Adrian Rosebrock, 2019)

The *ImageDataGenerator* class will apply in-place data augmentation as a part of the training process itself on the batches of the dataset and will return the transformed dataset to the deep neural network model during every epoch. The research will experiment with different settings of image data augmentation to analyze the impact on the model performance.

3.5 Feature Extraction and Selection

After completing data acquisition and data processing steps, the research will perform the most important steps of Feature Extraction and Feature Selection in Deep Learning process flow as shown in Figure 7. During this step, required features will be extracted from images for both the datasets which will help to classify tomato plant diseases and predict the degree of disease. The common features which will be extracted from the plant leaf images include color, shape, texture, and spot size details. The research work will consider various feature extraction techniques such as processing color histograms to extract color and texture features and Grey Level Co-occurrence Methodology (GLCM) to extract shape features. Also, Spatial Gray-level Dependence Matrices (SGDM) will be used to extract energy, contrast, correlation, and local homogeneity features (Chapaneri et al., 2020). The study will be aiming to experiment with a combination of various feature extraction algorithms and by performing automatic feature extraction using CNN with a transfer learning approach.

3.5.1 Automatic Feature Extraction

This research will be leveraging pre-trained CNN base models from Keras Application API as an arbitrary feature extractor and using transfer learning techniques to perform automatic feature

extraction. Figure 9 shows the indicative approach this study will be adopting for automatic feature extraction. The left-hand side is the original VGG16 network architecture, which will be updated by removing a fully connected layer, which is shown in the right-hand side model.

As shown by general CNN architecture in Figure 9, this research work uses convolution layers + ReLU activation function + pooling layers which are commonly known as “Convolution Base” to learn the features from input plant leaf images.

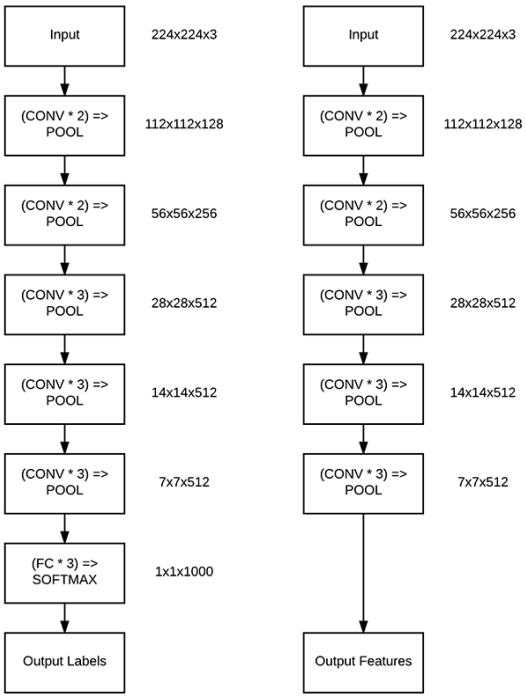


Figure 9 – CNN as Feature Extractor (Rosebrock, 2017)

3.5.2 Manual Feature Extraction

For predicting a degree of disease, features such as deformation in color, shape, and texture of tomato leaves play an important role. Even, the previous research work commonly used HOG, SIFT, and SURF features for the Plant Disease Detection task. Histograms of Oriented Gradient (HOG) (Dalal and Triggs, 2005) were used to get Hue histograms. Scale Invariant Feature Transforms (SIFT) (Lowe, 1999) extracts scale-invariant features by applying a staged filtering approach. Speeded Up Robust Features (SURF) (Bay et al., 2008) detects feature which is scale and rotation invariant using Hessian matrix-based measure and distribution-based descriptor. Based on the literature review performed, it is evident that color and SIFT features have shown good performance for plant disease detection tasks. However, SIFT and SURF algorithms are

patented and cannot be freely used for a mobile app that is distributed through Android App Store. Therefore, this study will be extracting color features and ORB feature vectors to predict the severity of the disease. The ORB features will provide the best of two algorithms which include Features from Accelerated Segment Test (FAST) keypoint detector algorithm and Binary Robust Independent Elementary Features (BRIEF) feature descriptor. This study will be performing feature extraction using traditional computer vision algorithms for image features extraction and compare the classification model trained with these features with the CNN model trained with transfer learning.

3.5.3 Feature Selection

Feature selection will be the next important step after feature extraction, as not all features will contribute equally to plant disease detection. The research work will focus on removing data redundancy by selecting those important features or a combination of feature vectors that contribute to improving the model efficacy. For the task of plant disease detection using leaf images, every feature will have certain advantages and disadvantages. Based on the review of previous work done, it is clear that a higher number of features will increase the complexity of the trained model. However, it will help to identify interesting regions containing key points to measure the degree of disease accurately. To strike a balance between the accuracy and complexity of a model, this study will experiment with different features and select a few common features such as color and texture which will be contributing higher.

3.6 Model Training

This section explains details on the proposed model training approach for this research work.

3.6.1 Proposed CNN Model

To achieve the objective of identifying six types of commonly found tomato crop diseases, the research work will be designing and implementing the deep convolutional network. Also, this research will further train a model to predict the degree of disease spread and classify the input plant images within each disease category. These four sub-categories for the degree of the disease will include – L0-Healthy, L1-Early stage, L2-Middle stage, and L3-Severe Damage stage. Image classification is a supervised machine learning technique, which assigns a label to an input image based on the predefined set of categories (Rosebrock, 2017). This research will classify the input

images into 19 different categories, which will be a combination of disease types (or healthy) and level of degree of disease as shown in Table 4.

Table 4 – Classification models Categories (classes)

#	**Category Labels**	#	**Category Labels**
1	Healthy-L0	2	Bacterial_spot-L1...L3
3	Early_blight- L1...L3	4	Late_blight-L1...L3
5	Leaf_mold-L1...L3	6	Septoria_leaf_spot-L1...L3
7	Yellow_Leaf_Curl_Virus-L1…L3		

The traditional image classification approach involves the extraction of hand-crafted features using various techniques and applies machine learning algorithms such as K-Nearest Neighbor (KNN) or SVM classifier. However, with recent advancements in the field of deep learning and convolutional neural networks with transfer learning, remarkable results are achieved in various computer vision problem areas such as image classification and object detection. This research work will leverage Convolutional Neural Network (CNN) based solution for tomato disease classification. Figure 10 shows the CNN general architecture applied for this research, which can be divided into three main stages, which include acquisition of input image data along with data processing, feature learning, and image classification.

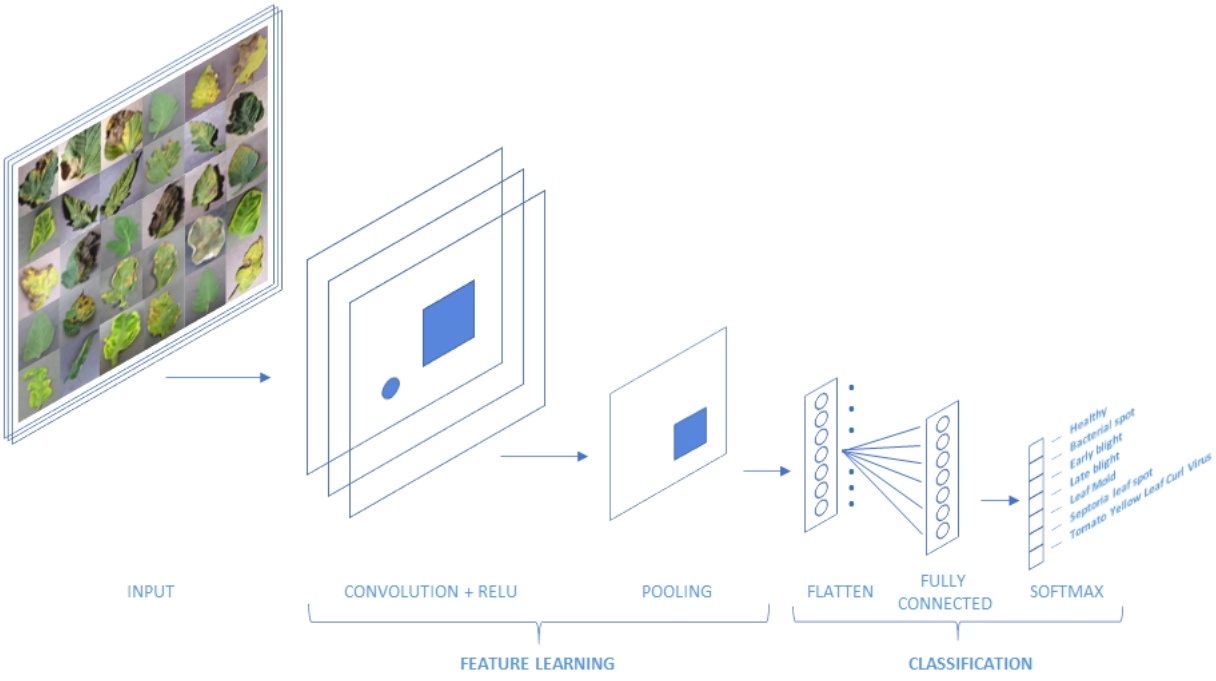


Figure 10 – CNN General Architecture

After data acquisition and applying the data pre-processing techniques as mentioned in section 3.4

Data Processing to normalize and standardize the image dataset, the research experiment will pass the resulted data set through the convolutional neural network. The CNN structure design will contain five types of layers which include convolutional layers followed by the activation function, pooling layers, flatten layers, fully connected layers, and finally a SoftMax Classification Layer. In the feature learning stage, the CNN model will extract and learn the features from the input image. It will start with learning generic features such as edges and corners of the tomato plant leaf images and then use these generic features to learn specific features using convolution layers at the end. This research will make use of a flatten out and fully connected layer for performing actual image classification in healthy and the six types of tomato plant diseases.

3.6.1.1 Convolutional Layer

The Convolutional Layer is the core building block of the proposed CNN model, which will apply a convolution operation to the input image, and then it will pass the result to the next layer. A convolutional layer will shrink the input by extracting features of interest. For responding to different feature detectors, it will be creating feature maps. As it is practically not possible to connect neurons to all the neurons in the previous layer due to large spatial dimensions, each convolutional neuron will process data only for its receptive field. The Convolutional Layer will consist of a set of learnable filters (also known as “kernels”) with the same width and height i.e., of a square shape. The output size after applying the convolution operation will be dependent on depth, stride, and padding parameters. The depth of the output volume will control the number of neurons i.e., filters in the Convolutional layer that will connect to a local region of the input volume. A convolution operation will slide a small matrix across a large matrix, stopping at each coordinate, computing an element-wise multiplication and sum, then storing the output. A “Stride” (denoted by “S”) will decide how much a filter matrix will slide from left to right and then top to bottom. This research will experiment with a Stride step size of S = 1 for smaller input image resolutions (<200 pixels) and S = 2 for bigger input image resolutions (>200 pixels). The smaller strides will lead to overlapping receptive fields and larger output volumes and larger strides will result in less overlapping receptive fields and smaller output volumes. The research will use zero paddings to ensure the output volume size is matching with the input volume size. The padding technique will help to control the speed of decrease in spatial dimensions so that deep neural networks can learn useful patterns from input images. The output volume can be calculated using the input volume

size (*W*), the size of the receptive field (*F*), the Stride (*S*), and the amount of padding (*P*) using the equation [1]. It will be ensured that equation [1] results in an integer value.

$$((W - F + 2P)/ S) + 1 \quad [1]$$

The convolutional layer will allow reducing the free parameters of the deep CNN model which will result in fewer network parameters. As this study focuses on deploying the solution on low compute devices, the number of network parameters and overall model size play a very important role.

3.6.1.2 Activation Function

After applying the Convolution operation, the output (weighted sum + bias) will be passed to the non-linear ReLU activation function. ReLU activation function will convert any negative input value to zero and positive value to the same input number using the equation [2].

$$f(x) = \max(0, x) \quad [2]$$

A non-linear activation function like ReLU will ensure that a neural network becomes a universal approximator of continuous functions in a Euclidean space.

3.6.1.3 Pooling Layer

The pooling layer will serve two main purposes for the proposed CNN model. First, it will allow reducing the spatial size dimensions (width and height) of input volume progressively. Second, it will allow controlling the overfitting with pooling window, concentrating local data. Also, the pooling layer will introduce translation, scale, and rotational invariance which will ensure slight scaling or dislocations do not make any differences. This research will try applying max pooling or average pooling as per the base network architecture and evaluate model performance.

3.6.1.4 Flattening Layer

The flattening layer will perform the key operation of transforming two-dimensional matrix output from the convolutional layer into a single-dimensional vector which will be passed to a fully connected layer. This layer will be added in between the convolutional layer and the fully connected layer.

3.6.1.5 Fully Connected Layer with SoftMax

This research experiment will place one or two fully connected layers at the end of network architecture, which will be fully connected to all activations in the previous layer. The fully connected layer(s) will be added before the SoftMax classifier. Finally, the SoftMax classifier will compute class probabilities for each disease type in scope to perform the disease detection.

3.6.1.6 Batch Normalization Layer

Batch Normalization layers are used to standardize the inputs to a deep neural network layer for each mini batch. Batch normalization will increase the model efficacy by reducing the number of epochs it takes to train a CNN model. It will also stabilize the training by allowing a larger variety of learning rates and providing regularization strengths. However, it might increase the time taken for training the neural network model because of additional computations needed per batch and the normalization process. As per the original paper by Ioffe and Szegedy (2015), Batch Normalization (BN) layer is added before applying the activation function. However, based on the literature review, this research will consider adding the BN layer after the ReLU activation function.

3.6.1.7 Dropout Layer

Dropout is a regularization technique that increases test accuracy by avoiding overfitting for trained neural network models (Srivastava, 2014). For each mini batch, the dropout layer will randomly disconnect inputs from the previous layer to the next layer during the training process. A dropout layer will be added between two fully connected layers in the CNN model with a drop probability ratio of 50% ($p = 0.5$). Since this research will be using a transfer learning approach and will not train the CNN model from scratch, all the above parameters will apply to only a trainable layer of network architecture.

The visualization of the CNN model structure will be included for the final dissertation model after completion of model training experimentations and identification of the best CNN model architecture.

3.6.2 CNN with Transfer Learning

Training a Convolutional Neural Network from scratch requires a lot of computational resources and considerable training time. Also, achieving state-of-the-art model performance by training CNN from scratch requires a large image dataset. Due to absence of such a large image dataset for

tomato leaf images from the actual field and to ensure the image classification model converges faster with a higher classification accuracy rate, this research work will leverage Transfer Learning. The transfer learning approach refers to a process in which a pre-trained deep learning network on a large dataset (ImageNet or COCO) will be utilized in some way for another related problem. This research work will be using the transfer learning process as an integrated feature extractor and fine-tune the CNN model for the disease classification task.

Based on the detailed literature review about previous CNN model comparative performance analysis, this research work shortlisted MobileNet V1, V2, EfficientNet-B0, and NasNetMobile as backbone CNN models for Transfer Learning. The experiment will be using the trained weights on the ImageNet dataset. Initial convolution layers of the network from the pre-trained model on the ImageNet dataset will extract generic features which are – edges, corners, and color intensity. As the model will evolve, the later convolution layers will identify more specific and complex features such as the texture of the plant leaves images. Considering the difference between the ImageNet dataset and plant leaves datasets used for this research and further considering size variations for these datasets, the model will be re-training some of the top (later) layers while leaving initial layers frozen in the network architecture.

The research experiment will first train a CNN model on the PlantVillage dataset using the transfer learning approach where a short-listed lighter CNN network architecture will be used as a feature extractor. The different CNN models based on selected backbone networks will be evaluated to identify the best model. Only fully connected layers will be replaced with new FC layers keeping all convolutional layers freeze as shown in Figure 111.

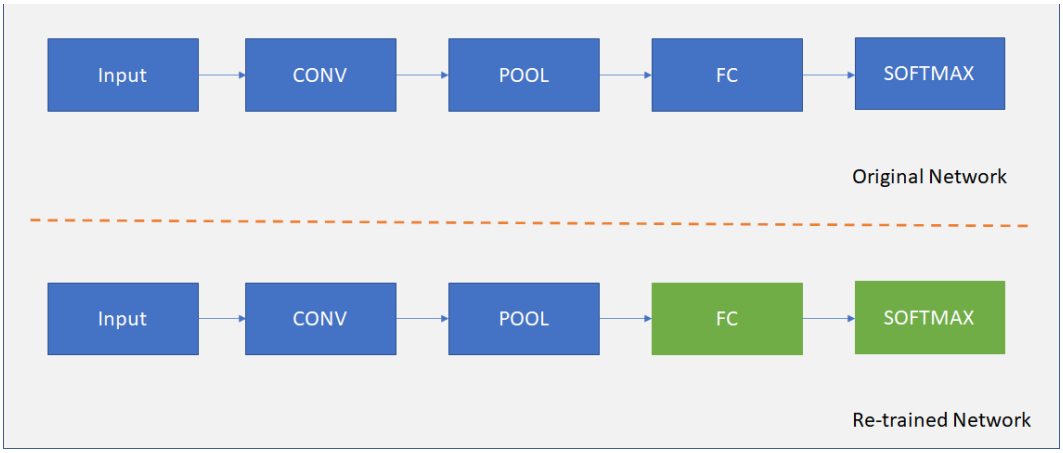


Figure 11 – Transfer Learning Model as a Feature Extractor

As a next step, an experiment will be performed to unfreeze some of the Convolutional Layers and re-train the CNN model to tune the model performance on the PlantVillage dataset as shown in Figure 12.

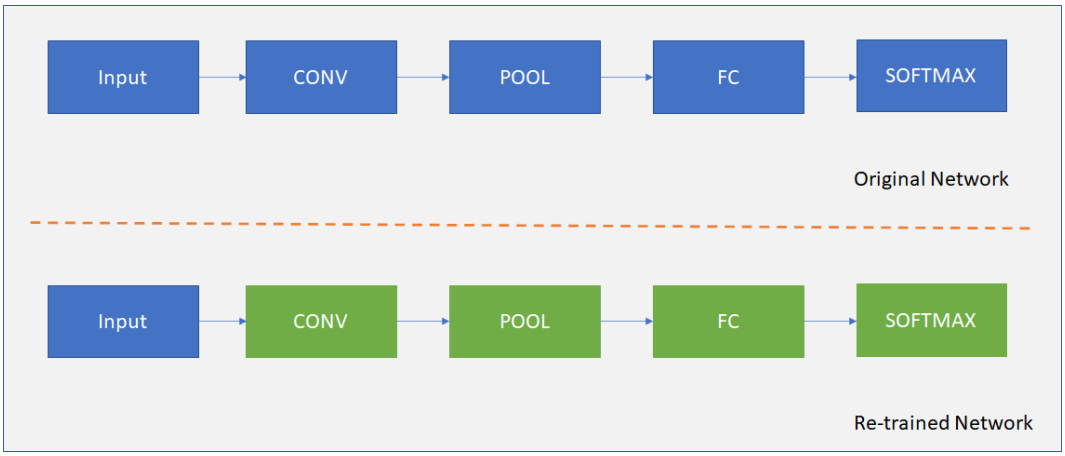


Figure 12 – Transfer Learning to fine tune CNN model

The best CNN-trained model with the PlantVillage dataset will then be used to test against PlantDoc dataset images, which contains real-world images with various types of background noise. In case of drop-in model performance (which is an expected scenario), a new CNN model will be fine-tuned using the PlantDoc dataset. The CNN model with the PlantDoc dataset will leverage the already trained best CNN model with the PlantVillage dataset as the backbone model as shown in Figure 7 – Process Flow. As the PlantDoc dataset is small in size and does not differ much from the PlantVillage dataset, a larger number of layers will be locked as untrainable and will not be re-trained to avoid overfitting.

3.6.3 Model Evaluation and Testing

This section explains various model evaluation considerations and lists down the classification matrix used by this research.

3.6.3.1 Model Evaluation Considerations

This research will be training and evaluating multiple models for classifying the input plant images into seven different categories (healthy + six types of disease categories) and then further subclassify as per the degree of disease spread (L0 to L3). During the model training and evaluation, various parameters will experiment which include - choice of activation functions, optimizer method with different momentum values, learning rate, weight initialization techniques,

batch size, and several epochs. As the research will be leveraging the transfer learning approach, the models will be evaluated with different short-listed CNN model architectures as backbone networks such as MobileNet V1, V2, and EfficientNet B0. During the transfer learning process, the model will be trained and evaluated with different input sizes for images, filter sizes, stride and padding values, number of freeze and trainable convolutional layers, and dropout ratio. Further, important features such as color, shape, and texture will be manually extracted and evaluated to measure the degree of disease for selected disease type(s). These models will be evaluated using different classification metrics which are explained in the following sections. The trained classifier models will be evaluated using different ratios of train test splits and k-Fold Cross-Validation technique by experimenting with different values of k-fold.

3.6.3.2 Model Evaluation Metrics

The choice of correct model evaluation metrics plays an important role in improving Classification model performance. As not all the metrics will work equally well for different types of datasets and modeling techniques, the research will consider the appropriate metrics mentioned in this section. For example, overall accuracy will not be a good fit for an imbalanced dataset (Datacourses.com, 2020).

In the multi-class classification task for this research, overall accuracy will be calculated as the ratio of a total number of correct predictions to all the input samples, which is calculated using equation [3].

$$Accuracy = \frac{Correct\ Predictions}{All\ Predictions} \quad [3]$$

To evaluate model performance, training and validation accuracy will be compared.

The Loss function balancing is used to improve model accuracy while ensuring that the model is not overfitting and plays a critical role in neural network model building. In this research, the output class labels will be integers and hence it will make use of sparse categorical cross-entropy. It optimizes time in memory and computation due to the use of integer values for a class instead of a vector.

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \quad [4]$$

During the evaluation process, training and test loss will be plotted and compared with the training and test accuracy plot.

Precision is the ratio between correct positive predictions compared against total positive predictions made by the classifier which includes true and false positives as indicated by equation [5].

$$Precision = \frac{Correct\ Positive\ Predictions}{All\ Positive\ Prediction} \quad [5]$$

Recall or Sensitivity matrix will evaluate the fraction of positive samples which are correctly predicted as positive. In multi-class classification, it will indicate the probability of detection of correct healthy and different types of disease classes. It will be calculated using the equation [6].

$$Recall = \frac{Correct\ Positive\ Predictions}{All\ Positives} \quad [6]$$

F1 Score will measure test accuracy and is an indicator of harmonic mean between precision and recall. It indicates the precision and recalls trade-off i.e., the preciseness (number of instances classified correctly) and robustness (a significant number of instances not missed) of the classifier.

$$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \quad [7]$$

3.6.3.3 Data Splitting

This research will experiment with different techniques of splitting data to evaluate the model performance on "unseen data." Initially, it will start with a hold-out strategy by leveraging the *train_test_split* method from the ScikitLearn library. The research will experiment with different percentages of the train and test splits such as 80-20, 70-30, and 67-33. It will also allocate 10-20% for the validation dataset on which hyperparameters tunning will be performed.

Considering that the dataset size is not large, the next experiment will leverage the k-Fold Cross-Validation technique with different values of k such as 5 or 10. It will split the training dataset into k partitions of equal size and train the model on all partitions of data except the one which is held out. This process will be repeated for k times with different partition held out. The experiment will be using *StratifiedKFold* class from the ScikitLearn library to perform k-Fold Cross-Validation.

As the PlantDoc dataset is already divided into train and test datasets, the above experiments will be carried out with the PlantVillage dataset only.

3.7 Model Deployment & Prediction

This research will be implementing a disease detection deep learning model using Keras open-source library with TensorFlow backend. After training and evaluating various model performances, it will select the best model for deployment onto the Android app. To deploy the TensorFlow model onto mobile phones or other low compute embedded devices, such as microcontrollers, the model needs to be optimized for deployment. This includes compressing the model size to reduce memory consumption and shorten the download time. The computation required to perform predictions by model needs to be minimized and optimized to ensure low latency and reduced battery usage which can avoid battery heating.

This research will leverage TensorFlow Lite tools to deploy and run the SmartCropCNNLite model into Android Smartphones. TensorFlow Lite provides two important components – TensorFlow Lite interpreter and TensorFlow Lite converter. To start with model deployment, the TensorFlow model will be extracted from the backend of Keras, and it will be saved as a TensorFlow Lite model. The Keras API used from Python code will convert the Keras model to a TensorFlow Lite model (*.tflite). The *.tflite file will then be imported into Android Studio by creating a project. Android Studio is an Integrated Development Environment (IDE) for creating Android apps. The Android app will be using TensorFlow Lite interpreter Java APIs to execute the model and perform inferences using smartphones in the plantation field. This flow is explained in Figure 13.

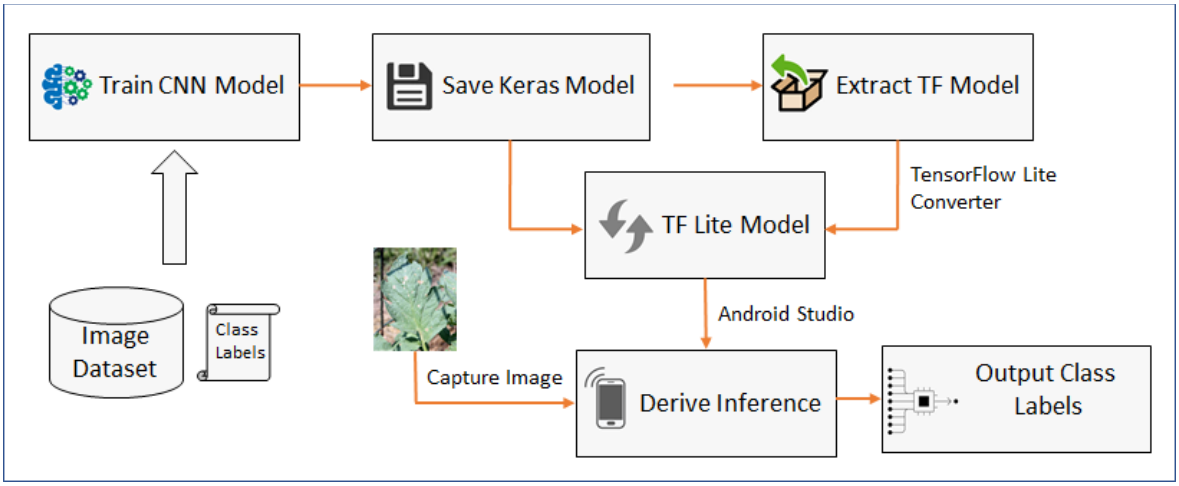


Figure 13 – CNN Model Deployment Flow

The research will then further experiment with TensorFlow Optimization Toolkit to apply quantization and perform model pruning. The quantization will reduce the precision of the numbers used to represent model parameters which will reduce the model size further. The model pruning technique will remove those parameters which have a minor impact on model predictions which can compress the model for reduced download size.

3.8 Tools and Utilities

This section provides details of research tools and utilities used for model training and evaluation which includes hardware as well as software resources.

3.8.1 Software Libraries

The research work utilized the following software libraries for data loading, processing, feature extraction, model training, evaluation, and deployment.

TensorFlow is open-source software for Machine Learning which is developed and maintained by Google. Though it can be used for multiple ML tasks, it primarily focuses on the training and inference of deep neural networks. TensorFlow expresses computations as a stateful data flow graph, enabling easy scaling of ANN training with parallelization and replication. It is widely adopted as it provides an interface to express common ML algorithms and executable code of the models. Models created in TensorFlow can be ported to heterogeneous systems with little or no change, which supports multiple devices, ranging from mobile phones to distributed servers. This research work has trained various CNN models on computational servers and later ported the best-performing model to mobile devices, which makes TensorFlow highly suitable. In this research, TensorFlow 2.4.1 stable release is used.

Keras is a primary deep learning library used to train the CNN models for this research which provides a user-friendly Python interface on top of the TensorFlow machine learning platform. The Keras Application APIs are widely used to implement all the CNN models and to apply transfer learning techniques in addition to data loading, processing, and augmentation capabilities.

TensorFlow Lite library contains a set of tools to embed and run the machine learning models on mobile, embedded, and IoT devices. This research has used TF Lite Converter to convert the pre-

trained Keras models with TensorFlow background to optimized FlatBuffer TFLite format which is then integrated into the Android App using the Android Studio.

OpenCV is a real-time image processing library which is originally developed by Intel. The OpenCV library is written in C/C++ programming language, however, it provides Python bindings which makes it easy to use in Python code. It is mainly used in this research work for image data analysis and implementing manual feature extraction functionality to predict the degree of tomato plant disease.

Mahotas image processing library is used to complement the OpenCV as Mahotas interface for some of the feature extraction algorithms, such as color, shape, and edge are easier to use.

Scikit-learn is a free machine learning library that provides various classification, regression, and clustering ML algorithms. It also provides the capabilities to process data and evaluates model performance using various metrics. In this research, the Scikit-learn library is used to implement a hold-out strategy (train-test split) and evaluate the performances of CNN image classifier models.

Matplotlib and Seaborn graphic libraries are used for creating and visualizing many graphs and charts to analyze the image datasets and compare the models' performances.

Python programming language is being used for data loading, data processing, manual feature extraction, EDA, and model development. Python language has intuitive syntax, and it is very easy to understand Python code due to the similarity of keywords with native English language. This made Python a perfect choice for this research project.

Java programming language is used for creating Android applications as it is widely accepted for cross-platform code development. **Android Studio** is used to import the CNN model, design the UI screens, and integrate the CNN model with the app.

3.8.2 Hardware Configuration

The CNN models are trained using the below compute resources.

- CPU: 9th Generation i7-9700 Processor with 16 GB of memory
- GPU: NVIDIA T4 GPUs with 16 GB of memory

The model deployment and testing are done on an Android Studio Emulator with the below configuration.

- Android OS 4.4 (KitKat) compatibility
- Pixel 2 Device with API level 30

3.9 Summary

This chapter provided a detailed explanation of the research methodology adopted for this study. The chapter started with details of research process flow components which are then covered in the next subsections. The data loading subsection provided the details on which datasets will be acquired for performing tomato crop disease detection along with the prediction of the degree of disease spread. The data pre-processing section explained the process for resizing the image datasets and then apply various data scaling techniques including normalization, centering, and standardization to improve the training performance and speed. The section on data augmentation contained the details on various techniques used for augmenting image datasets to avoid class imbalance and to control overfitting which can lead to better model generalization. The next section covered the important aspects of feature engineering including feature extraction and feature selection.

The model training section explained the details on the CNN model network architecture and building blocks of the CNN model trained. The following section mentioned the advantages of transfer learning instead of training the CNN model from scratch and then explained the process of applying transfer learning for this research work. The next section listed the procedure for evaluating and testing the CNN models. The steps for selecting the best model for deployment and performing prediction in the field are explained later in this section. The next section on model deployment provided a process flow on how to deploy this model onto an Android app that can perform the inferences in the actual farm field using a smartphone device. The last subsection provided the details of tools and utilities used for this research work including hardware and software resources.

CHAPTER IV:

ANALYSIS AND DESIGN

4.1 Introduction

This chapter will provide details on data analysis, data processing, model training, and hyperparameter tuning. The chapter begins with the data preparation section which contains a subsection on data loading followed by a subsection on the analysis performed to validate the correctness of image annotations. Based on this analysis, data cleaning activities will be performed. The next two subsections will explain how cropped image dataset is created using bounding box information from PlantDoc dataset Pascal VOC XML annotations. The following subsection will explain the approach and steps followed for the creation of the disease severity dataset, which is created using the PlantDoc dataset. The important steps of data normalization performed by resizing and rescaling images will be covered in the next subsection. The data augmentation subsection will explain the steps performed for image data augmentation to avoid model overfitting.

The section on Train and Test dataset splitting will explain how the original dataset is split into train and test datasets by applying the hold-out strategy. The next important section on Exploratory Data Analysis (EDA) starts by explaining high-level data analysis to be carried out on both PlantVillage and PlantDoc datasets. The next subsection provides details of basic EDA conducted on both datasets. The analysis from the first two sections combined will help to decide on the data imbalance percentage and how it will be treated while model training and evaluation. The following section explains the details on how class imbalance is handled for both the datasets and which class imbalance techniques are used, such as oversampling and under-sampling.

The next section on model training explains the details of how various CNN classifier models are trained using transfer learning. This section further discusses the network architecture adopted for the best-selected model. The subsequent section explains the steps performed for hyperparameter tuning to avoid overfitting of the model and improve the model performance. It includes hyperparameter details such as different input image sizes, optimizers with various learning rates, learning rate decay (learning rate schedules), momentum, early stopping callback, and so on.

The section on model evaluation will discuss the metrics used for the evaluation model and how the best model is selected to deploy onto the Android App. The section on model deployment will explain the steps performed to create the TFLite model, which is then integrated with Android App. The final section on disease severity prediction will explain the details of manual feature extraction and machine learning classification models (Random Forest and SVM). Further, it will explain the second approach to classify the tomato plant images based on disease type and its severity.

4.2 Dataset Preparation

To begin with dataset loading and preparation, first, both the datasets were downloaded from the public links mentioned in Table 5 – Dataset Links.

Table 5 – Dataset Links

Dataset Name	**Public Link (URL)**
PlantVillage	https://www.kaggle.com/abdallahalidev/plantvillage-dataset
PlantDoc	https://github.com/pratikkayal/PlantDoc-Dataset

4.2.1 Load Dataset

After downloading the image dataset, the images were loaded using *ImageDataGenerator* class from Keras Data Processing API which allows the creation of batches of tensor image data and applies the data augmentation techniques while training the model in real-time. The research has leveraged the *flow_from_directory()* method which accepts the directory path as a parameter and then generates batches of tensor image data. Figure 14 shows the sample images loaded from both PlantVillage and PlantDoc datasets.

Figure 14 – Sample images from PlantVillage (left) dataset and PlantDoc (right) dataset

As observed from Figure 14, the PlantVillage dataset images are taken from a controlled environment and contain only the leaf image with no background and with consistent lighting. On the contrary, the images from the PlantDoc dataset on the right are taken from the real world i.e., either from farm or plantation fields. The PlantDoc dataset images contain other leaves, steams, tomato fruits, weeds, and other plants. These images are taken in varied lighting conditions.

4.2.2 Dataset Annotation and Cleansing

Although PlantVillage and PlantDoc datasets were already annotated and cleansed by experts, a high-level exploration was carried out as a part of this research work to ensure the images were labeled properly and there were no wrong annotations. There were no issues observed with the PlantVillage dataset. However, there were some instances of wrong labeling for the PlantDoc dataset which are listed below.

- A Tomato_Bacterial_Spot class folder also contains images for other disease types such as bacterial canker and bacterial speck which were not included in the scope of this research and Septoria leaf spot images.
- There were a couple of images labeled as Early blight that was not correct and show the symptoms of Late blight.
- The image “tomato.early.blightH.jpg” was wrongly labeled as Late blight.
- It was noticed that “early+blight+on+leaf+yel+halo+IMG_2313.JPG” and “tomato-early-blight-e1437503226856.jpg” images were classified as Septoria leaf spot.

The PlantDoc dataset was cleaned to address the above-listed and similar wrong annotation issues before starting data processing, analysis, and model training activities.

4.2.3 Create Cropped Image Dataset

As a part of this research work, multiple CNN models have been trained on both datasets. To eliminate background noise from the PlantDoc dataset, a new "Cropped" image dataset was created using bounding box details which were available in PascalVOC XML format as a part of the PlantDoc Object Detection dataset (Singh et al., 2020). The pascalvoc-to-image Python script ("pascalvoc-to-image · PyPI," 2022) was used to crop the images using bounding box details. As seen in Figure 15 – PlantDoc Cropped Dataset, this dataset contains the cropped images of leaves using the bounding box details.

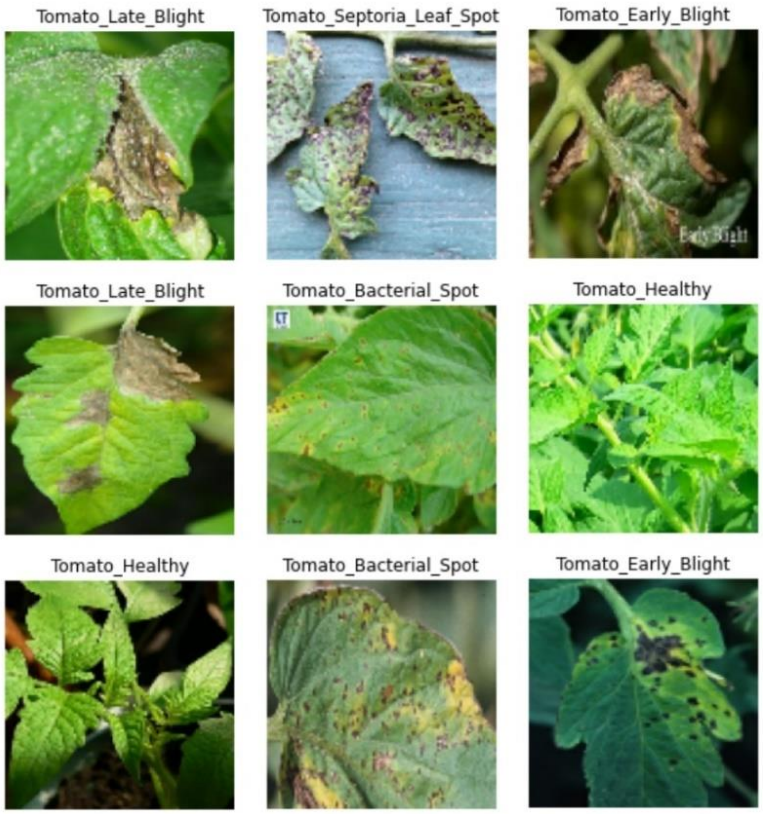


Figure 15 – PlantDoc Cropped Dataset

Although this dataset contains the cropped plant leaves images, still it differs from the PlantVillage dataset. The images in the cropped dataset are captured from different angles, in varying lighting conditions, and with varied sizes.

4.2.4 Create Disease Severity Dataset

One of the important objectives of this research project is to predict the spread of disease after identifying the disease type. To achieve this objective a balanced dataset has been created using plant leaves images from the PlantVillage dataset as the PlantDoc dataset was not containing sufficient images in each class which can be further subclassified based on the spread of the disease. This research shortlisted two common tomato plant disease types to predict the degree of disease spread which includes “Tomato Bacterial Spot” and “Tomato Late Blight.” The images for these disease types were further classified based on the severity levels of the disease into 3 sub-categories, which include – L1: Low Spread, L2: Medium Spread, 3: High Spread. The details of the classes and their respective image counts are provided in Table 6 - Plant-Severity-Dataset Description.

Table 6 - Plant-Severity-Dataset Description

#	Class	Train Image Count	Test Image Count
1	Tomato_Healthy_L0	150	50
2	Tomato_Bacterial_Spot_L1	150	50
3	Tomato_Bacterial_Spot_L2	150	50
4	Tomato_Bacterial_Spot_L3	150	50
5	Tomato_Late_Blight_L1	150	50
6	Tomato_Late_Blight_L2	150	50
7	Tomato_Late_Blight_L3	150	50

4.2.5 Resize and Rescale Datasets (Normalization and Standardization)

Different base models downloaded using Keras Application API expect specific input sizes for images passed for training the model. This research experimented with many base models which are lighter and suitable to deploy on low compute devices such as mobile phones. This research leveraged MobileNet V1, MobileNet V2, EfficientNet B0, and NasNetMobile with ImageNet trained weights. All of these base models can accept the input image size of (224, 224, 3), where width and height are 224 pixels and contains 3 channels (RGB). In addition, transfer learning models based on MobileNet V2 architecture with an input size of (128, 128, 3) have been trained

during the model experimentation to compare the model performance with down-sized input image resolution.

The pixel values for images with RGB channel ranges between 0-255, which needs to be normalized either between 0 to 1 or -1 to 1 depending on the base model used. The rescaling of pixel values in these ranges helps to improve the model training speed due to the reduced complexity of mathematical calculations. As each different base model expects a specific kind of input pre-processing, the respective preprocess_input method from Keras Application API was used (such as "*tf.keras.applications.mobilenet_v2.preprocess_input*" for MobileNetV2 model and "*tf.keras.applications.efficientnet.preprocess_input*" for EfficientNet B0). Then it was passed to the *preprocessing_function* parameter of *ImageDataGenerator* class.

4.2.6 Data Augmentation

The PlantDoc Dataset contains very few images for model training. Hence, it was decided to apply image augmentation to increase the size of the dataset, which can help to reduce the overfitting of the model. Image augmentation makes the trained model more robust to the variations in the images such as images with different angles (sheer range), zoom ratio, and different rotations. To decide on which type of transformations to apply on both datasets, the first few sample images were carefully analyzed. Based on this sample image analysis, it was decided to apply random rotations with a degree of 20, apply sheer range and a zoom range of 20% and apply horizontal flip. The research leveraged *ImageDataGenerator* class from Keras API and applied "just-in-time" image augmentation techniques while training the CNN models. As *ImageDataGenerator* class applied the image augmentation on each of the training batches, it resulted in optimized usage of available memory.

Then the *flow()* method was applied before training the model to save the images to disk during the training process. Figure 16 shows the sample augmented images from PlantVillage and PlantDoc datasets.

Figure 16 – Sample augmented images from PlantVillage (left) dataset and PlantDoc (right) dataset

4.2.7 Training and Test dataset splitting

The last important step of data preparation before training a model on the dataset was splitting the data into train, test, and validation datasets. One of the important objectives of this research is to validate the CNN model with a real-world image dataset which can ensure its adoption in the actual plantation field. To achieve this goal a trained CNN model was tested using real-world images from the PlantDoc dataset. Hence, the PlantVillage dataset was split between Train and Validation datasets which were used during model training and hyperparameters tuning. A hold-out methodology was used to split the PlantVillage dataset by applying an 80% (train) - 20% (test) ratio.

The original PlantDoc dataset was already split between train and test datasets, and it did not require any further processing.

4.3 Exploratory Data Analysis (EDA)

Though this research leveraged the unstructured image data and not the structured data, performing data exploration and analysis helped a lot to gain the required understanding of the dataset characteristics. The EDA was performed to finalize important decisions about data loading, input tensor size to be considered for various CNN models, ideal batch size selection for model training

with PlantVillage and PlantDoc datasets, evaluation of class imbalance ratio, and decide on the model evaluation metrics.

4.3.1 Dataset Class Distribution

The first step of understanding the data distribution for different classes in both datasets can help to understand the degree of class imbalance by iterating through the dataset folder structure using Python code.

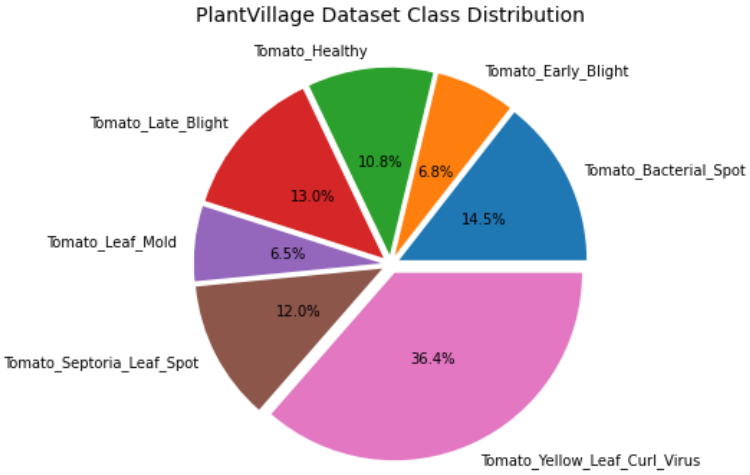


Figure 17 – PlantVillage Dataset Class Distribution

As seen from Figure 17 – PlantVillage Dataset Class Distribution, it was evident that the "Tomato_Yellow_Leaf_Curl_Virus" class contains more than double images as compared to other diseases classes and healthy image class.

Next, the PlantDoc dataset class distribution was analyzed using Python code and visualized using the Matplotlib library. The PlantDoc dataset was already split between the train and test datasets.

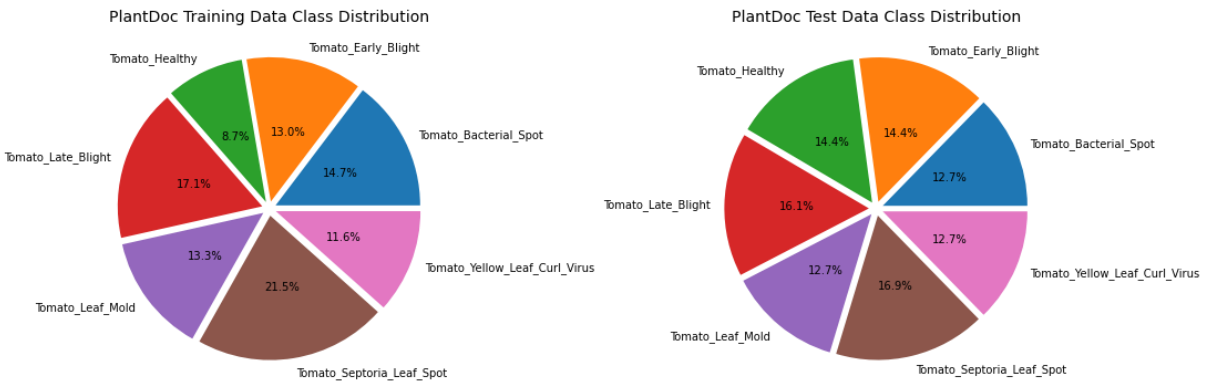


Figure 18 – PlantDoc Dataset Class Distribution

As seen in Figure 18, for the training dataset the number of images for "Tomato_Septoria_Leaf_Spot" is more as compared to other classes. The healthy image class (Tomato_Healthy) contains comparatively a smaller number of images. For the test dataset, there is no noticeable class imbalance and all the classes range between 13% to 17% distribution of image data.

Based on the dataset class distribution analysis, it was decided to create derived "Balanced" datasets for both PlantVillage and PlantDoc datasets. To evaluate model performance against these imbalanced datasets, appropriate metrics were considered such as Precision, Recall, F1 Score, AUC/ ROC, and Confusion Matrix in addition to Accuracy metrics.

4.3.2 Basic Image EDA

After a high-level analysis of PlantVillage Dataset Class Distribution to understand the class imbalance ratio for both the datasets, the next step performed was to conduct EDA on both the datasets to check the basic properties of the image data. The research leveraged the "basic-image-eda" tool to perform this analysis. As the first step of EDA, the PlantVillage original dataset was analyzed by passing a folder path along with other parameters such as file extensions, number of threads, and so on. The results from basic-image-eda tools were considered to decide the input image resolution, resizing, and applying normalization to the image dataset.

```
*-------------------------------------------------------------------------------------*
number of images                        |  14707

dtype                                   |  uint8
channels                                |  [3]
extensions                              |  ['jpg', 'jpeg', 'JPG']

min height                              |  256
max height                              |  256
mean height                             |  256.0
median height                           |  256

min width                               |  256
max width                               |  256
mean width                              |  256.0
median width                            |  256

mean height/width ratio                 |  1.0
median height/width ratio               |  1.0
recommended input size(by mean)         |  [256 256] (h x w, multiples of 8)
recommended input size(by mean)         |  [256 256] (h x w, multiples of 16)
recommended input size(by mean)         |  [256 256] (h x w, multiples of 32)

channel mean(0~1)                       |  [0.3929381  0.4646508  0.44374096]
channel std(0~1)                        |  [0.19917132 0.16221574 0.1823055 ]
*-------------------------------------------------------------------------------------*
```

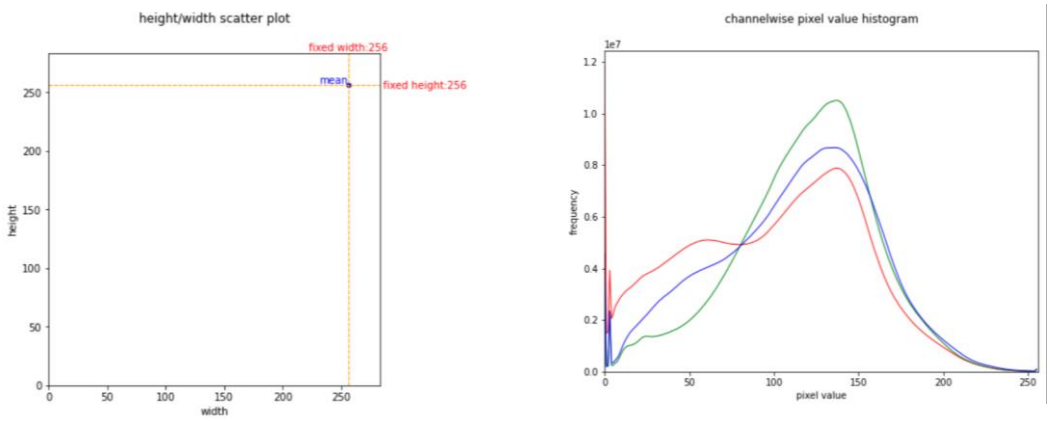


Figure 19 – PlantVillage Dataset EDA Results

As per the Basic EDA analysis from Figure 199, the PlantVillage dataset contains a total of 14707 images representing 7 classes. All the images size is consistent, i.e., 256 pixels width and 256 pixels height with mean and median height/width ratio of 1. The recommended input size for this dataset is 256 pixels X 256 pixels of height and width which is in the multiples of 8, 16, and 32. The mean for the red channel is 0.3929381, for the green channel it is 0.4646508 and for the blue channel, it is 0.44374096. The standard deviation ranges between 0.16 to 0.20.

The scatter plot of height/ width shows that all the PlantVillage dataset images have identical height and width of 256 pixels and hence mean is also 256 pixels. As seen from the channel-wise pixel value histogram, all three channels have a peak between 100 pixels to 150 pixels. The green channel shows the highest intensities which represents the plant leaf in the center of each image. The red and blue pixel intensities represent the background of the images. The histogram contains a slight peak at the beginning for a blue and red channel which might be due to darker backgrounds for few

images. Their frequency is lower in the pixel range of 200 to 256 indicating no white color dominance in any of the images.

The basic EDA was also conducted for PlantVillage train and test datasets after applying the hold-out strategy with the 80-20 split ratio. The results of the basic EDA tool were almost identical with only a difference in the number of images as seen in Table 7.

Table 7 – PlantVillage Train and Test Dataset EDA

PlantVillage Train Dataset Basic EDA		PlantVillage Test Dataset Basic EDA	
number of images	11766	number of images	2940
dtype	uint8	dtype	uint8
channels	[3]	channels	[3]
extensions	['jpg', 'JPG']	extensions	['jpg', 'JPG']
min height	256	min height	256
max height	256	max height	256
mean height	256.0	mean height	256.0
median height	256	median height	256
min width	256	min width	256
max width	256	max width	256
mean width	256.0	mean width	256.0
median width	256	median width	256
mean height/width ratio	1.0	mean height/width ratio	1.0
median height/width ratio	1.0	median height/width ratio	1.0
recommended input size(by mean)	[256 256] (h x w, multiples of 8)	recommended input size(by mean)	[256 256] (h x w, multiples of 8)
recommended input size(by mean)	[256 256] (h x w, multiples of 16)	recommended input size(by mean)	[256 256] (h x w, multiples of 16)
recommended input size(by mean)	[256 256] (h x w, multiples of 32)	recommended input size(by mean)	[256 256] (h x w, multiples of 32)
channel mean(0~1)	[0.39293888 0.4646221 0.44371736]	channel mean(0~1)	[0.39294052 0.46476522 0.44387266]
channel std(0~1)	[0.19945994 0.16254409 0.18266569]	channel std(0~1)	[0.19800426 0.16090317 0.18082581]

After completing the EDA of the PlantVillage dataset, the "basic-image-eda" tool was leveraged to perform image data analysis for the PlantDoc dataset. As the original downloaded PlantDoc dataset was already split into train and test datasets, the tool was executed by passing the folder path of the train and test datasets separately. The results of basic EDA tool execution are shown in Table 8 followed by the height/ width scatter plots and channel-wise pixel value histograms for PlantDoc Train and Test datasets.

Table 8 – PlantDoc Train and Test Dataset EDA

PlantDoc Train Dataset Basic EDA		PlantDoc Test Dataset Basic EDA	
number of images	585	number of images	118
dtype	uint8	dtype	uint8
channels	[3, 4]	channels	[3, 4]
extensions	['png', 'jpg']	extensions	['jpeg', 'jpg']
min height	85	min height	137
max height	4000	max height	4032
mean height	809.1111111111111	mean height	771.3898305084746
median height	598	median height	567
min width	115	min width	150
max width	6000	max width	6000
mean width	957.6974358974359	mean width	923.0508474576271
median width	667	median width	660
mean height/width ratio	0.8448504514924507	mean height/width ratio	0.8356959236136614
median height/width ratio	0.896551724137931	median height/width ratio	0.8590909090909091
recommended input size(by mean)	[808 960] (h x w, multiples of 8)	recommended input size(by mean)	[768 920] (h x w, multiples of 8)
recommended input size(by mean)	[816 960] (h x w, multiples of 16)	recommended input size(by mean)	[768 928] (h x w, multiples of 16)
recommended input size(by mean)	[800 960] (h x w, multiples of 32)	recommended input size(by mean)	[768 928] (h x w, multiples of 32)
channel mean(0~1)	[0.43573222 0.5188795 0.34979722]	channel mean(0~1)	[0.42644852 0.5061471 0.31287658]
channel std(0~1)	[0.24791934 0.2416777 0.2561991]	channel std(0~1)	[0.23238924 0.22404481 0.22157168]

Figure 20 shows the height and width scatter plot and channel-wise pixel value histogram for the PlantDoc train dataset.

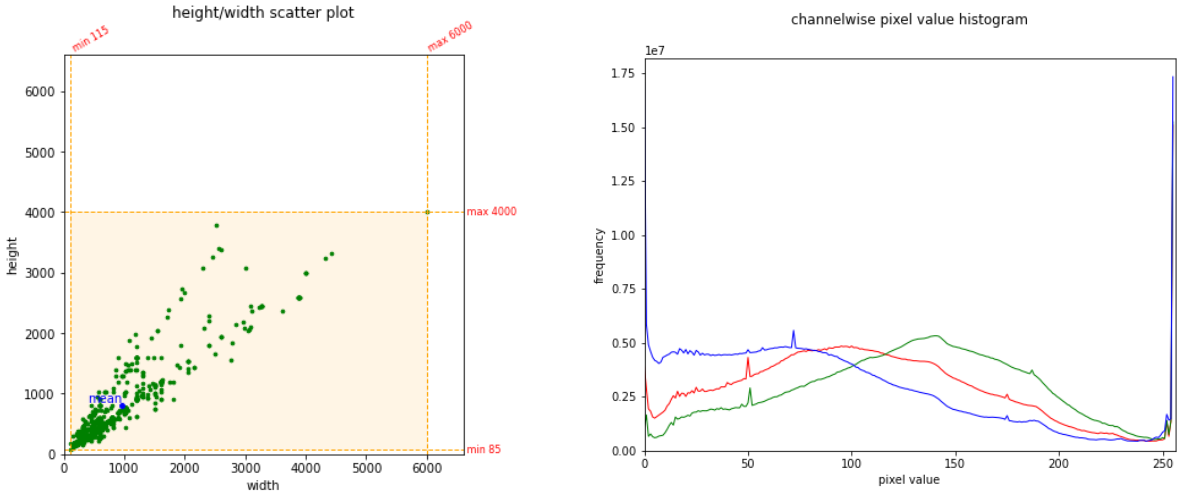


Figure 20 – PlantDoc Train Dataset EDA Results

Figure 21 shows the height and width scatter plot and channel-wise pixel value histogram for the PlantDoc test dataset.

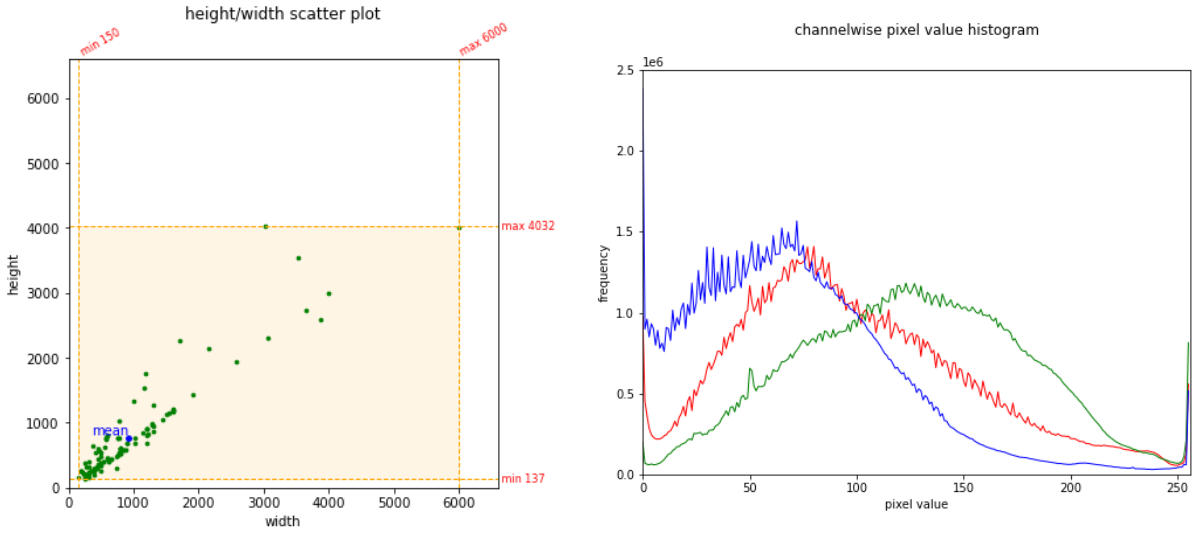


Figure 21 – PlantDoc Test Dataset EDA Results

Based on the analysis of basic image EDA results, it is clear that the PlantDoc train dataset contains images from varied resolutions. The height ranges from 85 to 4000 pixels with an average (mean) height of 809.11 pixels and a median value of 598 pixels. The width of the PlantDoc train dataset ranges between 115 to 6000 pixels with a mean value of 958 and a median value of 667. The

recommended input size based on the mean in multiples of 8, 16, and 32 is between 800 pixels for height and 960 pixels for width. The height/ weight scatter plot for the PlantDoc train dataset shows that most images are having height and width between 100 to 1000 pixels and some of the images fall within a range of 1000 to 2000 pixels. The number of images having more than 2000 pixels of height and width is very less and there are hardly a few images having more than 4000 pixels of height and width. The channel-wise pixel values histogram for PlantDoc training data indicates that all three channel pixel intensities are widely distributed from 0 to 255 pixels. The pixel frequency for the red and blue channels starts decreasing after 100 pixels and for the green channel, it starts decreasing from a 150-pixel value. The peak can be observed for a red channel at a 100-pixel value. For the green channel, the highest peak can be noticed at a 140-pixel value. The blue channel shows the peak at a 75-pixel value and then there is a steep increase in the frequency at 255, which is representing dark background details in the image.

As per the results of the image basic EDA for the PlantDoc test dataset, the image height varies from 137 to 4032 pixels, and the width ranges from 150 to 6000 pixels. The mean (average) height is 771 pixels, and the mean width is 927 pixels. The height median value is 567 and the width median value is 660. The recommended input size is 768 pixels in height and 928 pixels in width by the basic image EDA tool. As observed in the height/ width scatter plot for the test dataset, the majority of the image's height and width ranges between 100 pixels to 1000 pixels. There are very few images with a height and width of more than 2000 pixels. The channel-wise pixel values histogram for PlantDoc training data analysis shows that blue and red channels skewed towards the left indicating dark blue backgrounds such as sky or soil. The peak value for blue and red channels ranges between 50 to 100 pixels indicating darker shades of these colors. The green channel shows the widely distributed spread from 0-255 with a sudden steep frequency increase at 255-pixel value. The channel histogram represents the plant leaves dominance in the images. The peak for a green channel can be seen at a 130-pixel value.

4.4 Class Balancing

Based on the class distribution analysis, it was clear that both datasets have a considerable degree of class imbalance. However, the PlantVillage dataset which was generated from a controlled environment already contains the augmented images. Hence, it was decided not to apply the oversampling technique for this dataset, which can result in duplicate images causing overfitting

of the model. As the PlantVillage dataset contains a sufficient number of images (14707), it was decided to evaluate the CNN classifier performance before applying any oversampling technique and check if the CNN model can handle the class data imbalance for this dataset. It was decided to first monitor the performance of the CNN model with classification metrics other than Accuracy (such as Precision, Recall, AUC/ROC, and Confusion Matrix). As the model performance was not optimal, class balancing techniques were applied.

For the PlantDoc dataset, it was decided to handle the class imbalance using two types of techniques, which include oversampling and undersampling. Oversampling was applied using image augmentation techniques for selected classes which were having fewer images for only the train dataset. The images were augmented using the OpenCV image library. The oversampling leveraged OpenCV and Python scripts. Table 9 shows the comparison between the original PlantDoc Train dataset and oversampled balanced dataset.

Table 9 – PlantDoc Balance Dataset (Oversampled)

Class Name	**Train Image Count**	
	Original	**Updated**
Tomato_Bacterial_Spot	86	500
Tomato_Early_Blight	76	500
Tomato_Healthy	51	500
Tomato_Late_Blight	100	500
Tomato_Leaf_Mold	78	500
Tomato_Septoria_Leaf_Spot	126	500
Tomato_Yellow_Leaf_Curl_Virus	68	500

As another experiment, the research has also created a new balanced dataset version using under-sampling techniques. The images were randomly removed from the PlantVillage and PlantDoc train datasets using Python script to match with the lowest class image count. Table 10 shows the original image count and the updated reduced image count in the balanced datasets.

Table 10 – PlantDoc Balance Dataset (Under sampled)

Class Name	**PlantVillage**		**PlantDoc**	
	Original	**Updated**	**Original**	**Updated**

Tomato_Bacterial_Spot	2127	952	86	51
Tomato_Early_Blight	1000	952	76	51
Tomato_Healthy	1591	952	51	51
Tomato_Late_Blight	1909	952	100	51
Tomato_Leaf_Mold	952	952	78	51
Tomato_Septoria_Leaf_Spot	1771	952	126	51
Tomato_Yellow_Leaf_Curl_Virus	5357	952	68	51

4.5 CNN Model Design and Implementation

This research leveraged a supervised transfer learning technique to train a CNN classifier that can identify six types of common tomato plant diseases along with healthy plant leaves. The below subsections explain the implementation details for the model training process flow explained in Figure 7.

4.5.1 Feature Engineering

After completing data loading, data pre-processing, and data splitting into train and test datasets, the next important step in model training workflow is feature engineering. For CNN models, feature extraction and selection steps are managed automatically. However, for Disease Severity Predictions, feature extraction and selection were done manually, which is explained in the further subsection 4.8 Disease Severity Prediction - Design and Implementation which covers the details of the Disease Severity Prediction model.

4.5.2 CNN Model Training using Transfer Learning

In this research, the CNN classifier model is trained using a transfer learning approach. This research leveraged pre-trained state-of-the-art modern CNN mobile architectures as a feature extractor and then further fine-tuned those models to adapt to the PlantVillage and PlantDoc datasets and their derived variations.

4.5.2.1 Base model selection

As the final solution was planned to integrate with the Android mobile application for real-time inferencing in the tomato plantation field, the research focused on the lighter CNN model architectures which are suitable for deployment on low compute devices. Based on the detailed

literature study conducted, four different CNN model architectures were selected as a base model for transfer learning and their different variations were used.

1. MobileNet V1
2. MobileNet V1 (50% parameters)
3. MobileNet V2
4. MobileNet V2 (50% parameters)
5. EfficientNet B0
6. NasNetMobile

Keras Application API allows setting the *alpha* value for MobileNet class of models which can decide how many parameters to be used in various layers. This hyperparameter was helpful to reduce the model size which helps to update the deployed model on mobile phones using an internet connection.

4.5.2.2 Feature Extractor CNN model

Keras Application API was used to load the base model along with the ImageNet trained weights. The classification layer for the base CNN model was removed by setting the *include_top* parameter to false and the custom classification layer was defined as per the requirements of this research work. The transfer learning approach leveraged the pre-trained model on the ImageNet dataset as a convolutional base which was used to train the Feature Extractor model on the PlantVillage dataset. This research leveraged four different CNN models during experimentation to train the feature extraction models and their performance details are discussed in CHAPTER V: RESULTS AND DISCUSSIONS.

4.5.2.3 Fine-tuned CNN model

The pre-trained CNN model with ImageNet weights was used to extract generic features, such as edge detection and color intensity using early (left-most) layers of the network. With the gradual model evolution, the later layers of the network were used to identify more complex features such as texture. Finally, the classification layers at the last were leveraged to identify class-specific features i.e., detection of specific tomato plant disease types.

The plant leaves datasets used in this research had a considerable difference from the ImageNet dataset. Furthermore, the PlantDoc dataset contains very few images. To handle these dataset differences and to adopt the different problem domain of plant disease detection, it was decided to retrain some of the later layers of the pre-trained CNN model. The selected base networks are divided into multiple blocks. The model architecture design followed the approach to gradually unfreeze the layers starting from the last block. After retraining layers from the last block, the model performance was observed and compared again by unfreezing some more layers from the next immediate previous block. The number of layers was carefully chosen from the point when model performance stopped improving.

4.5.3 CNN Model Architecture

This research work leveraged pre-trained CNN models which are suitable to deploy on mobile which accept the input image of size 224 X 224 with three (RGB) channels. Although multiple experiments were conducted to identify the best suitable model with variations applied to network design, this section explains the architecture for the "best" selected model based on MobileNet V2 pre-trained model. CHAPTER V:

RESULTS AND DISCUSSIONS explains evaluation metrics and criteria being used to select the best model.

MobileNet V2 network design consists of a series of Depthwise Separable convolutions with Depthwise (3×3), and Pointwise (1×1) layers followed by batchnorm (BN) and ReLU. In the updated model architecture classification layer was updated, but no changes were made in the "Convolutional Base" as shown in Figure 22.

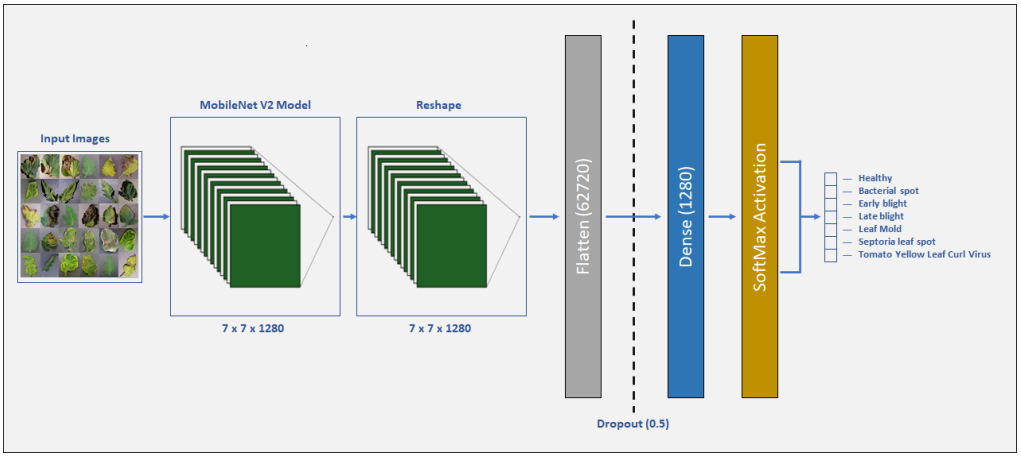


Figure 22 – MobileNetV2 (Final) model updated architecture

The *GlobalAveragePooling2D* layer was added which flattens the output (7 X 7 X 1280 = 62720) returned by the MobileNetV2 model's last Convolutional layer. To control the overfitting and ensure better generalization of the model, the *Dropout* layer is added with a rate of 0.5. A new *Dense* layer was created with an output size of 1280. Finally, the original *SoftMax* activation function with 1000 classes was replaced with a new *SoftMax* Activation layer with seven classes (6 disease types + healthy). A model summary from Figure 23 shows these layers, their output shape, and the number of parameters.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_24 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
sequential_1 (Sequential)    (None, 224, 224, 3)       0
_________________________________________________________________
tf.math.truediv_11 (TFOpLamb (None, 224, 224, 3)       0
_________________________________________________________________
tf.math.subtract_11 (TFOpLam (None, 224, 224, 3)       0
_________________________________________________________________
mobilenetv2_0.50_224 (Functi (None, 7, 7, 1280)        706224
_________________________________________________________________
global_average_pooling2d_1 (Glo (None, 1280)           0
_________________________________________________________________
dropout_1 (Dropout)             (None, 1280)           0
_________________________________________________________________
dense_1 (Dense)                 (None, 1280)           1639680
_________________________________________________________________
predictions (Dense)             (None, 7)              8967
=================================================================
Total params: 2,354,871
Trainable params: 2,300,327
Non-trainable params: 54,544
_________________________________________________________________
```

Figure 23 – MobileNetV2 (Final) model Input and Output Layers

The best model contains of total 2,354,871 parameters (706,224 + 1,639,680 + 8967) out of which 2,233,511 parameters are trainable and 121,360 parameters are non-trainable.

During the experimentation, various combinations of layers were tried for the classification layer which includes the addition of one more *Dense* layer between the *GlobalAveragePooling2D* layer and the *Dropout* layer. Also, model performance was verified by changing various dropout rates such as 0.2, 0.3, and 0.4 before finalizing the dropout rate of 0.5.

4.5.4 Hyperparameter selection and tunning

After finalizing the CNN model architecture, various models were trained using PlantVillage and PlantDoc original dataset, cropped dataset, and balanced dataset using this CNN model. The model performance was evaluated against the validation dataset and the hyperparameter was tuned to improve model performance. This section covers the details of hyperparameters tuned during the research experimentations.

1) Image input size

Based on the analysis of basic EDA, it was clear that the PlantVillage dataset contains all the images with a uniform resolution of 256 X 256 pixels. However, the PlantDoc dataset contains images with varied image resolution where height ranging between 85 to 4000 pixels and width ranging between 115 to 6000 pixels. The MobileNetV2 model was trained with three different input images sizes.

- (224, 224, 3)
- (192, 192, 3)
- (128, 128, 3)

The best performance was achieved with an input image size of (224, 224, 3) which was finalized for the final model training.

2) Batch size

For the PlantVillage dataset, two batch sizes of 32 and 64 were tried, and based on the results achieved it was decided to go with 64 as batch size for this dataset. For PlantDoc original dataset, cropped dataset, and balanced dataset, the batch sizes considered were 32 and 16 due to the smaller number of images in the dataset. It was decided to go with 16 as batch size for all the variants of the PlantDoc dataset based on the performance achieved from multiple model execution experiments.

3) Fine-tune layers

Due to dataset differences, it was critical to set the correct number of layers to fine-tune so that model can learn the complex features which are specific to tomato plant diseases for plant leaves images. This research experimented with different base model architectures which contain different numbers of layers depending on the network design. The approach adopted to fine-tune layers for multiple CNN models is explained in the second subsection of section 4.5.2 CNN Model Training using Transfer Learning. For details on the number of fine-tune layers chosen for different models, please refer to Table 11 – Hyperparameters used for training "Best Model".

3) Optimizer

During the experimentation phase for this research, more than 50 models have been trained and evaluated. The research leveraged multiple optimizers which include SGD, RMSprop, and Adam with different learning rates, decay, and momentum settings. Although, model training speed was improved with the choice of Adam optimizer, based on the model evaluation results it was noticed that model trained with the SGD optimizer generalized best. Hence, it was decided to go with SGD optimizer with a learning rate schedule to apply "learning rate decay" and EarlyStop callback.

4) Loss function

The *CategoricalCrossentropy* loss function was used to compute the cross-entropy loss between the labels and predictions for multi-class CNN classifiers. The research has used *ImageDataGenerator.flow_from_directory()* method to load the image dataset in batches while training the CNN model, which returns 2D one-hot encoded labels. Hence, the *CategoricalCrossentropy* loss function was used for all the models.

5) Learning rate

Learning rate is the most important hyperparameter to tune during the model training process. When a very low learning rate is set then the model will not be able to learn anything meaningful from data and it will also result in a long model training duration as a model. A large learning rate will allow a model to learn faster, but it might result in the sub-optimal final set of weights. During the initial model training process, multiple fixed learning rates were tried out with a combination of different optimizers as the same learning rate will not work best with different optimizers. Even

for the SGD optimizer, the “momentum” parameter was set to 0.9 which adds inertia to the weight update procedure. It causes past updates in one direction to continue in that direction in the future, which can avoid the learning process getting stuck or oscillating. The final selection of learning rates for various models is listed in Table 11.

6) Learning rate decay/ Learning rate schedules

A varying learning rate over a time interval which is called “learning rate decay” or “learning rate schedule” was used instead of a fixed learning rate. This allowed setting the initial learning rate arbitrarily and then adopt the appropriate learning rate based on the learning rate schedule defined by the monitoring validation loss matrix. As an initial experiment, the “decay” setting for the SGD optimizer was set to the initial learning rate divided by the number of epochs which is usually a recommended practice. Thereafter, different values of decay rate were tried, and model performance was analyzed.

As a next step, the researchers used the Keras Callbacks API to set the learning rate schedule. A *ReduceLROnPlateau* callback was set to adjust the learning rate when a plateau in model performance is detected, i.e., no reduction in validation loss for 10 training epochs. This callback reduced the learning rate when model performance stops improving.

The *LearningRateSchedule* from Keras Learning rate schedule API was then leveraged to apply an exponential decay schedule. To lower the learning rate while model training, it applied the exponential decay function to an SGD optimizer step based on the initial learning rate of 0.0001. The decay steps were set to 10000 and the decay rate to 0.96. Based on the satisfactory results obtained by using *keras.optimizers.schedules.ExponentialDecay*, it was decided to use this learning rate schedule for the final model training.

8) Number of Epochs

As the number of images is very less for the PlantDoc dataset, it tends to overfit the model if training is performed for a longer duration with multiple epochs. On the contrary, if a model is trained for too few epochs, it will not learn the accurate mappings between input features and output classes. Hence, deciding on the number of epochs for both feature extraction and fine-tuned models was an important step in the research. During the model training, a different number of epochs were trained, and model performance was analyzed. For the PlantVillage dataset, feature extractor

and fine-tuned models were trained using 10 epochs, which resulted in the best model performance. For the PlantDoc dataset, additional consideration was made to apply the EarlyStopping callback.

9) Early stopping

The early stopping technique was used for the model trained on the PlantDoc dataset. To start with the training process, a large number of training epochs (500) were set. The validation loss matrix was monitored, and training was stopped once the model performance was not improving on a hold-out validation dataset. Keras callbacks API provides support for early stopping of training using an EarlyStopping callback method. The patience parameter was set to 20 to wait for those many epochs before stopping the training. The *restore_best_weights* setting was set to "True," which allowed restoring model weights from the epoch with the best value of the monitored quantity i.e., epoch with lowest val_loss.

The various combinations of hyperparameters tunning were tried out and the best hyperparameters settings with optimal model performance were finalized, which are listed in Table 11 – Hyperparameters used for training "Best Model".

Table 11 – Hyperparameters used for training "Best Model"

Phase	Hyperparameter	MobileNetV1	MobileNetV2	EfficientNetB0	NasNetMobile
Feature Extraction	Input image size	(224, 224, 3)	(224, 224, 3)	(224, 224, 3)	(224, 224, 3)
	Optimizer	SGD (Default)	SGD (Default)	SGD (Default)	SGD (Default)
	Loss Function	Categorical Crossentropy	Categorical Crossentropy	Categorical Crossentropy	Categorical Crossentropy
	Learning Rate	0.1	0.1	0.1	0.1
	Decay LR	NA	NA	NA	NA
	Early Stopping	No	No	No	No
Fine Tuning	Input image size	(224, 224, 3)	(224, 224, 3)	(224, 224, 3)	(224, 224, 3)
	Optimizer	SGD	SGD	SGD	SGD
	Loss Function	Categorical Crossentropy	Categorical Crossentropy	Categorical Crossentropy	Categorical Crossentropy
	Learning Rate	0.0001	0.0001	0.0001	0.0001
	Decay Type	Exponential	Exponential	Exponential	Exponential
	Decay Steps	10000	10000	10000	10000
	Decay Rate	0.9	0.9	0.9	0.9
	Early Stopping	Yes	Yes	Yes	Yes
	Fine-tuned layers	50	94	20	173

4.6 CNN Model Evaluation

During the research implementation phase, multiple CNN classifier models were built leveraging selected SOA base models suitable for mobile devices. In addition, models were trained using different datasets which include PlantVillage original and balanced datasets and PlantDoc original, cropped, and balanced datasets. CNN models were evaluated using various classification metrics. A confusion matrix was used to analyse true positives, false positives, true negatives, and false negatives. Based on the confusion matrix, various other matrices were calculated such as accuracy, precision, and recall (sensitivity), precision-recall curve, AUC, and F1-score.

The evaluation metrics for trained CNN models were compared and the best CNN model was selected for deployment using the Android mobile application.

4.7 CNN Model Deployment using Android App

The final stage after performing the final CNN model evaluation against the test dataset was to create the TensorFlow Lite (.tflite) model and integrate the model into the Android application. This research leveraged the TensorFlow Python API to create the TFLite model. A *tf.lite.TFLiteConverter.from_keras_model()* method was called to create a converter object by passing the Keras model and then the model was converted using the TFLite Converter object. After the creation of the TFLite model, it was imported into the SmartCropCNNLite android mobile application using Android Studio. A SmartCropCNNLite provided two options to capture the images which can then be passed to the CNN model to predict the disease type. The first option allowed the user to start the camera from the Android device. The *Intent* class from Android Camera API is used to invoke the default camera application in Android devices. The capture image byte array was then converted to Bitmap and passed to the CNN model to detect the disease type. Also, permission to use a camera is requested by adding an appropriate declaration in the Manifest of the Android Application. Another method provided allowed users to select the image from the Image Gallery and perform the inferencing using the CNN model.

4.8 Disease Severity Prediction - Design and Implementation

After the classification of tomato plant diseases using the CNN classifier model, the next important step was to predict the disease spread, which can play an important role to decide on the appropriate

treatment recommendation for the tomato plant diseases. The research tied two approaches to predict the probabilities for disease spread and accordingly further classify the images into additional subclasses (L1, L2, and L3). In the first approach, manual feature extraction was performed, and traditional ML classification algorithms were tried to classify the images based on plant disease and the degree of disease spread. In the second approach, the research leveraged the CNN model with transfer learning for automatic feature extraction and selection and then trained a classification model to classify the images based on the disease type and disease severity. Both the classifier models were then evaluated, and the final model was selected based on the model performance.

The process flow of the Disease Severity Detection model is explained in Figure 24.

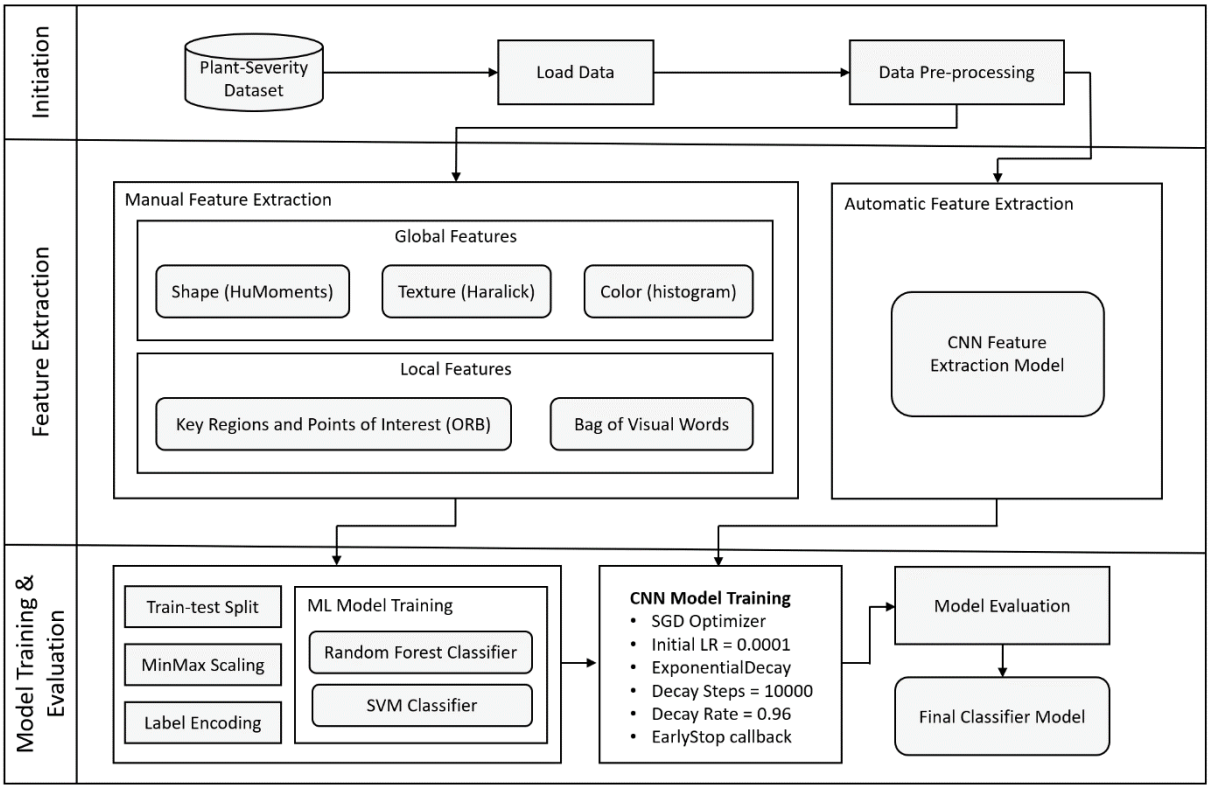


Figure 24 - Disease Severity Detection Process Flow

The next subsection explains the implementation details for both approaches.

4.8.1 Approach 1: Traditional ML model to detect Plant Disease Severity

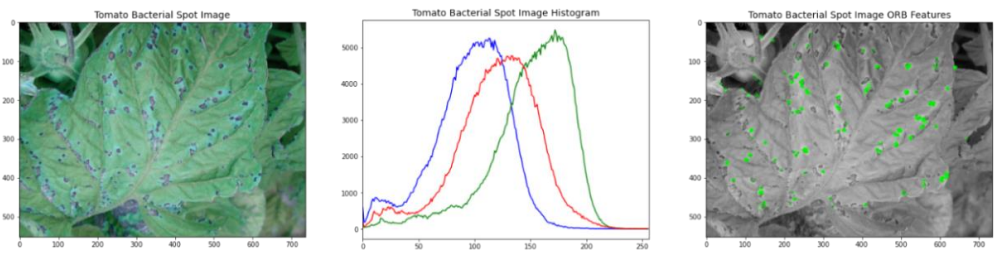
In this approach, the derived dataset for Plant-Disease-Severity was used to extract color, shape, and key points feature details from the images. The tomato plant diseases result in color changes such as turning the leaves into yellow or brown from green, shape deformations such as curling or

molding of the leaves and showing the symptoms in form of brown spots on patches on the leaves. The research has leveraged two types of features to identify plant disease symptoms and their severity – (1) the global features which include color and shape (2) the local feature containing key regions and points of interest details.

OpenCV is used to extract color histogram and shape features to calculate HuMoments. The *mahotas.features.haralick* method from the mahotas image processing library was used to extract texture features. Then these global features were combined, and feature vectors were created using the NumPy package from the Scikit-learn library. After feature extraction, those were standardized using *MinMaxScaler* and divided into train (0.8) and test (0.2) datasets. ML models were trained using Support Vector Machine (SVM) and Random Forest Classifiers on combined Global Features Train dataset and evaluation was carried out against Test dataset to compare the model performance.

As a next step, keypoint and feature descriptor details were extracted using ORB features. After extracting ORB features that describe keypoint at a particular location in the image, a bag-of-visual words (BOVW) technique was applied to get a uniform representation of feature vectors computed by ORB features. A MinMaxScaler was applied to these extracted ORB features and then the KMeans clustering technique was used to group the key points using 120 clusters that represented an image. The computed centroids by the KMeans algorithm were used as the visual dictionary's vocabularies. Labels were encoded using LabelEncoder and then the dataset was divided into train and test. SVM classifier and Random Forest classifier were used on this dataset to classify those images in seven classes. The model performance was analyzed and based on that; it was decided to also experiment with the CNN model to detect the severity of the tomato plant diseases.

Figure 25 shows sample plant leaves images (left), the calculated histogram for BGR channels (middle) and ORB keypoint indicated (right) for all the classes.



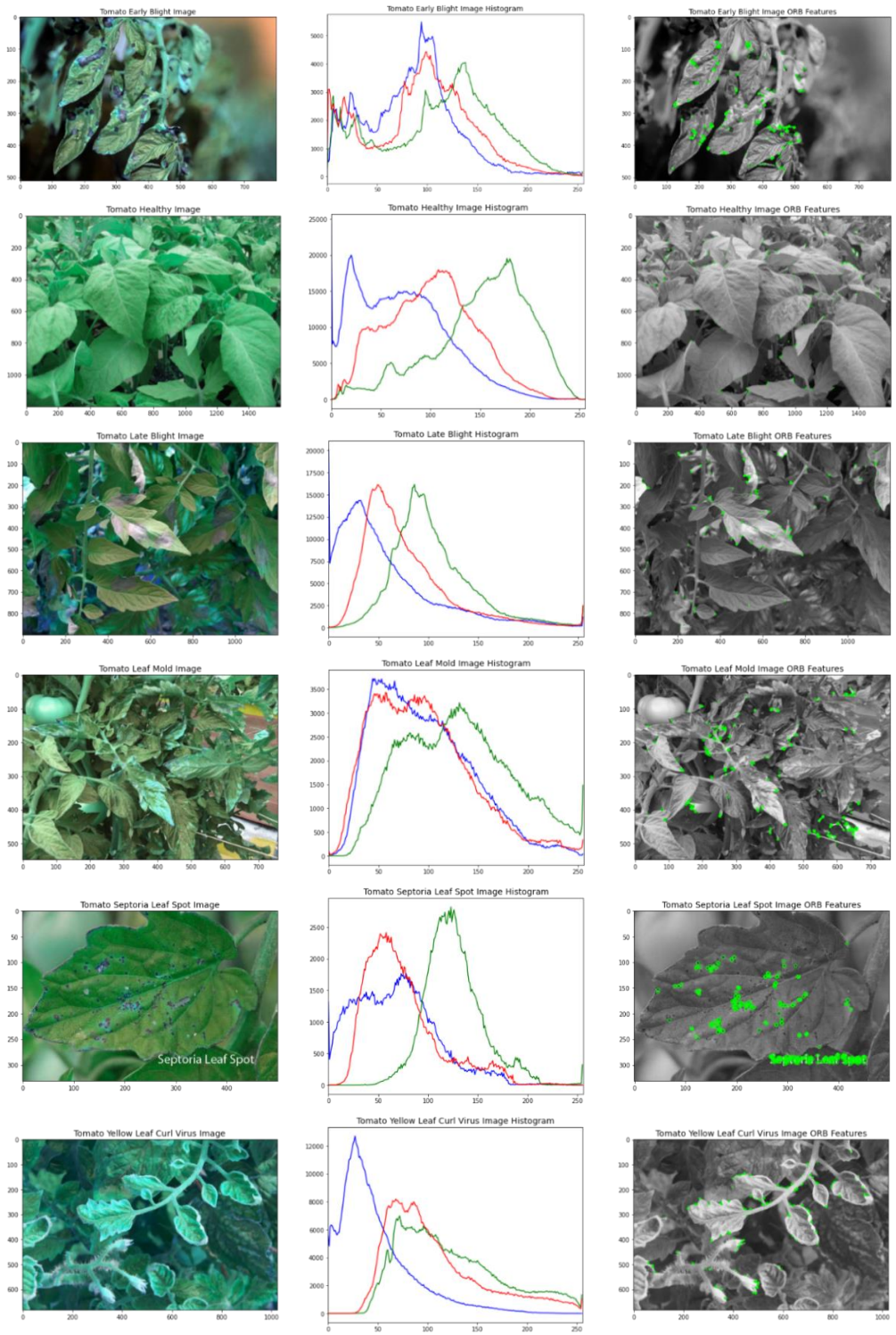


Figure 25 – Plant Images with Color histogram and ORB interest keypoint identified

In the sample image of Bacterial Spot Image, small (less than 1/8 inch) water-soaked circular or even irregular spots can be noticed. Those spots are initially yellowish green which turns brown with aging. The color histogram with BGR channels shows that blue color pixels are concentrated

between pixel value range of 50 to 150, which is contributed by the dark background. The red color pixels show a peak between 100 to 150 which represents the brown and reddish bacterial spots on the leaf. The green color peak can be noticed from 150 to 200 which is primarily due to the light green color of the leaves. ORB features correctly identified the key points in the images, which are the bacterial spots on the leaf. The green channel shows wide distribution with a peak at 150 representing leaves color in the image. ORB features are indicating the disease area of the plant leaves as key interest points.

The early blight sample image shows small brown lesions and some enlarged concentric rings in a bull's-eye pattern in the center of the disease area. The red and blue channel displays a peak at pixel-value 100 indicating the disease-affected areas. The histogram of healthy plant leaves shows the green channel is widely distributed but skewed towards higher pixel values at the right which indicated the light-green color of plant leaves. The blue and red channel distribution is skewed towards the left which represents soil and dark background colors in the image. ORB features have not detected any key point on the plant leaves indicating an absence of any noticeable area or region on the plant leaves. Tomato late blight sample images show enlarged water-soaked areas which are of greenish-black color. The color histogram indicates that all three channels are left-skewed indicating the dominance of dark colors in the image. The ORB features images show the identified key interest areas representing the diseased areas of the plant foliage.

The Tomato leaf mold sample image shows the yellow spots on the upper surface of the leaves. The color histogram shows the wide distribution for green color pixels with a peak of 150 which might be because of the yellow spots on the leaves. The blue and red channels show peaks between 50-pixel and 100-pixel values indicating the solid and background objects. The ORB features are highlighting the yellow spots and molds on the edges of the leaves. The sample image of the Septoria leaf spot image contains many small circular spots with dark brown margins. The blue and red color pixel distribution in the histogram is skewed towards the left because of these dark spots on the leaf and some background shades. The green channel shows the peak at the center of pixel value i.e., at 125 pixels. The ORB image highlights these spots accurately with exception of also indicating the text “Septoria Leaf Spot” written in white at the bottom right-hand side of the image. The Yellow leaf curl virus leaves show yellow leaf edges with upward leaf curling which results in reduced leaf size. The color histogram shows all the channels are skewed towards the left

which might be due to dark background colors in the image. The ORB features show the key points on the edges of the leave accurately which are the affected areas of the leaves due to the disease.

4.8.2 Approach 2: CNN with Transfer Learning to detect Plant Disease Severity

As the performance of manual feature extraction and models trained with Global (Color, Shape, and Texture) and Local (ORB key points) separately were not optimal, it was decided to leverage the CNN model to perform automatic feature extraction and train the CNN classifier model using Plant Severity dataset. Since the number of images was less, it was decided to use the Transfer Learning techniques instead of training a shallow (from scratch) CNN model. MobileNetV2 model architecture was selected as a base model and two models were trained on the Plant Severity data set, which included – (1) Feature Extraction model (2) Fine-tune Model.

The model performance was analyzed compared to classic ML models trained earlier. Based on the analysis, it was decided to use CNN fine-tune model to be used for final deployment.

4.9 Summary

In a nutshell, this chapter explained the detailed steps carried out during data analysis, model training, and evaluation as a part of this research work. The chapter started with details of the data preparation process which included loading of the dataset and the considerations for image annotations and data cleaning. The next subsection provided details of newly curated datasets, which included Cropped Image Dataset and Plant Severity Dataset. Then the steps of resizing and rescaling the dataset images were discussed. The subsection on data augmentation provided details on data augmentation techniques applied using ImageDataGenerator provided by Keras Data Processing APIs to avoid overfitting of the model due to a smaller number of images. The next section explained how the datasets were divided into train and test data set by applying the hold-out strategy.

The next section on Exploratory Data Analysis (EDA) covered the details of data analysis conducted for different datasets. It started with a high-level EDA to check the class distribution and analyzed the class imbalance ratio which indicated the existence of class imbalance that needs to be addressed. Then EDA was performed on actual images from multiple datasets using the "basic-image-eda" tool. The inferences derived from the data analysis were explained clearly which helped to make important decisions on input image size for the training process, the batch

size for training CNN model, and evaluation matrices to be used during model evaluation. The section on class balancing explained the decisions for handling the class imbalance and the steps performed to create balanced datasets. This subsection also provided dataset descriptions for balanced datasets which included various classes and respective image sizes in those classes and compared those with original datasets.

The section on CNN model design and implementation started with an explanation of the approach for feature engineering including automatic feature extraction and selection using the base CNN model. The following subsection on CNN model training using transfer learning explained the base models selected for this research. Then it explained the CNN model architecture and updates which were applied to the original CNN model network design. The subsection on hyperparameter tunning covered all the hyperparameters and their combinations which were tuned to improve the model performance. This section also provided the details of final hyperparameter settings for different CNN models in a table which included the selection of SGD optimizer with a learning rate of 0.0001, learning rate schedule using ExponentialDecay, and EarlyStop callback applied to avoid overfitting of the model. After model training, the next section provided details of how the various CNN models were evaluated using different classification metrics such as confusion matrix, precision, recall, F1-score, and AUC/ROC in addition to Accuracy. The final model was selected based on the comparative analysis of various model performances.

Finally, the last section covered the details of plant disease severity model training which includes two approaches. The first approach performed manual feature extraction for global features (color, shape, and texture) and local features (ORB key points and descriptors). The subsection covered details on how different ML classification models were trained such as Random Forest Classifier and Support Vector Machine (SVM) and model evaluation. The next subsection covered the second approach, in which the CNN model was used to perform automated feature extraction, and the model was trained using transfer learning. The CNN model evaluation process was explained later, and it concluded with a final decision of model selection to detect plant disease severity.

CHAPTER V:

RESULTS AND DISCUSSIONS

5.1 Introduction

This chapter will explain the results of all the experiments conducted during this research and will discuss the interpretation of the evaluation results. The chapter will begin with a section on the approach followed for the evaluation of CNN models. The further subsections will provide details of evaluation metrics for various CNN models trained using different base models, which include MobileNet V1, MobileNet V2, EfficientNet B0, NasNetMobile, and ResNet50 V2. After analysis of individual CNN models, the next section will explain the process of the final model selection, where the performance of CNN models against the PlantDoc dataset with various base models will be compared. The following subsection will also provide details of model sizes which play an important role in the selection of the final model with a goal of final model deployment to the mobile platform. The next section will explain the evaluation results using the final selected model against the PlantDoc test dataset. The section on the evaluation of the Plant Disease Severity model will provide details of model performance evaluation for classical ML models and will compare those with CNN models. Finally, the last section in this chapter will discuss the results obtained while making inferences using the Android Mobile application in the real plantation field.

5.2 CNN Models Evaluation and Results

During the research implementation phase, the first CNN model with transfer techniques was used as the Feature Extractor model by updating the classification layer of network architecture. This model was trained initially on PlantVillage (PVD) original dataset. The performance of the Feature Extractor model was evaluated by comparing training and validation accuracy and loss values. Based on the performance analysis, various hyperparameters were tweaked such as selection of optimizer, learning rate, and the number of epochs. After getting a satisfactory performance for the Feature Extractor model, a CNN model was fine-tuned by gradually unfreezing the last layers of the network. The performance of the fine-tuned model with the PVD dataset was monitored using a comparison of training and validation accuracy and loss. Next, the model performance was verified against a PVD test dataset using predict and evaluate methods. The research used various performance metrics to evaluate model performance, which included accuracy, precision, recall, AUC/ROC, Precision-Recall-Curve, and F1-Score. The classification report was generated to

understand the performance for each class in the dataset for these metrics. Finally, a confusion matrix was generated and plotted using Heatmap to compare the True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values.

After ensuring optimal performance for Fine-tuned model against the PVD dataset, the same model was evaluated using PlantDoc (PD) original dataset. As expected, model performance was drastically degraded against the PD dataset due to the difference between the controlled environment (PVD Dataset images) and images that were taken from the actual farm field. Then model containing ImageNet + PlantVillage (PVD) trained weights was fine-tuned against the PD dataset which showed significant model performance improvements. There were additional CNN models trained against PlantVillage Balanced Dataset (PVD-BAL) and PlantDoc Balanced Dataset (PD-BAL) and their performance was evaluated using similar metrics. Next, CNN models trained against PlantDoc Cropped Dataset were evaluated and their performance was compared with the CNN model trained against the original PlantDoc dataset.

As both PVD and PD original datasets are imbalanced, Precision, Recall, and Precision-Recall-Curve are important measures to evaluate model performance. This research focused on the identification of Tomato Plant Diseases using the CNN models and therefore high precision score indicates diseases are classified accurately by the model. A high recall score indicates the model is predicting most disease classes. The preferred model needs to have high precision and high recall values which is indicated by a higher value of precision-recall-curve (PRC) and high F1-Score which indicates a balance between the precision and recall values. This indicates the model is returning all positive (disease classes) cases accurately. This will ensure tomato plant diseases are identified in the early stage accurately. Further, it will help to avoid any excessive use of fertilizers, which not only impacts the crop yield but also adversely impacts the soil quality.

5.2.1 MobileNet V1 model with 100% parameters

CNN models built using MobileNet V1 architecture will 100% parameters (*alpha=1*) were evaluated against various datasets and using different performance metrics. Table 12 shows the evaluation results for these evaluations.

Table 12 – MobileNet V1 (100%) Model Evaluation Results

Dataset	Performance Evaluation Metrics					
	Accuracy	**Precision**	**Recall**	**F1-Score**	**PRC**	**AUC**
PVD (Train & Test)	0.99	0.99	0.99	0.99	1.00	1.00
PVD (Train) + PD (Test)	0.23	0.24	0.23	0.24	0.26	0.64
PD (Train & Test)	**0.66**	**0.65**	**0.66**	**0.64**	**0.76**	**0.93**
PVD-Bal (Train & Test)	0.98	0.98	0.98	0.98	1.00	1.00
PVD-Bal (Train) + PD-Bal (Test)	0.22	0.23	0.22	0.22	0.22	0.62
PD-Bal (Train & Test)	0.63	0.65	0.63	0.62	0.69	0.89
PD-Cropped (Train & Test)	0.63	0.64	0.63	0.62	0.71	0.93

The CNN model trained on PVD original dataset showed high accuracy, precision, recall, and F1-score values of 0.99. The AUC and PRC area were almost 1.00 because the images are uniform in size and do not contain any background noise. When the fine-tuned model trained with PVD dataset was used to predict against PD dataset, it showed significant performance decrease with accuracy and recall values of 0.23 and precision and F1-Score of 0.24. After re-training a model against the PD dataset by fine-tuning the last 50 layers, model performance was improved. The fine-tuned model was able to achieve higher accuracy (0.66), precision (0.65), recall (0.66), and f1-score (0.64). The AUC area was 0.93 and the PRC area was 0.76. Similar evaluation results were observed for PVD and PD balanced datasets. The result shown for balanced datasets were slightly on the lower side as compared to the original dataset. This is due to the reduced number of images for these datasets. The results for the Plant-Doc Cropped dataset showed a slightly lower accuracy, precision, and recall rate resulting in a slightly lower f1-score (0.62), and PRC area (0.71).

5.2.2 MobileNet V1 model with 50% parameters

The MobileNet class of models allowed to control the width of the network, which is mentioned as width multiplier in the MobileNet Architecture paper (Howard and Wang, 2012) using an *alpha* parameter. In this research, CNN models were trained using an *alpha* value of 0.5 which decreased the number of filters in the layers by 50%. This resulted in a smaller size of the trained CNN model which is more suitable for mobile deployment. The model evaluation results for these reduced-size models are listed in Table 13.

Table 13 – MobileNet V1 (50%) Model Evaluation Results

Dataset	Performance Evaluation Metrics					
	Accuracy	**Precision**	**Recall**	**F1-Score**	**PRC**	**AUC**
PVD (Train & Test)	0.98	0.98	0.98	0.98	1.00	1.00
PVD (Train) + PD (Test)	0.25	0.25	0.25	0.25	0.22	0.59
PD (Train & Test)	**0.60**	**0.62**	**0.60**	**0.59**	**0.70**	**0.90**
PD-Cropped (Train & Test)	0.56	0.57	0.56	0.57	0.65	0.90

As expected, the fine-tuned PVD model evaluation resulted in higher accuracy, precision, recall, and f1-score of 0.98 with AUC and PRC reaching a value of almost 1.00. However, the same model did not perform well against the PD dataset and obtained accuracy (0.25), precision (0.25), recall (0.25), f1-score (0.25), PRC (0.22), and AUC (0.59) values were significantly lower. The fine-tuned model with PlantDoc original dataset improved the model performance with an accuracy value of 0.60 and the f1-score of 0.59. AUC score was improved to 0.90 and the PRC area was 0.70. Next, a CNN model with MobileNet 50% parameters was trained using a PD-Cropped dataset. The results were on the lower side for this model as compared to the model trained with the PD original dataset. This indicates the MobileNet V1 model with reduced filters lowered the performance evaluation metrics as compared to the default MobileNet V1 model for the PlantDoc dataset.

5.2.3 MobileNet V2 model with 100% parameters

After model training with MobileNet V1, research experimented with the MobileNet V2 base model which enhanced the MobileNet V1 architecture by using residual connections between the bottleneck layers and by adding shortcut connections between those bottleneck layers (Sandler et al., 2018). Table 14 shows the results achieved using MobileNet V2 default model.

Table 14 – MobileNet V2 (100%) Model Evaluation Results

Dataset	Performance Evaluation Metrics					
	Accuracy	**Precision**	**Recall**	**F1-Score**	**PRC**	**AUC**
PVD (Train & Test)	0.98	0.98	0.98	0.98	1.00	1.00
PVD (Train) + PD (Test)	0.40	0.40	0.39	0.40	0.40	0.72
PD (Train & Test)	**0.67**	**0.66**	**0.67**	**0.65**	**0.74**	**0.92**
PVD-Bal (Train & Test)	0.88	0.92	0.88	0.88	0.94	0.98
PVD-Bal (Train) + PD-Bal (Test)	0.32	0.33	0.30	0.31	0.35	0.69
PD-Bal (Train & Test)	0.64	0.71	0.64	0.64	0.66	0.90

PD-Cropped (Train & Test)	0.63	0.63	0.63	0.62	0.72	0.92

The CNN model trained on the PlantVillage dataset resulted in a high accuracy rate, precision, recall, and f1-score of 0.98 with an AUC & PRC area of almost 1.00. However, the performance of the model degraded when it was used to predict against the PlantDoc dataset. The model was able to achieve an accuracy of only 40% with an AUC area of 72%. Hence, the model was fine-tuned using the PlantDoc dataset which showed significant improvement in model performance. The accuracy was increased to 0.67 with a precision of 0.66 and recall of 0.67. The f1-score achieved using this model was 0.65, the PRC area was 0.74, and AUC increased to 0.92. Then the CNN models were trained using a balanced PD dataset, which showed slightly lower results as compared to the original PlantDoc dataset. The accuracy, recall, and f1-score were less by 0.03 i.e., at 0.64. However, the precision value increased to 0.71. Finally, a model trained with PlantDoc cropped image dataset has resulted in accuracy, precision, and recall values of 0.63 and the f1-score of 0.62. AUC achieved was 0.92 and PRC area was 0.72. The evaluation results indicated that the CNN model trained against the original PlantDoc dataset was able to achieve better results as compared to the model trained with other variations of the PlantDoc dataset such as the PD balanced dataset and PD cropped dataset.

5.2.4 MobileNet V2 model with 50% parameters

In the attempt of limiting a model footprint by reducing the size of the CNN model, an experiment was conducted to train multiple models using the MobileNet V2 base model with a 50% reduced number of parameters. The model evaluation results obtained from the experiment are listed in Table 15.

Table 15 – MobileNet V1 (50%) Model Evaluation Results

Dataset	**Performance Evaluation Metrics**					
	Accuracy	**Precision**	**Recall**	**F1-Score**	**PRC**	**AUC**
PVD (Train & Test)	0.97	0.97	0.97	0.97	1.00	1.00
PVD (Train) + PD (Test)	0.34	0.33	0.30	0.32	0.36	0.70
PD (Train & Test)	**0.67**	**0.69**	**0.67**	**0.67**	**0.72**	**0.92**
PD-Bal (Train & Test)	0.58	0.64	0.58	0.58	0.63	0.88
PD-Cropped (Train & Test)	0.62	0.64	0.62	0.62	0.68	0.91

A model trained on the PVD dataset resulted in higher evaluation metrics values, reaching accuracy, precision, recall, and f1-score to 0.97 and the AUC & PRC area of almost 1.00. As expected, performance was degraded with the PlantDoc dataset for the model trained with the PlantVillage dataset where the accuracy achieved was just 0.34. When the model was fine-tuned with the PlantDoc dataset, it resulted in slightly better results as compared to the CNN model trained with 100% parameters. The CNN model was able to achieve accuracy, recall, an f1-score of 0.67, and precision of 0.69 with an AUC of 0.92. The CNN model trained with a balanced dataset resulted in lower values of accuracy (0.58), precision (0.64), recall (0.58), f1-score (0.58), and AUC of 0.88. The same results were observed with a model trained on PlantDoc cropped dataset where accuracy, recall, and f1-score were slightly reduced to 0.62 and precision value of 0.64.

Based on the analysis of results obtained with CNN models with 50% parameters, it was concluded that the results were at par with the CNN models trained using the MobileNet V2 default model (100% parameters). The performance was even slightly better against the PlantDoc original dataset.

5.2.5 MobileNet V2 (50% parameters) model with lower input image sizes

As a part of this research work, CNN models were trained with different input image sizes to understand the impact of change in input image size on model performance. As the previous experiments proved that the CNN model trained with MobileNet V2 with 50% parameters yields the best results, it was selected as a base model to experiment with the different image input sizes. The CNN models were trained by resizing the input images to 192 X 192 pixels height and width. Then images were again resized to 128 X 128 pixels height and width and models were trained on the different datasets. Table 16 shows the results of the CNN models evaluated using various classification metrics.

Table 16 – MobileNet V2 Model Evaluation Results with different input sizes

Input Size	Dataset	Performance Evaluation Metrics					
		Accuracy	Precision	Recall	F1-Score	PRC	AUC
(128, 128, 3)	PVD (Train & Test)	0.96	0.96	0.96	0.96	0.99	1.00
	PVD (Train) + PD (Test)	0.32	0.33	0.32	0.33	0.33	0.64
	PD (Train & Test)	0.61	0.65	0.61	0.62	0.67	0.90
	PD-Cropped (Train & Test)	0.65	0.65	0.65	0.65	0.68	0.91

(192, 192, 3)	PVD (Train & Test)	0.96	0.96	0.96	0.96	0.99	1.00
	PVD (Train) + PD (Test)	0.32	0.30	0.29	0.29	0.29	0.68
	PD (Train & Test)	**0.67**	**0.69**	**0.67**	**0.67**	**0.73**	**0.92**
	PD-Cropped (Train & Test)	0.64	0.66	0.64	0.64	0.71	0.92

The CNN model trained using a lower input image size of (128, 128, 3) showed slightly reduced performance against PVD and PD dataset variations as compared to the model trained with an input image size of (224, 224, 3). The classification metrics against the PVD dataset were able to achieve a 0.96 value for accuracy, precision, recall, and f1-score. The accuracy and recall with PD original dataset were also decreased to 0.61 and precision value to 0.65. As a next experiment, CNN models were trained using an input image size of (192, 192, 3). The model performance has shown a reduction in performance metrics for the PlantVillage dataset as compared to the higher input size. However, for PlantDoc original dataset, the model achieved the same performance as the model trained with 224 X 224-pixel size. Based on the performance evaluation of these models trained with different input sizes, it was decided to use an input image size of (224, 224, 3) for final model training.

5.2.6 Evaluation of EfficientNet B0 CNN model

EfficientNet (Tan and Le, 2019) models provide a combination of efficiency and accuracy by introducing various scales for model scaling. This research leveraged the smallest base model (B0) from the EfficientNet model family which is suitable for this research objective. Table 17 shows results obtained from various CNN models by using the EfficientNet B0 base model.

Table 17 – EfficientNet B0 Model Evaluation Results

Dataset	**Performance Evaluation Metrics**					
	Accuracy	**Precision**	**Recall**	**F1-Score**	**PRC**	**AUC**
PVD (Train & Test)	0.97	0.97	0.97	0.97	1.00	1.00
PVD (Train) + PD (Test)	0.27	0.29	0.22	0.25	0.26	0.67
PD (Train & Test)	**0.66**	**0.69**	**0.66**	**0.66**	**0.76**	**0.93**
PVD-Bal (Train & Test)	0.95	0.95	0.95	0.95	0.99	1.00
PVD-Bal (Train) + PD-Bal (Test)	0.26	0.31	0.21	0.25	0.25	0.68
PD-Bal (Train & Test)	0.59	0.63	0.59	0.59	0.69	0.91
PD-Cropped (Train & Test)	0.64	0.65	0.64	0.64	0.72	0.93

A fine-tuned model trained on PVD original dataset with EfficientNet B0 base model was able to achieve accuracy, precision, recall, and f1-score of 0.97, which is lower by 0.02 as compared to MobileNet family models trained for the same dataset. Similar results were monitored for PVD balanced dataset which was able to achieve a 0.95 score for all evaluation metrics which is less by 0.03 as compared to MobileNet models. The CNN model fine-tuned using PD original dataset was able to achieve accuracy, recall, and f1-score of 0.66 and precision score of 0.69. The CNN models trained with PD Balanced dataset were able to achieve lower performance (accuracy, recall, and f1-score of 0.59 and precision of 0.63) due to the reduced number of images in the dataset. The CNN model trained on the PD cropped image dataset produced slightly lower results with accuracy, recall, and f1-score of 0.64 and precision of 0.65 with an AUC score of 0.93. Overall, model performance analysis indicated slightly reduced performance of EfficientNet B0 models on variations of both the dataset as compared to MobileNet model family.

5.2.7 Evaluation of NasNetMobile CNN model

This research leveraged the NasNetMobile CNN base model and trained various models on different variations of PlantVillage and PlantDoc datasets. The evaluation results for those models are listed in Table 18.

Table 18 – NasNetMobile Model Evaluation Results

Dataset	Performance Evaluation Metrics					
	Accuracy	**Precision**	**Recall**	**F1-Score**	**PRC**	**AUC**
PVD (Train & Test)	0.95	0.96	0.95	0.95	0.99	1.00
PVD (Train) + PD (Test)	0.23	0.24	0.22	0.23	0.20	0.58
PD (Train & Test)	0.54	0.57	0.54	0.54	0.59	0.88
PVD-Bal (Train & Test)	0.91	0.93	0.91	0.91	0.97	0.99
PVD-Bal (Train) + PD-Bal (Test)	0.27	0.30	0.26	0.28	0.26	0.62
PD-Bal (Train & Test)	**0.66**	**0.69**	**0.66**	**0.66**	**0.59**	**0.87**
PD-Cropped (Train & Test)	0.55	0.57	0.55	0.55	0.64	0.90

Based on the analysis of results obtained from CNN models trained using NasNetMobile base model was slightly on the lower side as compared to CNN models trained with MobileNet (V1 & V2) and EfficientNet B0 models. A fine-tuned CNN model with the PlantVillage dataset was able to achieve 95% of accuracy, recall, and f1-score, and 96% of precision value. A model trained on the PlantDoc dataset resulted in lower accuracy, recall, and f1-score metrics of 0.54 and precision of 0.57. However, a model trained on a balanced PlantDoc dataset resulted in higher accuracy

(0.66), precision (0.69), recall (0.66), f1-score (0.66) as compared to other CNN models trained on this dataset. The CNN model trained using cropped images dataset showed almost similar results as PlantDoc original dataset.

5.2.8 Evaluation of ResNet50 V2 CNN model

To compare the performance of large deep CNN architecture on PlantVillage and PlantDoc datasets with smaller architectures suitable for mobile, CNN models were trained using the ResNet50 V2 base model. Table 19 shows the summary of evaluation metrics for various CNN models trained using the ResNet50 V2 base model.

Table 19 – ResNet50 V2 Model Evaluation Results

Dataset	Performance Evaluation Metrics					
	Accuracy	**Precision**	**Recall**	**F1-Score**	**PRC**	**AUC**
PVD (Train & Test)	0.99	0.99	0.99	0.99	1.0	1.00
PVD (Train) + PD (Test)	0.27	0.28	0.26	0.27	0.24	0.63
PD (Train & Test)	**0.67**	**0.69**	**0.67**	**0.66**	**0.70**	**0.90**
PVD-Bal (Train & Test)	0.98	0.98	0.98	0.98	1.00	1.00
PVD-Bal (Train) + PD-Bal (Test)	0.28	0.28	0.27	0.28	0.25	0.66
PD-Bal (Train & Test)	0.56	0.59	0.56	0.56	0.64	0.90
PD-Cropped (Train & Test)	0.65	0.67	0.65	0.65	0.73	0.92

CNN model fine-tuned with PVD original dataset resulted in high accuracy, precision, recall, and F1 score of 0.99 and AUC & PRC area almost reaching to value 1.00. When the same model was used to classify the tomato plant disease using the PlantDoc dataset, as expected results were poor with a lower value of accuracy (0.27), precision (0.28), recall (0.26), f1-score (0.27), PRC (0.24), and AUC (0.63). The same model was then retrained using the PlantDoc train dataset and accuracy was improved to 0.67 with an AUC score of 0.90. The precision (0.69), recall (0.67), and f1-score (0.66) metrics also showed significant improvements. The results obtained with the PlantDoc Balanced dataset were on the lower side as compared to PD original dataset with accuracy, recall, f1-score values of 0.56, and precision value of 0.59. CNN model trained with PlantDoc cropped dataset resulted in all classification metrics values smaller by 0.02 except AUC which was increased by 0.02.

Based on the analysis of evaluations performed and results obtained with the ResNet50 V2 model, it was evident that for both the PlantVillage and PlantDoc datasets, there were no significant model performance improvements by using a large deep CNN network instead of lighter CNN model architecture such as MobileNet, EfficientNet, NasNetMobile.

5.3 Final CNN model selection

During the research implementation, more than 50 CNN models were trained using different variations of the PlantVillage and PlantDoc dataset and with different hyperparameter combinations. This subsection provides the comparative analysis of the evaluation metrics carried out to select the final best model for deployment.

5.3.1 Comparison of CNN models performance

The results obtained from various CNN models were analyzed together by combining the performance metrics values from those models. To meet the objective of detecting tomato plant diseases using the mobile application in the plantation field, the analysis was focused on the model performance evaluation with the PlantDoc dataset, which contains real-world images as opposed to the PlantVillage dataset, which contains the images from controlled (lab) environment. The comparative analysis helped to select the best performing model as the "Final Model" to integrate with the Android Mobile App and deployed in the field to carry out inferences.

Table 20 – Performance evaluation metrics of CNN models

Models	Performance Evaluation Metrics					
	Accuracy	**Precision**	**Recall**	**F1-Score**	**PRC**	**AUC**
MobileNet V1 (100% parameters)	0.66	0.65	0.66	0.64	0.76	0.93
MobileNet V1 (50% parameters)	0.60	0.62	0.60	0.59	0.70	0.90
MobileNet V1 (Balanced Dataset)	0.63	0.65	0.63	0.62	0.69	0.89
MobileNet V1 (Cropped Dataset)	0.63	0.64	0.63	0.62	0.71	0.93
MobileNet V2 (100% parameters)	0.67	0.66	0.67	0.65	0.74	0.92
MobileNet V2 (50% parameters)	**0.67**	**0.69**	**0.67**	**0.67**	**0.72**	**0.92**
MobileNet V2 (Balanced Dataset)	0.64	0.71	0.64	0.64	0.66	0.90
MobileNet V2 (Cropped Dataset)	0.63	0.63	0.63	0.62	0.72	0.92
MobileNet V2 (128 px input size)	0.61	0.65	0.61	0.62	0.67	0.90
MobileNet V2 (192 px input size)	**0.67**	**0.69**	**0.67**	**0.67**	**0.73**	**0.92**
EfficientNet B0	0.66	0.69	0.66	0.66	0.76	0.93

EfficientNet B0 (Balanced Dataset)	0.59	0.63	0.59	0.59	0.69	0.91
EfficientNet B0 (Cropped Dataset)	0.64	0.65	0.64	0.64	0.72	0.93
NasNetMobile	0.54	0.57	0.54	0.54	0.59	0.88
NasNetMobile (Balanced Dataset)	0.66	0.69	0.66	0.66	0.59	0.87
NasNetMobile (Cropped Dataset)	0.55	0.57	0.55	0.55	0.64	0.90
ResNet50 V2	0.67	0.69	0.67	0.66	0.70	0.90
ResNet50 V2 (Balanced Dataset)	0.56	0.59	0.56	0.56	0.64	0.90
ResNet50 V2 (Cropped Dataset)	0.65	0.67	0.65	0.65	0.73	0.92

The performance evaluation metrics in Table 20 indicate that CNN models leveraging MobileNet family and EfficientNet B0 base models resulted in better performance as compared to NasNetMobile on PlantDoc dataset variations. There were no significant improvements with a cropped or balanced version of PlantDoc datasets indicated that CNN models were able to learn similar feature mapping from the original PlantDoc dataset. The CNN model trained using (192, 192, 3) input image size was able to achieve similar results as the model trained using (224, 224, 3) input size. However, model performance was degraded when input size was further reduced to (128, 128, 3). The models trained using reduced width multiplier factor (50% model parameters) for the MobileNet family indicated similar results as a model trained with 100% parameters. As seen in Table 20, CNN models trained with MobileNet V2 using 224 X 224 pixels height and width and 192 X 192 pixels input sizes performed best on PlantDoc original dataset. The performance metrics were almost similar with a minor difference of 0.01 for the PRC area. Hence, the CNN model trained with MobileNet V2 network architecture having image input size of (224, 224, 3) was considered as the final model based on performance evaluation metrics analysis.

5.3.2 Comparison of CNN model size

After completing the analysis of evaluation metrics to compare model performance, the next step was to compare the model sizes. As the objective of this research is to integrate the final model with the Android mobile app, the model size was one of the important considerations. A smaller model size helps to optimize the mobile resource consumption such as memory usage. It is also easier to update the Android app with a newly trained model to improve the model efficacy in the future with new data points when the model size is smaller. During the research implementation phase, various CNN models were saved using Keras SavedModel format, which is the

recommended approach for saving the model architecture, weights, and all model configurations. Table 21 shows the details of various saved model files with their size details.

Table 21 – Model File Size Details

Models	File Name	Size (in MB)
MobileNet V1 (100% parameters)	PD-MobileNet-FT.h5	17
MobileNet V1 (50% parameters)	**PD-MobileNet-FT-Alpha-0.5.h5**	**8**
MobileNet V1 (Balanced Dataset)	PD-MobileNet-FT-Balanced.h5	17
MobileNet V1 (Cropped Dataset)	PD-CD-MobileNet-FT	17
MobileNet V2 (100% parameters)	PD-MobileNetV2-FT.h5	15
MobileNet V2 (50% parameters)	**PD-MobileNetV2-FT-Alpha-0.5.h5**	**9**
MobileNet V2 (Balanced Dataset)	PD-MobileNetV2-FT-Balanced.h5	15
MobileNet V2 (Cropped Dataset)	PD-CD-MobileNetV2-FT	15
MobileNet V2 (128 px input size)	PD-MobileNetV2-FT-Alpha-0.5-128px.h5	9
MobileNet V2 (192 px input size)	PD-MobileNetV2-FT-Alpha-0.5-192px.h5	9
EfficientNet B0	PD-EfficientNet-B0-FT.h5	22
EfficientNet B0 (Balanced Dataset)	PD-EfficientNet-B0-FT-Balanced.h5	22
EfficientNet B0 (Cropped Dataset)	PD-CD-EfficientNet-B0-FT.h5	22
NasNetMobile	PD-NasNetMobile-FT.h5	23
NasNetMobile (Balanced Dataset)	PD-NasNetMobile-FT-Balanced.h5	23
NasNetMobile (Cropped Dataset)	PD-CD-NasNetMobile-FT.h5	23
ResNet50 V2	PD-ResNet50V2-FT.h5	106
ResNet50 V2 (Balanced Dataset)	PD-ResNet50V2-FT-Balanced.h5	106
ResNet50 V2 (Cropped Dataset)	PD-CD-ResNet50V2-FT.h5	106

As observed from model size details, the CNN models trained using MobileNet family architecture resulted in smaller model sizes ranging between 15 to 17 MB of size on disk. The model size was further optimized with the models trained using 50% of parameters resulting in the MobileNet V1 model of 8 MB and Mobile V2 model size of 9 MB. After comparing the model evaluation results based on the various classification metrics and model size comparison, it was decided to select a CNN model trained with 50% parameters using MobileNet V2 architecture as the “Final Model”.

5.4 Evaluation on validation datasets using Final Model

The final model was evaluated against the PlantDoc test dataset and against the PlantVillage test dataset to check if a single model can be leveraged for tomato plant disease detection using real-world and controlled (lab) environment images. The evaluate() method called for the final model resulted in Figure 26.

```
loss :  1.047428011894226
tp :  68.0
fp :  23.0
tn :  649.0
fn :  44.0
accuracy :  0.6785714030265808
precision :  0.7472527623176575
recall :  0.6071428656578064
auc :  0.9119832515716553
prc :  0.7285691499710083
```

Figure 26 – Final CNN model evaluation results (PlantDoc Dataset)

The model resulted in an accuracy of 0.68 against the PlantDoc test dataset with an AUC area of 0.91 and a PRC area of 0.73. To analyze the performance for each disease class type, a classification report was generated using the Scikit-learn metrics package. The classification results are shown in Figure 27.

Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.43	0.46	0.44	13
Tomato_Early_Blight	0.69	0.65	0.67	17
Tomato_Healthy	0.65	0.65	0.65	17
Tomato_Late_Blight	0.67	0.50	0.57	16
Tomato_Leaf_Mold	0.69	0.79	0.73	14
Tomato_Septoria_Leaf_Spot	0.76	0.80	0.78	20
Tomato_Yellow_Leaf_Curl_Virus	0.81	0.87	0.84	15
accuracy			0.68	112
macro avg	0.67	0.67	0.67	112
weighted avg	0.68	0.68	0.68	112

Figure 27 – Final CNN mode classification report (PlantDoc Dataset)

The precision, recall, and f1-scores comparison for different classes show that there were some misclassifications between Tomato_Early_Blight and Tomato_Late_Blight classes. It is due to the similarity of the symptoms for these two diseases. Tomato_Yellow_Leaf_Curl_Virus and Tomato_Leaf_Mold showed overall good performance with higher precision, recall, and f1-score values. To analyze the misclassification rate further, a confusion matrix was generated and plotted using a heatmap which is shown in Figure 28.

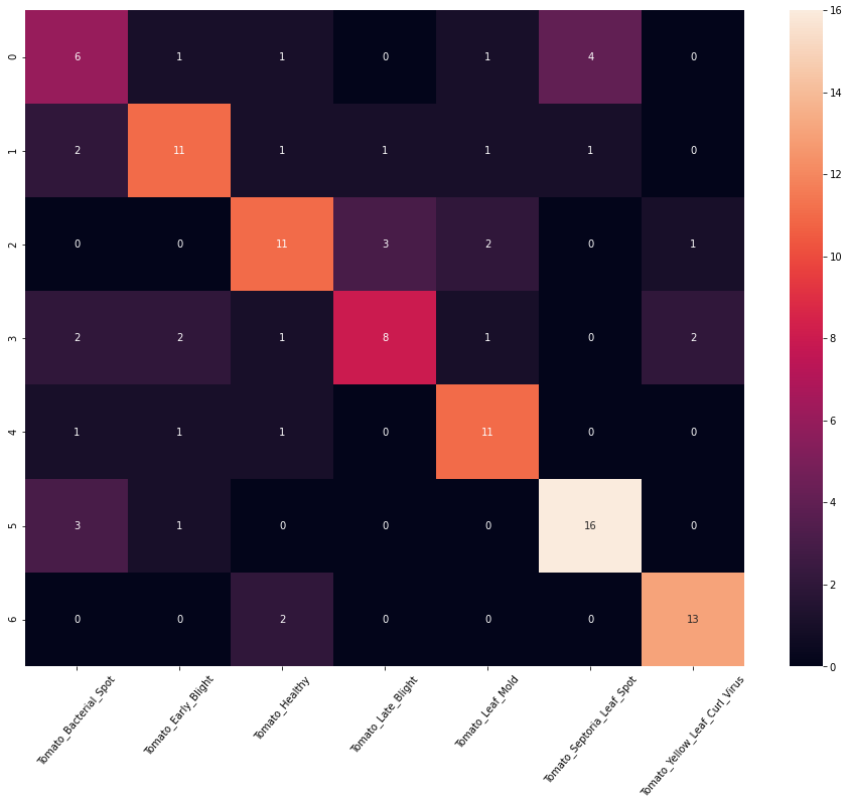


Figure 28 – Final CNN model Confusion Matrix (PlantDoc Dataset)

As seen from the Confusion Matrix heatmap, for Tomato_Early_Blight the FP value is 6 and FN value is 5 while TP is 11 and TN is 90. For Tomato_Late_Blight TP value is 8 and the TN value is 92, whereas FP value is 8 and the FN value is 4. That indicates for Tomato_Early_Blight, the final CNN model misclassified 11 instances and for Tomato_Late_Blight total misclassification was 12. The total misclassification instances for Tomato_Yellow_Leaf_Curl_Virus were 5 where the FP value was 2 and the FN value was only 3. Based on the analysis of the confusion matrix, it is evident that the model misclassified very few instances of Tomato_Yellow_Leaf_Curl_Virus disease class. There were more misclassification instances observed for Tomato_Early_Blight and Tomato_Late_Blight due to similarity in disease symptoms for these two classes. The behaviour was observed for Tomato_Bacterial_Spot and Tomato_Septoria_Leaf_Spot due to the similarity of disease symptoms.

The final CNN model performance was also verified against the PlantVillage dataset. Figure 29 shows the classification report generated which contains various classification metrics for this evaluation.

```
Classification Report
                              precision    recall  f1-score   support

        Tomato_Bacterial_Spot       0.35      0.26      0.30       425
          Tomato_Early_Blight       0.15      0.20      0.17       200
               Tomato_Healthy       0.77      0.96      0.86       318
           Tomato_Late_Blight       0.44      0.90      0.60       382
             Tomato_Leaf_Mold       0.16      0.56      0.25       190
    Tomato_Septoria_Leaf_Spot       0.41      0.27      0.32       354
Tomato_Yellow_Leaf_Curl_Virus       1.00      0.25      0.40      1071

                     accuracy                           0.43      2940
                    macro avg       0.47      0.49      0.41      2940
                 weighted avg       0.62      0.43      0.42      2940
```

Figure 29 – Final CNN mode classification report (PlantVillage Dataset)

The final CNN model was able to achieve the overall 43% accuracy rate with a precision value of 0.62 and a recall value of 0.43. The f1-score resulted in 0.42. This evaluation metrics indicates that the performance of a fine-tuned model with PlantDoc real-world images is degraded for the PlantVillage dataset. This is due to the difference in images in both datasets. Based on this analysis, it can be concluded that a single CNN model cannot be leveraged for two different types of images i.e., images from a controlled environment and images from a real-world plantation field.

5.5 Evaluation of Plant Disease Severity Model

One of the important objectives of the research is to detect the severity of plant disease. To meet this objective, the research adopted two different approaches. In the first approach, features were extracted manually, and machine learning models were trained. In the second approach, research leverage the CNN model to detect plant disease severity. The next subsections will explain the results obtained for both approaches and compare those against each other.

5.5.1 Machine Learning Models

In this approach, multiple models were trained using the feature vectors created by combining Global (color, shape, and texture) features and local (ORB key points and feature descriptors). In this research, two types of Machine Learning classification models were trained using Random Forest and SVM classifier. Table 22 shows the evaluation results for ML models achieved during those experiments.

Table 22 – Plant Disease Severity ML Models Evaluation Results

Features	ML Algorithm	Performance Evaluation Metrics			
		Accuracy	Precision	Recall	F1-Score
Raw Image features	Random Forest	0.54	0.54	0.54	0.53
	SVM classifier	0.65	0.68	0.65	0.65
Global Features (color, shape, and texture)	Random Forest	0.80	0.82	0.80	0.80
	SVM classifier	0.37	0.30	0.37	0.27
ORB features + Bag of Visual Words (BOVW)	Random Forest	0.33	0.35	0.33	0.33
	SVM classifier	0.34	0.22	0.34	0.24

The model trained using Random Forest classifier on raw image features resulted in a low accuracy rate of 0.54, precision and recall value of 0.54, and f1-score of 0.53. However, the SVM classifier trained on the raw image features improved the accuracy, recall, and f1-score values to 0.65 and precision value to 0.68. The model trained on combined global feature vectors using Random Forest classifier resulted in a good performance with a high accuracy rate, recall and f1-score of 0.80, and precision value of 0.82. On the contrary, the SVM model trained on the same combined global feature vectors shown significantly lower performance with decreased accuracy (0.37), precision (0.30), recall (0.37), and f1-score (0.27) values. The model trained using a combination of ORB features extracted and visual vocabulary created using Bag of Visual Words (BOVW) technique resulted in degraded model performance with both Random Forest and SVM classifiers. The Random Forest model has achieved accuracy, recall, and f1-score of 0.33 and precision value of 0.35, whereas SVM classifier resulted in accuracy and recall of 0.34, the precision value of 0.22, and the f1-score of 0.24.

Due to schedule constraints, this research could not try an approach to combine global features with the local feature, which expected to be achieved even better results.

5.5.2 CNN Model with Transfer Learning (MobileNet V2)

After training machine learning models (Random Forest and SVM) on hand-engineered features, research focused on training CNN classifier models leveraging transfer learning techniques to detect disease severity. The research leveraged the MobileNet V2 base model and trained the classifier using the default version of MobileNet V2 with 100% parameters as well as a 50% reduced width model. The results from both experiments are listed in Table 23.

Table 23 – Plant Disease Severity CNN Models Evaluation Results

Model	Performance Evaluation Metrics				
	Accuracy	Precision	Recall	F1-Score	AUC
MobileNet V2 (100% parameters)	0.81	0.82	0.81	0.81	0.98
MobileNet V2 (50% parameters)	0.78	0.80	0.78	0.78	0.98

A CNN model with 100% parameters was able to achieve an accuracy of 0.81, a precision of 0.82, a recall rate of 0.81 resulting in the f1-score of 0.81. AUC obtained was 0.98. However, a trim-down version of the CNN model with 50% parameters shown slightly lower values of accuracy (0.78), precision (0.80), recall (0.78), and f1-score (0.78) with the same AUC score (0.98). This indicated that CNN models performed better as compared to traditional ML classification models with a manual feature engineering approach.

5.6 Inferences using Android Mobile App

As the end state of the study is to publish the Android Application in Android Play Store Marketplace, the research conducted model performance versus inference speed trade-off analysis. Figure 30 shows the navigation flow to carry out inferences using the SmartCropCNNLite mobile application.

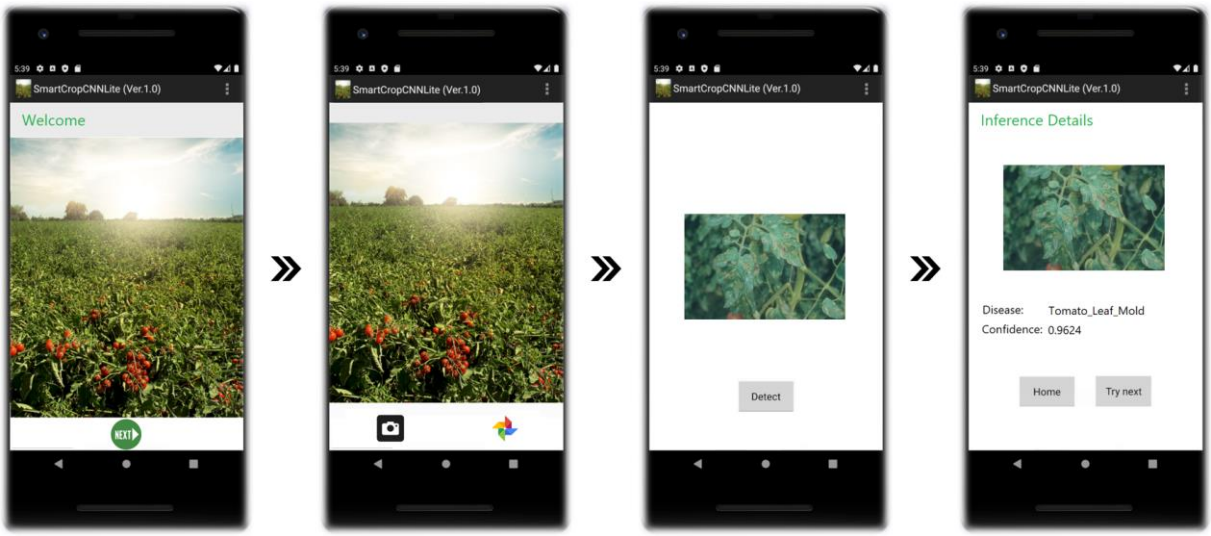


Figure 30 – Inferencing flow for Android Mobile App

The model size of deployed TFLite model was 9 MB, which ensured optimal resource usage for mobile devices. The mobile application is designed to capture a single image using the default

camera app or by selecting it from Photo Gallery. Hence, it was decided not to analyze FPS (Frames per seconds) rate during the evaluation process (which can be considered later when the app can be enhanced to work with continuous video stream). The inference time was measured from the point of triggering the prediction using the CNN model by submitting the image. During the multiple evaluation iteration with a varied range of images, the results returned were within 30 to 45 seconds. This confirmed that the SmartCropCNNLite mobile application can be leveraged by farmers to perform disease identification in real-time in actual plantation fields.

5.7 Summary

To summarize, this chapter discussed the results obtained using various evaluation metrics for multiple CNN models trained based on different CNN model architectures suitable for the mobile platform. Initially, results obtained using feature extractor models against the PlantVillage dataset and fine-tuned models against variations of the PlantDoc dataset were analyzed. The results showed MobileNet family of models and EfficientNet B0 models performed well against both datasets. After individual model evaluation, a comparative analysis was performed using evaluation metrics details obtained from various base CNN models. As the next step, the model file sizes were also compared. Based on these analysis details, the MobileNet V2 model with 50% parameters was selected as the final model and it was used to evaluate against the PlantDoc Test dataset. To achieve the objective of detecting plant disease severity, models trained using trading ML classification algorithms such as Random Forest and SVM were evaluated. The model trained using consolidated global features (color, shape, and texture) and a Random Forest classifier were able to achieve the best results. Further, CNN models were trained using MobileNet V2 default base model as well with 50% parameters. The CNN model trained with 100% parameters obtained comparatively good results with a high accuracy rate of 81% and AUC of 98%. Finally, the best CNN model integrated with the Android mobile application was used to carry out inferences with a new set of tomato plant images and results were analyzed.

CHAPTER VI:

CONCLUSIONS AND RECOMMENDATIONS

6.1 Introduction

In this chapter, the conclusions for this research study will be drawn based on the data analysis findings, model training process, and model evaluation conducted. The section about discussion and conclusion will provide a summary of the overall process adopted in this study including data preparation, data analysis, data processing, model training, and model validations. The conclusions drawn from these stages will justify the objectives of this study and will indicate if the overall aim of this study is achieved or not. The next section will discuss the contributions made by this study to an existing body of knowledge. Finally, a section on future recommendations will show the path on how this study can be enhanced in the feature to ensure better model performance and implement the end-to-end solution for plant disease type identification, severity detection, and to recommend treatment for tomato plant diseases.

6.2 Discussion and Conclusion

This research work began with the identification of suitable datasets which can be leveraged to detect six common types of tomato plant diseases which have the most adverse impact on the tomato crop yield. The PlantVillage dataset containing images from a controlled (lab) environment was selected due to enough images for training CNN models. As one of the important objectives of this research is to design and implement a real-world solution which can be used by farmers in actual farm plantation field, it was important to train and validate the proposed models on the real-world images. This leads to the selection of the PlantDoc dataset which contains a smaller number of images as compared to the PlantVillage dataset, however, those images are taken from actual farm fields. The images part of this dataset is of varied sizes which were taken from different angles and captured in varied lighting conditions. The images also contain background noise such as other plants, fruits, weeds, person hands, and other objects. It also contains images with a combination of diseased and healthy plant foliage, which is not the case for the PlantVillage dataset. After downloading the datasets, a quick validation was performed to ensure the correctness of image annotations and to find any issues with image labeling and address those. The image data analysis was conducted, and important decisions were made about input sizes of images and batch size to be considered for model training. As a part of EDA, the class imbalance ratio was analyzed, and it

was found that each dataset had a considerable degree of imbalance. It was decided to apply both oversampling and undersampling techniques and verify model performance against those derived datasets. As the PlantVillage dataset already contains augmented images, it was decided not to apply oversampling techniques which could potentially result in duplicate image leading model to overfit. For the PlantDoc dataset, images were augmented using the OpenCV image library to balance the dataset using oversampling. During this research, cropped image dataset was curated using the PlantDoc object detection dataset and PascalVOC XML notations. A separate dataset was derived for the detection of plant disease severity using the PlantVillage dataset due to the availability of required sample images with various degrees of disease spread. In the data pre-processing stage, images were resized and rescaled using Keras APIs rather than custom python scripts. The images were also normalized to ensure optimal training performance for CNN models. To avoid overfitting and ensure trained CNN models generalized well with unseen image data, various image augmentation techniques were applied such as random zoom, rotations, sheer, and horizontal flip. Finally, the processed dataset was split into training (80%) and validation (20%) dataset using a hold-out methodology.

The research evaluated two approaches of features engineering – (1) automatic feature extraction using CNN models and (2) manual feature extraction and selection process for plant severity detection. During the CNN model training with transfer learning, first SOA modern CNN architectures were shortlisted which are suitable for building lighter CNN models which can be deployed onto the Android mobile application. After the selection of base CNN models, various feature extraction models were trained by replacing the classification layer of the CNN network to fit the requirements of this research. The feature extraction models were trained against the PlantVillage dataset first. As the next step, the model was fine-tuned by gradually unlocking the last layers and making those trainable. The model performance was found to be optimal when almost half of the last layers of the CNN network are retrained. Based on this observation, it can be concluded that for the dataset with plant images, the middle layers of the CNN model are learning some important complex features which differ between the ImageNet dataset and both the datasets used for this research. At every iteration, a model was evaluated by comparing training accuracy plots with validation accuracy and training loss with validation loss. It was decided to rely on performance evaluation metrics which worked best with imbalanced datasets such as confusion matrix, precision, recall, f1-score, AUC, and PRC areas rather than only validating the

model accuracy. The fine-tune model performance was then evaluated using real-world images from the PlantDoc dataset which resulted in degraded performance almost by 60% to 70%. As next experimentation, models trained using ImageNet and PlantVillage dataset training weights were fine-tuned using the PlantDoc training dataset. This resulted in significant improvements in overall model performance. The model validation accuracy was increased almost double from approximately 0.30 to 0.65 for various models and all the other classification evaluation metrics showed the same improvements. During this research, multiple experiments were carried out and more than 50 CNN models were trained and evaluated. The different variations of both datasets which include over-sampled / under-sampled balanced datasets and cropped image datasets were used to train these various CNN models. In addition, MobileNet family models (V1 & V2) were trained using different parameter sizes to evaluate the performance of smaller size models with the various datasets. It was concluded that the model trained with 50% parameters for the MobileNet family of models has similar or for some models even better performance against the dataset variations used in this research. sss

During the hyperparameter tunning, different input image sizes were considered to train and evaluate the models. The research performed extensive hyperparameters tunning by trying multiple combinations of parameter settings. The important model parameters considered were multiple optimizer options (SGD, RMSprop, and Adam) with different learning rates, learning rate decay and schedules, number of fine-tune layers, number of epochs, and early stopping techniques while model training to control overfitting of the models. Although the RMSprop and Adam optimizers improved the training duration, the models trained using SGD optimizer with momentum were generalized best. The exponential decay schedule along with the early stopping method proved to be the best approach to train the CNN models for exact required iterations and helped to control the overfitting of the models. After finalizing the optimal hyperparameter values, CNN models were trained using those settings and evaluated. During the evaluation phase, models' performance was validated individually and then all the performance evaluation metrics were compared together for all CNN models. Further, trained CNN model sizes were compared. Based on this comparative analysis of various classification performance metrics and model sizes, it was finalized to select a CNN model trained using MobileNet V2 50% parameters as the "Best Model". The final CNN model had achieved accuracy, recall, and f1-score of 0.67. It resulted in a precision value of 0.69 with a higher AUC area of 0.92 against the PlantDoc test dataset. After achieving satisfactory

results against PlantDoc original dataset, model performance was tested against the PlantVillage dataset. It was noticed that the performance of the final model was decreased on the PlantVillage dataset due to the differences in images from controlled (lab) setting and real-world plantation field. From this experiment, it was concluded that a single fine-tuned model cannot be leveraged for both PlantVillage and PlantDoc datasets. As the objective of this research is to focus on real-world plant disease detection, this model was considered for deployment. The final model was then converted into a TFLite model using Keras Python API and imported into Android studio. The final model was integrated with SmartCropCNNLite Android Application and inferences were carried out using the mobile device. Based on the inferences carried out using the mobile phone and results obtained, it can be concluded that the lighter CNN model trained using MobileNet V2 architecture and transfer learning has a lot of potentials to perform tomato disease identification in the plantation field.

This research also focused on plant severity detection which is the next important consideration after the identification of tomato disease. To limit the scope of experimentation due to schedule constraints for this research work, only two tomato disease classes were considered. Accurate identification of the degree of disease spread once the specific tomato plant disease type is detected can help to build the "Tomato Plant Disease Treatment Recommendation System." The first step to detect plant disease severity was to select which features to be analyzed. This research decided to experiment with both global image features such as Color, Shape, and Texture and local image features which include key points and feature descriptors extracted using the ORB algorithm. The choice of ORB algorithm made instead of SIFT proprietary methods due to its better performance and alignment with the OSS distribution model, which made it a preferred option to integrate with the Android mobile app which can be freely distributed from the Android PlayStore marketplace. Initially, classical ML classification algorithms such as Random Forest and SVM were used to classify the plant severity levels. The evaluation of those models indicated that the Random Forest model trained using feature vectors combining global features resulted in better model performance (accuracy rate, recall, and f1-score of 0.80) as compared to models trained using ORB features. Further, CNN models were trained using the MobileNet V2 SOA base model to classify the severity levels of two pre-selected Tomato plant diseases. The CNN model trained using MobileNet V2 default (100% parameters) resulted in the best performing model with an accuracy rate of 0.81, a precision of 0.82, a recall rate of 0.81, f1-score of 0.81, and a high AUC area of 0.98 for plant

severity classification problems. This indicates that the trained CNN model can be leveraged for plant disease severity classification.

To conclude, the SmartCropCNNLite Android App leveraging the CNN model can be used by farmers as a reliable solution to identify the tomato plant diseases in an actual farm field. In addition to the identification of specific tomato plant diseases, this research has shown significant potential to detect the plant disease severity levels which can be a foundation layer to build the "Tomato Plant Disease Treatment Recommendation System."

6.3 Contributions to Knowledge

This research project on tomato plant disease identification and severity detection leveraging public datasets with different characteristics made three important contributions to the existing body of knowledge. The previous research work for tomato plant disease detection was focused primarily on improving deep learning model accuracy using the public image dataset from a controlled (lab) environment. The focus of this research, however, was on implementing a real-world solution that can be leveraged by farmers in the tomato plantation field for the early identification of common tomato plant diseases. The CNN models were trained using real-world images with varied sizes, different angles, different lighting conditions, without removing any background noise from the images intentionally to resemble the real-world scenario. The focus of this research was not only improving the model accuracy but also enhancing the overall model efficacy by reducing the model size and optimizing inference time. This resulted in a lighter CNN model which is suitable to deploy on mobile devices. Another important contribution to knowledge based on this research study includes combining the tomato plant disease identification and severity detection in a single solution that can be deployed on mobile devices (which was not covered in the previous research work based on a literature study). Furthermore, this research work also compared the state-of-the-art modern CNN architecture suitable for mobile platforms against the real-world tomato plant images dataset and recommended the best model architecture based on multiple model performance evaluations.

6.4 Future Recommendations

This research work can be enhanced by improving the CNN model performance against the real-world image dataset which contains a greater number of images as compared to the PlantDoc

dataset which has a very limited number of images. The research can be further extended to include the other types of tomato plant diseases in addition to six common diseases considered in this study. The study considered only two types of diseases to detect the severity of the disease, which can be extended to include additional plant disease types. This requires plant pathology experts to annotate the images based on disease type and severity. The new dataset with real-world images to identify plant disease and further classify severity level can help to improve model performance, making it a more robust solution for farmers to use in the tomato plantation field. The CNN model trained to detect tomato plant disease type and severity can be used as a foundation to build a complete solution to recommend treatment for tomato plant diseases which can help to reduce huge crop yield losses. This can improve the lifestyle of small-sector farmers, which will directly contribute to the small and medium-scale agriculture sector and finally the overall economy of developing countries in the world.

REFERENCES

Adrian Rosebrock, 2019. Keras ImageDataGenerator and Data Augmentation - PyImageSearch [WWW Document]. URL https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/ (accessed 3.8.21).

Agarwal, M., 2019. A Convolution Neural Network based approach to detect the disease in Corn Crop 176–181.

Al Bashish, D., Braik, M., Bani-Ahmad, S., 2011. Detection and classification of leaf diseases using K-means-based segmentation and neural-networks-based classification. Information Technology Journal 10, 267–275. https://doi.org/10.3923/itj.2011.267.275

Arya, M.S., Anjali, K., Unni, D., 2018. Detection of unhealthy plant leaves using image processing and genetic algorithm with Arduino. EPSCICON 2018 - 4th International Conference on Power, Signals, Control and Computation 1–5. https://doi.org/10.1109/EPSCICON.2018.8379584

Ashqar, B.A.M., Abu-Naser, S.S., 2018. Image-Based Tomato Leaves Diseases Detection Using Deep Learning. International Journal of Academic Engineering Research 2, 10–16.

Bay, H., Ess, A., Tuytelaars, T., Van Gool, L., 2008. Speeded-Up Robust Features (SURF). Computer Vision and Image Understanding 110, 346–359. https://doi.org/10.1016/j.cviu.2007.09.014

Chapaneri, R., Desai, M., Goyal, A., Ghose, S., Das, S., 2020. Plant Disease Detection: A Comprehensive Survey. 2020 3rd International Conference on Communication Systems, Computing and IT Applications, CSCITA 2020 - Proceedings 220–225. https://doi.org/10.1109/CSCITA47329.2020.9137779

Dalal, N., Triggs, B., 2005. Histograms of oriented gradients for human detection. Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005 I, 886–893. https://doi.org/10.1109/CVPR.2005.177

Datacourses.com, 2020. Classification Model Evaluation Metrics in Scikit-Learn - Data Courses [WWW Document]. URL https://www.datacourses.com/classification-model-evaluation-metrics-in-scikit-learn-924/ (accessed 3.13.21).

Dhakate, M., B, I.A., 2015. A Review on Pomegranate Disease Classification Using Machine Learning and Image Segmentation Techniques. Proceedings of the International Conference on Intelligent Computing and Control Systems, ICICCS 2015 455–460. https://doi.org/10.1109/ICICCS48265.2020.9121161

Durmus, H., Gunes, E.O., Kirci, M., 2017. Disease detection on the leaves of the tomato plants by using deep learning. 2017 6th International Conference on Agro-Geoinformatics, Agro-Geoinformatics 2017. https://doi.org/10.1109/Agro-Geoinformatics.2017.8047016

Elhassouny, A., Smarandache, F., 2019. Smart mobile application to recognize tomato leaf diseases using Convolutional Neural Networks 1–4. https://doi.org/10.1109/ICCSRE.2019.8807737

FAO | IFAD | UNICEF | WFP and WHO, 2021. The State of Food Security and Nutrition in the World 2021. The State of Food Security and Nutrition in the World 2021. https://doi.org/10.4060/CB4474EN

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial networks. Advances in Neural Information Processing Systems 27 (NIPS 2014) 63, 139–144. https://doi.org/10.1145/3422622

Gould, S., Fulton, R., Koller, D., 2009. Decomposing a scene into geometric and semantically consistent regions. Proceedings of the IEEE International Conference on Computer Vision 1–8. https://doi.org/10.1109/ICCV.2009.5459211

Hanks, T.W., Stewart, E.L., Kaczmar, N., Dechant, C., Wu, H., Nelson, R.J., Lipson, H., Gore, M.A., 2018. Image set for deep learning : field images of maize annotated with disease symptoms. BMC Res Notes 10–12. https://doi.org/10.1186/s13104-018-3548-6

He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE International Conference on Computer Vision 2015 Inter, 1026–1034. https://doi.org/10.1109/ICCV.2015.123

Hossain, E., Hossain, M.F., Rahaman, M.A., 2019. A Color and Texture Based Approach for the Detection and Classification of Plant Leaf Disease Using KNN Classifier. 2nd International Conference on Electrical, Computer and Communication Engineering, ECCE 2019 7–9. https://doi.org/10.1109/ECACE.2019.8679247

Howard, A.G., Wang, W., 2012. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K., 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size 1–13.

Indian Council of Agricultural Research, 2017. Tomato cultivation.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 32nd International Conference on Machine Learning, ICML 2015 1, 448–456.

Islam, M., Dinh, A., Wahid, K., 2017. Detection of Potato Diseases Using Image Segmentation and Multiclass Support Vector Machine 8–11.

Janakiraman, A., 2019. Advances in plant pathology: Impact on tomato diseases [WWW Document]. URL https://www.openaccessgovernment.org/advances-in-plant-pathology/74198/ (accessed 1.11.21).

Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li, Li Fei-Fei, 2009. ImageNet: A large-scale hierarchical image database 248–255. https://doi.org/10.1109/cvprw.2009.5206848

Keras ImageDataGenerator and Data Augmentation - PyImageSearch [WWW Document], n.d. URL https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/ (accessed 3.8.21).

Khan, S., Narvekar, M., 2019a. Survey of Techniques for Disorder Detection in Tomato(Solanum Lycopersicum). Proceedings of the 2nd International Conference on Intelligent Computing and Control Systems, ICICCS 2018 1853–1858. https://doi.org/10.1109/ICCONS.2018.8662933

Khan, S., Narvekar, M., 2019b. Survey of Techniques for Disorder Detection in Tomato(Solanum Lycopersicum). Proceedings of the 2nd International Conference on Intelligent Computing and Control Systems, ICICCS 2018 1853–1858. https://doi.org/10.1109/ICCONS.2018.8662933

Kluepfel, M., Blake, J., Keinath, A., 2020. Tomato Diseases & Disorders. Factsheet | HGIC 2217 1036, 1–18.

Krizhevsky, B.A., Sutskever, I., Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks.

Kumari, C.U., Prasad, S.J., Mounika, G., 2019. Leaf Disease Detection : Feature Extraction with K-means clustering and Classification with ANN 1095–1098.

Laricchia, F., 2024. Statista, Smartphone penetration worldwide [WWW Document]. URL https://www.statista.com/statistics/203734/global-smartphone-penetration-per-capita-since-2005/ (accessed 1.18.21).

Lowe, D.G., 1999. Object recognition from local scale-invariant features. Proceedings of the IEEE International Conference on Computer Vision 2, 1150–1157. https://doi.org/10.1109/iccv.1999.790410

Mahlein, A.-K., 2016. Present and Future Trends in Plant Disease Detection. Plant Dis 100, 1–11. https://doi.org/10.1007/s13398-014-0173-7.2

Majumder, S., Shirvaikar, M., 2020. Real-time detection of maize crop disease via a deep learning-based smartphone app. https://doi.org/10.1117/12.2557317

Meena Prakash, R., Saraswathy, G.P., Ramalakshmi, G., Mangaleswari, K.H., Kaviya, T., 2018. Detection of leaf diseases and classification using digital image processing. Proceedings of 2017 International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2017 2018-Janua, 1–4. https://doi.org/10.1109/ICIIECS.2017.8275915

Mohanty, S.P., Hughes, D., Salathé, M., 2016. Using Deep Learning for Image-Based Plant Disease Detection.

Mokbel, B., Paassen, B., Schleif, F.M., Hammer, B., 2015. Metric learning for sequences in relational LVQ. Neurocomputing 169, 306–322. https://doi.org/10.1016/j.neucom.2014.11.082

Mutka, A.M., Bart, R.S., 2015. Image-based phenotyping of plant disease symptoms. Front Plant Sci 5, 1–8. https://doi.org/10.3389/fpls.2014.00734

O'Dea, S., 2022. Statista, Mobile OS market share worldwide, Statistics and Market Data on Telecommunications.

Owomugisha, G., Mwebaze, E., 2017. Machine learning for plant disease incidence and severity measurements from leaf images. Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016 158–163. https://doi.org/10.1109/ICMLA.2016.126

Padol, P.B., Yadav, A.A., 2016. SVM classifier based grape leaf disease detection. Conference on Advances in Signal Processing, CASP 2016 175–179. https://doi.org/10.1109/CASP.2016.7746160

pascalvoc-to-image · PyPI [WWW Document], 2022. URL https://pypi.org/project/pascalvoc-to-image/ (accessed 5.11.21).

Plant Disease: Pathogens and Cycles | CropWatch [WWW Document], 2021. URL https://cropwatch.unl.edu/soybean-management/plant-disease (accessed 3.4.21).

Ramakrishnan, M., Sahaya, A.N.A., 2015. Groundnut leaf disease detection and classification by using back probagation algorithm. 2015 International Conference on Communication and Signal Processing, ICCSP 2015 964–968. https://doi.org/10.1109/ICCSP.2015.7322641

Ramcharan, A., Baranowski, K., Mccloskey, P., Ahmed, B., Legg, J., Hughes, D.P., 2017. Deep Learning for Image-Based Cassava Disease Detection 8, 1–7. https://doi.org/10.3389/fpls.2017.01852

Richard Szeliski, 2011. Computer vision algorithms and applications. Springer pp. 10–17, 43-1-43–23.

Rosebrock, A., 2017. Deep Learning for Computer Vision with Python: Practitioner Bundle, Deep learning for computer vision with Python. PyImageSearch.

Rqjodl, L., Glvhdvh, S., Khos, Z., Uhgxfh, W.R., Orvvhv, L., 2018. Unsupervised Representation Learning of Image-Based Plant Disease with Deep Convolutional Generative Adversarial Networks 9159–9163. https://doi.org/10.23919/ChiCC.2018.8482813

Rublee, E., Rabaud, V., Konolige, K., Bradski, G., 2011. ORB: An efficient alternative to SIFT or SURF. Proceedings of the IEEE International Conference on Computer Vision 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544

Sandler, M., Zhu, M., Zhmoginov, A., Mar, C. V, 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks.

Sardogan, M., 2018. Plant Leaf Disease Detection and Classification # ased on CNN with LVQ Algorithm. International Conference on Computer Science and Engineering (UBMK) 6–9. https://doi.org/10.1109/UBMK.2018.8566635

Savary, S., Ficke, A., Aubertot, J.N., Hollier, C., 2012. Crop losses due to diseases and their implications for global food production losses and food security. Food Secur 4, 519–537. https://doi.org/10.1007/s12571-012-0200-5

Singh, D., Jain, N., Jain, P., Kayal, P., Pepper, B., Blight, E., 2020. PlantDoc : A Dataset for Visual Plant Disease Detection. CoDS COMAD. https://doi.org/10.1145/3371158.3371196

Singh, V., Misra, A.K., 2017. Detection of plant leaf diseases using image segmentation and soft computing techniques. Information Processing in Agriculture 4, 41–49. https://doi.org/10.1016/j.inpa.2016.10.005

Singh, V., Varsha, Misra, A.K., 2015. Detection of unhealthy region of plant leaves using image processing and genetic algorithm. Conference Proceeding - 2015 International Conference on Advances in Computer Engineering and Applications, ICACEA 2015 1028–1032. https://doi.org/10.1109/ICACEA.2015.7164858

Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., Stefanovic, D., 2016. Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification. Comput Intell Neurosci 2016. https://doi.org/10.1155/2016/3289801

Srivastava, N., 2014. Dropout : A Simple Way to Prevent Neural Networks from Overfitting 15, 1929–1958.

Statista, India - production volume of tomatoes [WWW Document], n.d. URL https://www.statista.com/statistics/1039712/india-production-volume-of-tomatoes/#statisticContainer (accessed 1.18.21).

Suryawati, E., Sustika, R., Yuwana, R.S., Subekti, A., Pardede, H.F., 2018. Deep structured convolutional neural network for tomato diseases detection. 2018 International Conference on Advanced Computer Science and Information Systems, ICACSIS 2018 385–390. https://doi.org/10.1109/ICACSIS.2018.8618169

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 07-12-June, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Tan, M., Le, Q. V., 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. ArXiv.

The Evolution Of Mobile CNN Architectures | Weights & Biases [WWW Document], 2020. . Weights & Biases. URL https://wandb.ai/carlolepelaars/mobile_architectures/reports/The-Evolution-Of-Mobile-CNN-Architectures--VmlldzoyMDQ0ODQ (accessed 1.17.21).

Tm, P., Pranathi, A., Ashritha, K.S., Chittaragi, N.B., Koolagudi, S.G., 2018. Tomato Leaf Disease Detection using Convolutional Neural Networks 2–4.

United Nations, 2019. World Population Prospects.

Vaishnnave, M.P., Devi, K.S., 2019. Diseases using KNN classifier 1–5.

Wang, G., Sun, Y., Wang, J., 2017. Automatic Image-Based Plant Disease Severity Estimation Using Deep Learning. Comput Intell Neurosci 2017. https://doi.org/10.1155/2017/2917536

World Bank Group, A. and F.O., 2020. Agriculture Overview [WWW Document]. URL https://www.worldbank.org/en/topic/agriculture/overview (accessed 2.7.21).

Yun, J.W., 2015. Deep Residual Learning for Image Recognition arXiv:1512.03385v1. Enzyme Microb Technol 19, 107–117.

APPENDIX A: DATASETS AND SOURCE CODE DETAILS

Due to the large size of datasets, those are uploaded into Google Drive and shared. This table provides the details of all the datasets used in this research study.

#	Datasets	Location Details	Size (MB)
1	PlantVillage Original Dataset	https://drive.google.com/drive/folders/1XCQMHC76xys-FzP82WM6Fc7hrQ3Jd2EE?usp=sharing	239
2	PlantDoc Original Dataset	https://drive.google.com/drive/folders/1yozH54yDqjgAmRydDN2o_Qb-jvJNcs7y?usp=sharing	224
3	PlantDoc Cropped Dataset	https://drive.google.com/drive/folders/1QcmYlu9yYITrIPRvWa2-ueCIXSo2bT31?usp=sharing	111
4	PlantVillage Balanced Dataset (Under-sampled)	https://drive.google.com/drive/folders/1VbtlUYWUK_s5n1FaH4PhD6dZLPX44na3?usp=sharing	142
5	PlantDoc Balanced Dataset (Under-sampled)	https://drive.google.com/drive/folders/10Mb15hdxHAAmawXQG2NTDnaKMqaR-M8a?usp=sharing	161
6	PlantDoc Balanced Cropped Dataset (Over-sampled)	https://drive.google.com/drive/folders/1gmzLYeBkvTiGZr3JTBKtSXiI0GXWsmB-?usp=sharing	482
7	PlantDoc Severity Dataset	https://drive.google.com/drive/folders/1UMBeDbWRF08xhCBlugag9vrOunluut3X?usp=sharing	107

The source code for all the notebooks created for Data Preparation, Data Exploration, and Analysis, 60+ CNN models for Disease Identification, Machine Learning Classifier models, and CNN models for Plant Severity Detection are included in the uploaded ZIP file (Source Code.zip) along with the Final Thesis Report. This table provides the details of each notebook and the intention of building the notebook (what it covers).

#	Notebook File Name	Description
1	Step1A-Data-Preparation.ipynb	Create dataset variations such as a balanced dataset using the over-sampling technique.
2	Step1B1-Exploratory-Data-Analysis.ipynb.ipynb	Conduct Exploratory Data Analysis using PlantVillage and PlantDoc datasets.
3	Step1B2-Exploratory-Data-Analysis.ipynb.ipynb	Additional Exploratory Data Analysis using PlantVillage and PlantDoc datasets to generate the preview of sample images and augmented images.

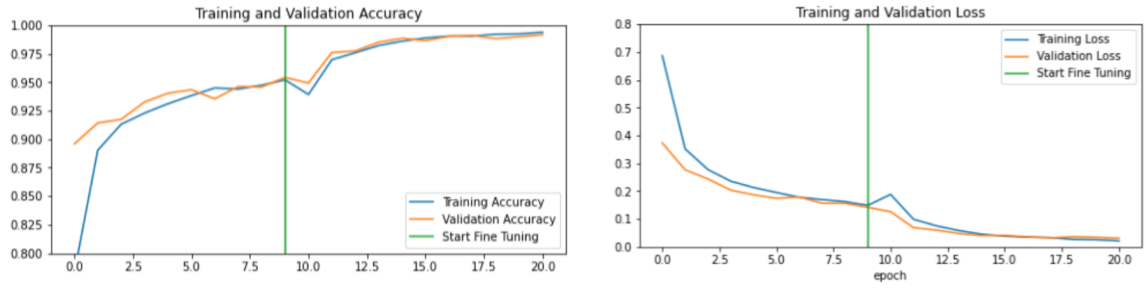
4	Step2-Split-Dataset.ipynb	Split PlantVillage dataset into Train and Validation datasets.
5	Step3A1-Model-Training-MobileNet(Default).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V1 default architecture.
6	Step3A2-Model-Training-MobileNet(Alpha-0.5).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V1 reduced width (50% parameters) architecture.
7	Step3A3-Model-Training-MobileNet(Balanced).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V1 default architecture against PlantVillage and PlantDoc Balanced datasets.
8	Step3B1-Model-Training-MobileNet-V2(Default).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V2 default architecture.
9	Step3B2-Model-Training-MobileNet-V2(Alpha-0.5).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V2 reduced width (50% parameters) architecture.
10	Step3B3-Model-Training-MobileNetV2(Balanced).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V2 default architecture against PlantVillage and PlantDoc Balanced datasets.
11	Step3B4-Model-Training-MobileNet-V2(Balanced-Alpha-0.5).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V2 default architecture against PlantVillage and PlantDoc Cropped Balanced datasets which are created using over-sampling techniques.
12	[128px]Step3B2-Model-Training-MobileNet-V2(Alpha-0.5).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V2 reduced width (50% parameters) architecture with input image size of (128,128,3).
13	[192px]Step3B2-Model-Training-MobileNet-V2(Alpha-0.5).ipynb	Train and evaluate the plant disease identification CNN models using MobileNet V2 reduced width (50% parameters) architecture with input image size of (192,192,3).
14	Step3C1-Model-Training-EfficientNet-B0(Default).ipynb	Train and evaluate the plant disease identification CNN models using EfficientNet B0 architecture.
15	Step3C2-Model-Training-EfficientNet-B0(Balanced).ipynb	Train and evaluate the plant disease identification CNN models using EfficientNet B0 against PlantVillage and PlantDoc Balanced datasets.

16	Step3D1-Model-Training-NasNetMobile(Default).ipynb	Train and evaluate the plant disease identification CNN models using NasNetMobile architecture.
17	Step3D2-Model-Training-NasNetMobile(Balanced).ipynb	Train and evaluate the plant disease identification CNN models using NasNetMobile against PlantVillage and PlantDoc Balanced datasets.
18	Step3E1-Model-Training-ResNet50V2(Default).ipynb	Train and evaluate the plant disease identification CNN models using ResNet50 V2 architecture.
19	Step3E2-Model-Training-ResNet50V2(Balanced).ipynb	Train and evaluate the plant disease identification CNN models using ResNet50 V2 against PlantVillage and PlantDoc Balanced datasets.
20	Step4-Model-Evaluation-Prediction-TFLite-Export.ipynb	Validate the "Final" CNN model performance with PlantDoc and PlantVillage test dataset and convert the final model in TensorFlow Lite model.
21	Step5A-Disease-Severity-Prediction-ML-Models(RFC&SVM).ipynb	Perform Exploratory Data Analysis (EDA) and train the Machine Learning Model to predict Tomato Plant Disease Severity using Global (color, shape, and texture) and local (ORB) features.
22	Step5B1-Disease-Severity-Prediction-MobileNet-V2(Default).ipynb	Train and evaluate the CNN models to predict Tomato Plant Disease Severity using MobileNet V2 Default architecture.
23	Step5B2-Disease-Severity-Prediction-MobileNet-V2(Alpha-0.5).ipynb	Train and evaluate the CNN models to predict Tomato Plant Disease Severity using MobileNet V2 reduced (50% parameters) architecture.

APPENDIX B: CNN MODEL EVALUATION RESULTS AND ANALYSIS FOR ALL RESEARCH EXPERIMENTS

MobileNet V1 Models

Fine-tuned model (PlantVillage Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantVillage Dataset) – Classification Report

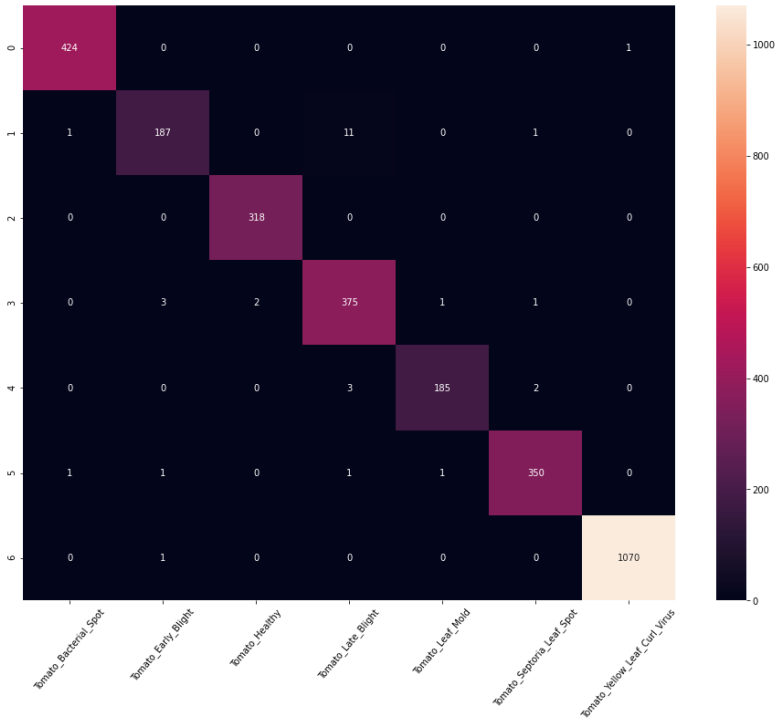
Model Evaluation Matrix

loss : 0.031517598778009415
tp : 2904.0
fp : 30.0
tn : 17610.0
fn : 36.0
accuracy : 0.9894557595252991
precision : 0.9897750616073608
recall : 0.9877551198005676
auc : 0.9999063014984131
prc : 0.9994492530822754

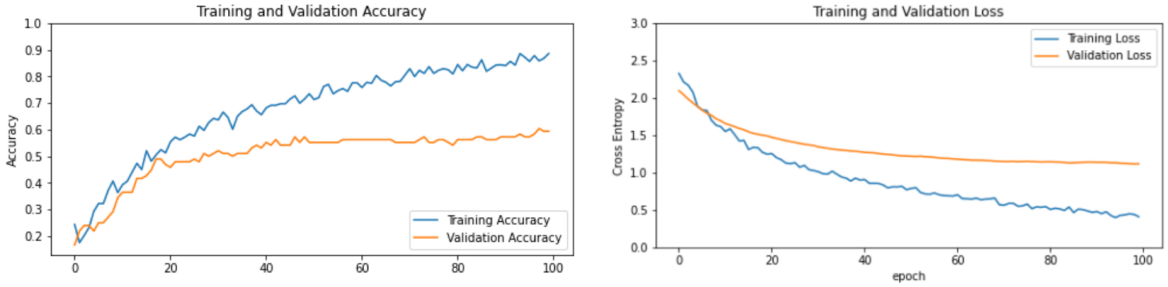
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	1.00	1.00	1.00	425
Tomato_Early_Blight	0.97	0.94	0.95	200
Tomato_Healthy	0.99	1.00	1.00	318
Tomato_Late_Blight	0.96	0.98	0.97	382
Tomato_Leaf_Mold	0.99	0.97	0.98	190
Tomato_Septoria_Leaf_Spot	0.99	0.99	0.99	354
Tomato_Yellow_Leaf_Curl_Virus	1.00	1.00	1.00	1071
accuracy			0.99	2940
macro avg	0.99	0.98	0.98	2940
weighted avg	0.99	0.99	0.99	2940

Fine-tuned model (PlantVillage Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Dataset) – Classification Report

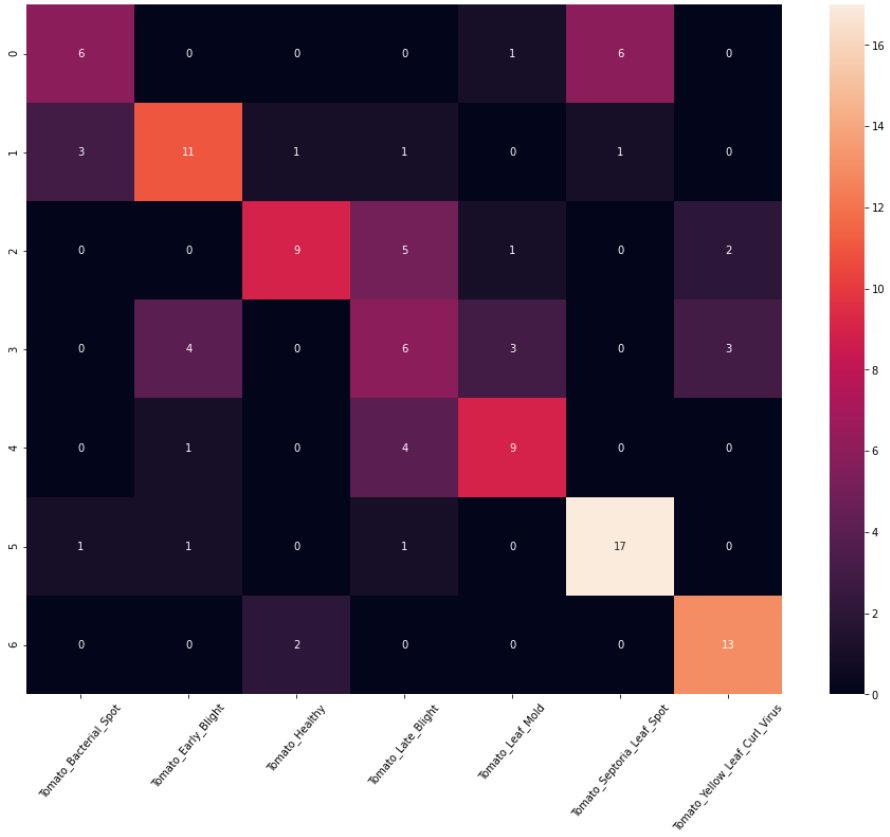
Model Evaluation Matrix

loss : 1.0363370180130005
tp : 59.0
fp : 21.0
tn : 651.0
fn : 53.0
accuracy : 0.6339285969734192
precision : 0.737500011920929
recall : 0.5267857313156128
auc : 0.9130261540412903
prc : 0.7136785984039307

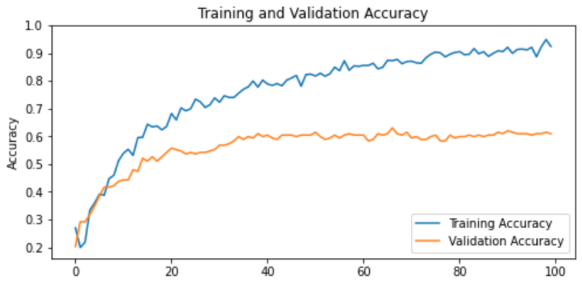
Classification Report

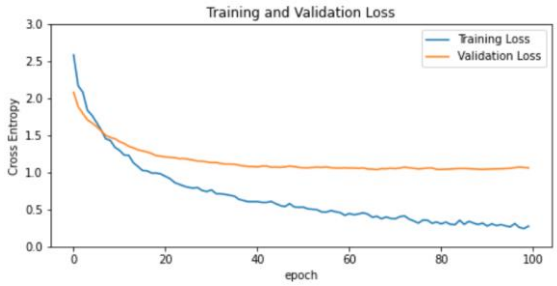
	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.60	0.46	0.52	13
Tomato_Early_Blight	0.65	0.65	0.65	17
Tomato_Healthy	0.75	0.53	0.62	17
Tomato_Late_Blight	0.35	0.38	0.36	16
Tomato_Leaf_Mold	0.64	0.64	0.64	14
Tomato_Septoria_Leaf_Spot	0.71	0.85	0.77	20
Tomato_Yellow_Leaf_Curl_Virus	0.72	0.87	0.79	15
accuracy			0.63	112
macro avg	0.63	0.62	0.62	112
weighted avg	0.64	0.63	0.63	112

Fine-tuned model (PlantDoc Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Cropped Dataset) – Train and Validation Accuracy & Loss



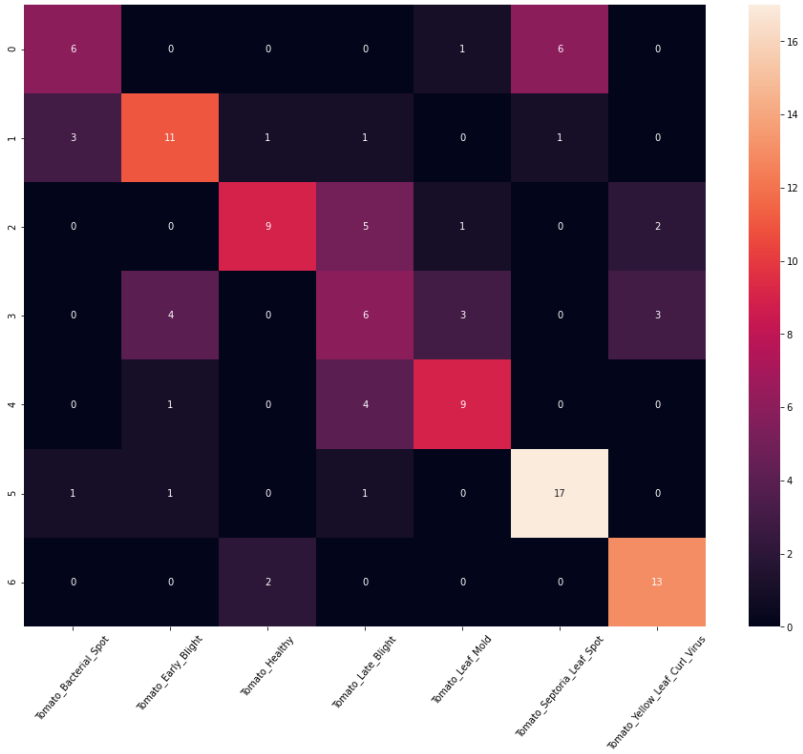


Fine-tuned model (PlantDoc Cropped Dataset) – Classification Report

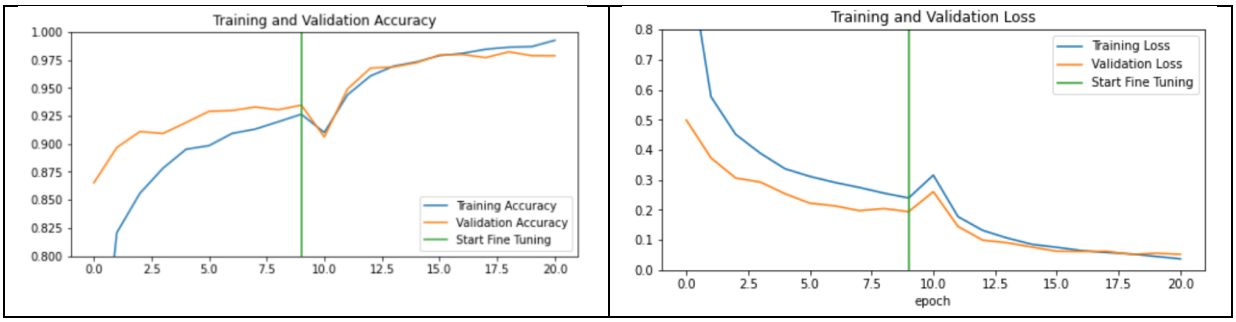
```
Model Evaluation Matrix

loss :  0.9698753952980042
tp :  112.0
fp :  44.0
tn :  1126.0
fn :  83.0
accuracy :  0.6358974575996399
precision :  0.7179487347602844
recall :  0.5743589997291565
auc :  0.923473596572876
prc :  0.7318645715713501
```

```
Classification Report
                                precision    recall  f1-score   support

        Tomato_Bacterial_Spot       0.54      0.41      0.46        32
          Tomato_Early_Blight       0.62      0.56      0.59        36
              Tomato_Healthy        0.75      0.68      0.71        31
           Tomato_Late_Blight       0.44      0.69      0.54        16
             Tomato_Leaf_Mold       0.69      0.85      0.76        26
    Tomato_Septoria_Leaf_Spot       0.59      0.59      0.59        34
Tomato_Yellow_Leaf_Curl_Virus       0.85      0.85      0.85        20

                     accuracy                           0.64       195
                    macro avg       0.64      0.66      0.64       195
                 weighted avg       0.64      0.64      0.63       195
```

Fine-tuned model (PlantDoc Cropped Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantVillage Balanced Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantVillage Balanced Dataset) – Classification Report

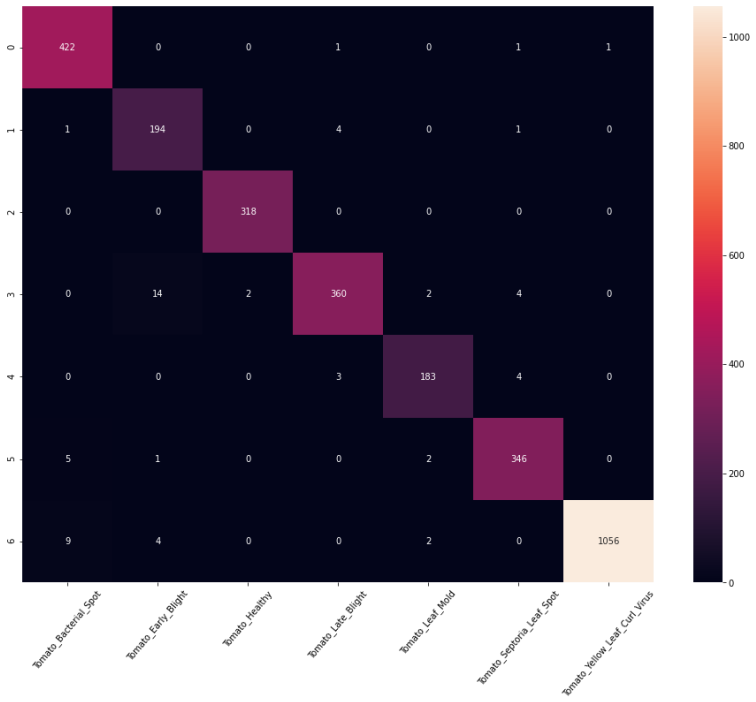
Model Evaluation Matrix

loss : 0.05138034000992775
tp : 2877.0
fp : 52.0
tn : 17588.0
fn : 63.0
accuracy : 0.9792516827583313
precision : 0.9822465181350708
recall : 0.9785714149475098
auc : 0.9997764229774475
prc : 0.9986932277679443

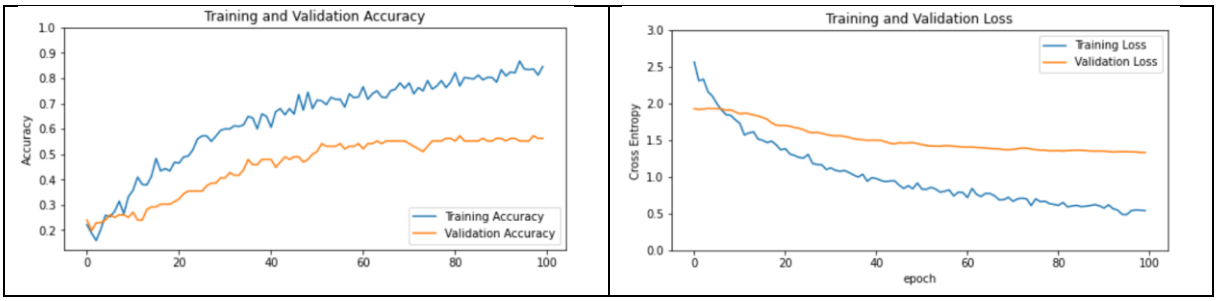
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.97	0.99	0.98	425
Tomato_Early_Blight	0.91	0.97	0.94	200
Tomato_Healthy	0.99	1.00	1.00	318
Tomato_Late_Blight	0.98	0.94	0.96	382
Tomato_Leaf_Mold	0.97	0.96	0.97	190
Tomato_Septoria_Leaf_Spot	0.97	0.98	0.97	354
Tomato_Yellow_Leaf_Curl_Virus	1.00	0.99	0.99	1071
accuracy			0.98	2940
macro avg	0.97	0.98	0.97	2940
weighted avg	0.98	0.98	0.98	2940

Fine-tuned model (PlantVillage Balanced Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Balanced Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Balanced Dataset) – Classification Report

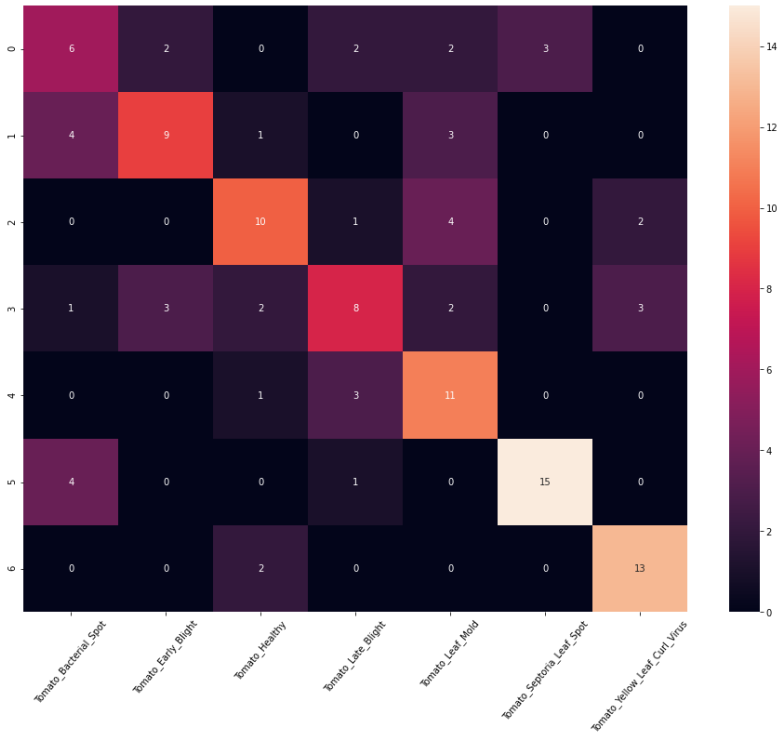
Model Evaluation Matrix

loss : 1.2522138357162476
tp : 51.0
fp : 15.0
tn : 693.0
fn : 67.0
accuracy : 0.6101694703102112
precision : 0.7727272510528564
recall : 0.43220338225364685
auc : 0.8655319213867188
prc : 0.6631430387496948

Classification Report

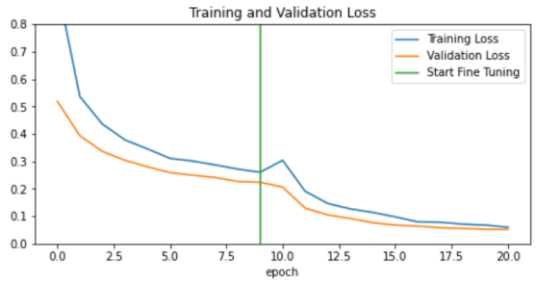
	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.40	0.40	0.40	15
Tomato_Early_Blight	0.64	0.53	0.58	17
Tomato_Healthy	0.62	0.59	0.61	17
Tomato_Late_Blight	0.53	0.42	0.47	19
Tomato_Leaf_Mold	0.50	0.73	0.59	15
Tomato_Septoria_Leaf_Spot	0.83	0.75	0.79	20
Tomato_Yellow_Leaf_Curl_Virus	0.72	0.87	0.79	15
accuracy			0.61	118
macro avg	0.61	0.61	0.60	118
weighted avg	0.62	0.61	0.61	118

Fine-tuned model (PlantDoc Balanced Dataset) – Confusion Matrix Heatmap



MobileNet V1 Alpha 0.5 (50%) Models

Fine-tuned model (PlantVillage Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantVillage Dataset) – Classification Report

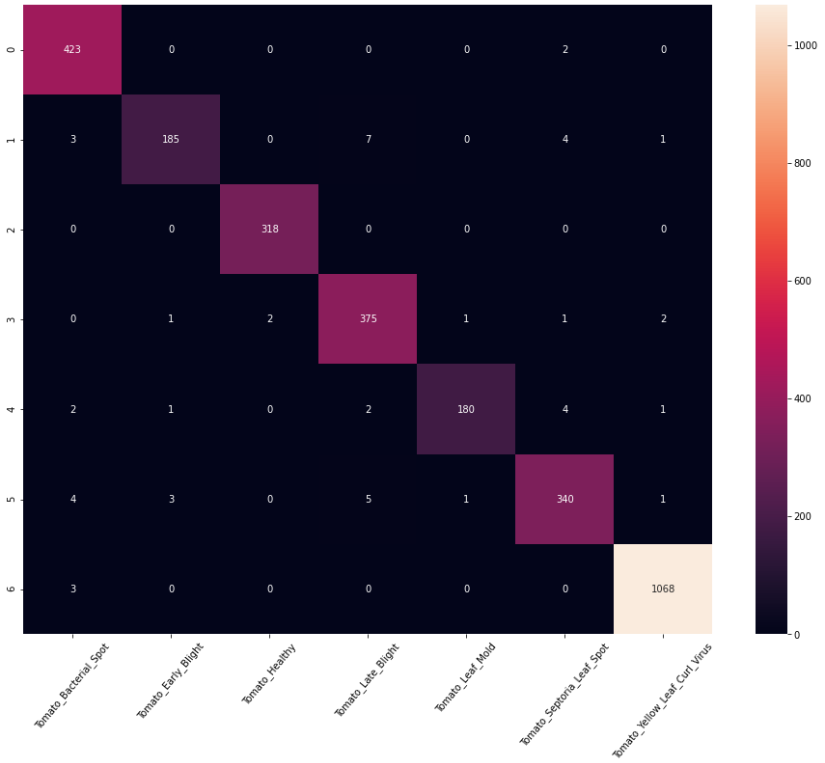
Model Evaluation Matrix

```
loss :  0.05066238343715668
tp :  2884.0
fp :  40.0
tn :  17600.0
fn :  56.0
accuracy :  0.9826530814170837
precision :  0.9863201379776001
recall :  0.9809523820877075
auc :  0.9994398951530457
prc :  0.99834805727005
```

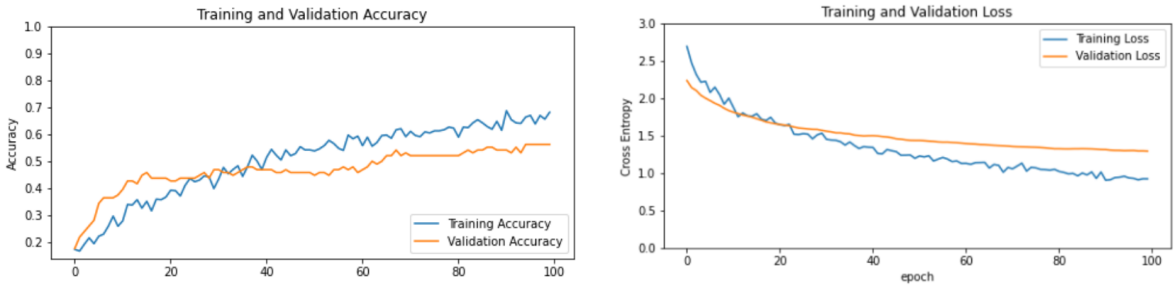
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.97	1.00	0.98	425
Tomato_Early_Blight	0.97	0.93	0.95	200
Tomato_Healthy	0.99	1.00	1.00	318
Tomato_Late_Blight	0.96	0.98	0.97	382
Tomato_Leaf_Mold	0.99	0.95	0.97	190
Tomato_Septoria_Leaf_Spot	0.97	0.96	0.96	354
Tomato_Yellow_Leaf_Curl_Virus	1.00	1.00	1.00	1071
accuracy			0.98	2940
macro avg	0.98	0.97	0.98	2940
weighted avg	0.98	0.98	0.98	2940

Fine-tuned model (PlantVillage Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Original Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Original Dataset) – Classification Report

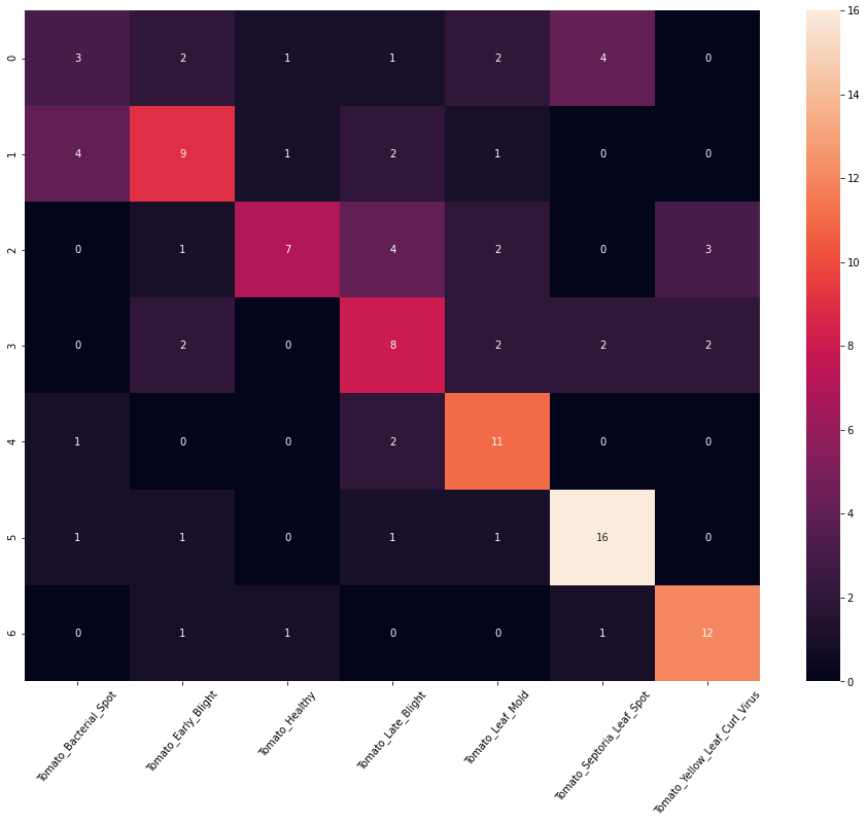
Model Evaluation Matrix

loss : 1.21323561668396
tp : 45.0
fp : 16.0
tn : 656.0
fn : 67.0
accuracy : 0.5892857313156128
precision : 0.7377049326896667
recall : 0.4017857015132904
auc : 0.8743954300880432
prc : 0.624873161315918

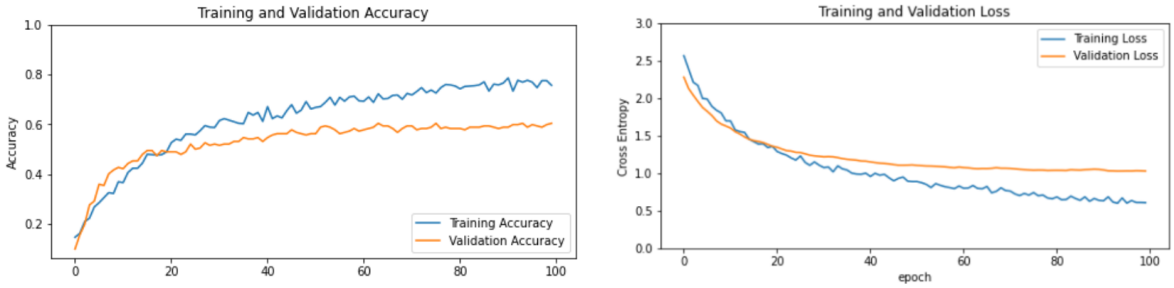
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.33	0.23	0.27	13
Tomato_Early_Blight	0.56	0.53	0.55	17
Tomato_Healthy	0.70	0.41	0.52	17
Tomato_Late_Blight	0.44	0.50	0.47	16
Tomato_Leaf_Mold	0.58	0.79	0.67	14
Tomato_Septoria_Leaf_Spot	0.70	0.80	0.74	20
Tomato_Yellow_Leaf_Curl_Virus	0.71	0.80	0.75	15
accuracy			0.59	112
macro avg	0.57	0.58	0.57	112
weighted avg	0.58	0.59	0.58	112

Fine-tuned model (PlantDoc Original Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Cropped Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Cropped Dataset) – Classification Report

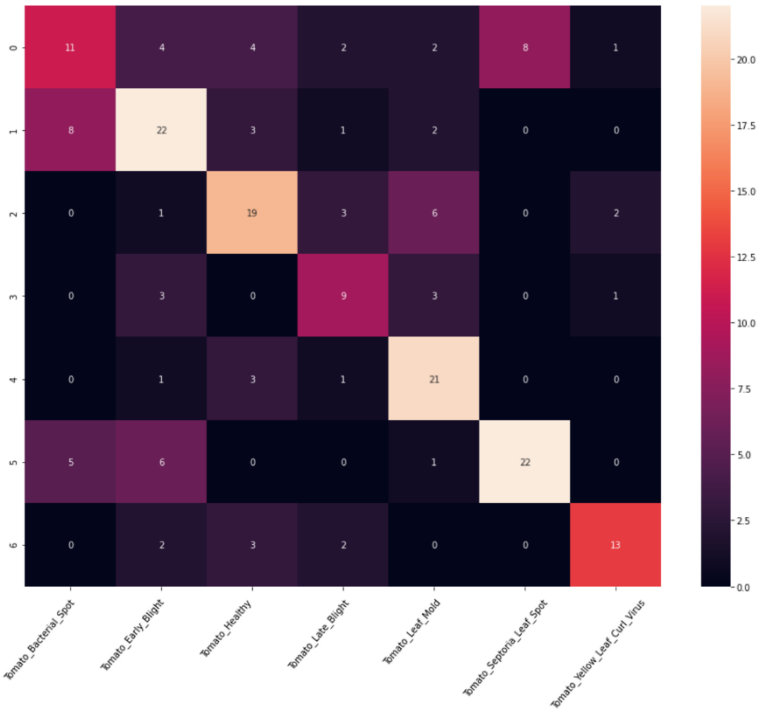
Model Evaluation Matrix

```
loss :  1.035414457321167
tp :  97.0
fp :  41.0
tn :  1129.0
fn :  98.0
accuracy :  0.6000000238418579
precision :  0.7028985619544983
recall :  0.49743589758872986
auc :  0.9114552140235901
prc :  0.6907453536987305
```

Classification Report

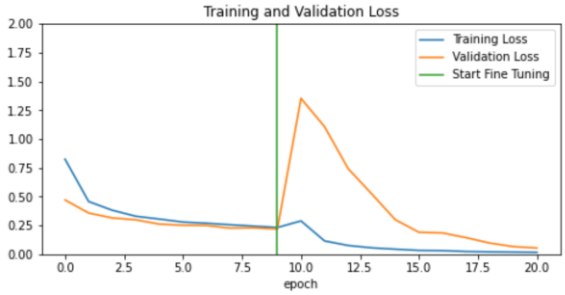
	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.46	0.34	0.39	32
Tomato_Early_Blight	0.56	0.61	0.59	36
Tomato_Healthy	0.59	0.61	0.60	31
Tomato_Late_Blight	0.50	0.56	0.53	16
Tomato_Leaf_Mold	0.60	0.81	0.69	26
Tomato_Septoria_Leaf_Spot	0.73	0.65	0.69	34
Tomato_Yellow_Leaf_Curl_Virus	0.76	0.65	0.70	20
accuracy			0.60	195
macro avg	0.60	0.61	0.60	195
weighted avg	0.60	0.60	0.60	195

Fine-tuned model (PlantDoc Cropped Dataset) – Confusion Matrix Heatmap



MobileNet V2 Models

Fine-tuned model (PlantVillage Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantVillage Dataset) – Classification Report

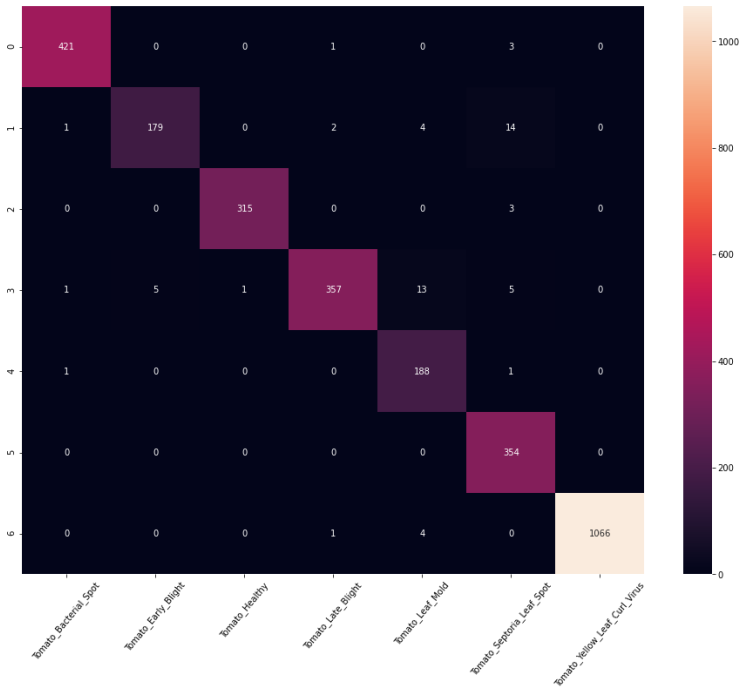
Model Evaluation Matrix

```
loss :  0.05367273464798927
tp :  2878.0
fp :  58.0
tn :  17582.0
fn :  62.0
accuracy :  0.9795918464660645
precision :  0.9802452325820923
recall :  0.9789115786552429
auc :  0.9995782971382141
prc :  0.9983317852020264
```

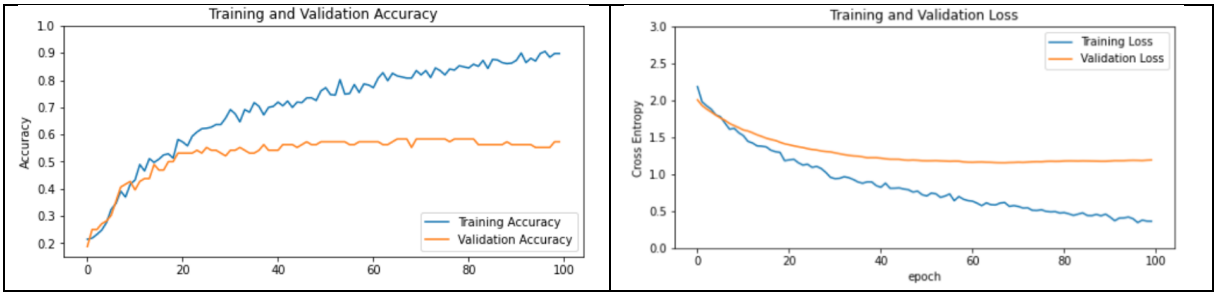
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.99	0.99	0.99	425
Tomato_Early_Blight	0.97	0.90	0.93	200
Tomato_Healthy	1.00	0.99	0.99	318
Tomato_Late_Blight	0.99	0.93	0.96	382
Tomato_Leaf_Mold	0.90	0.99	0.94	190
Tomato_Septoria_Leaf_Spot	0.93	1.00	0.96	354
Tomato_Yellow_Leaf_Curl_Virus	1.00	1.00	1.00	1071
accuracy			0.98	2940
macro avg	0.97	0.97	0.97	2940
weighted avg	0.98	0.98	0.98	2940

Fine-tuned model (PlantVillage Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Original Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Original Dataset) – Classification Report

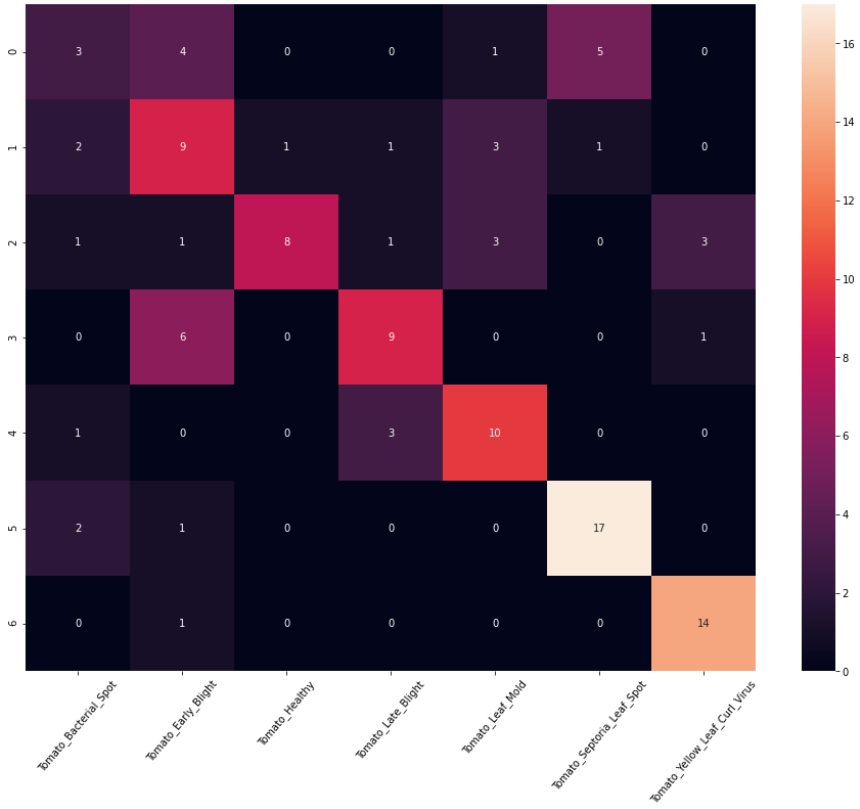
Model Evaluation Matrix

loss : 1.069245457649231
tp : 64.0
fp : 27.0
tn : 645.0
fn : 48.0
accuracy : 0.625
precision : 0.7032967209815979
recall : 0.5714285969734192
auc : 0.910946786403656
prc : 0.7067406177520752

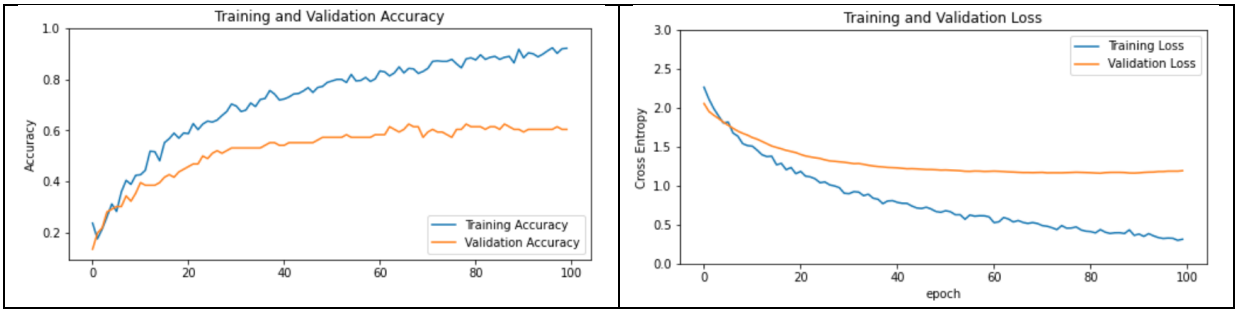
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.33	0.23	0.27	13
Tomato_Early_Blight	0.41	0.53	0.46	17
Tomato_Healthy	0.89	0.47	0.62	17
Tomato_Late_Blight	0.64	0.56	0.60	16
Tomato_Leaf_Mold	0.59	0.71	0.65	14
Tomato_Septoria_Leaf_Spot	0.74	0.85	0.79	20
Tomato_Yellow_Leaf_Curl_Virus	0.78	0.93	0.85	15
accuracy			0.62	112
macro avg	0.63	0.61	0.60	112
weighted avg	0.64	0.62	0.62	112

Fine-tuned model (PlantDoc Original Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Cropped Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Cropped Dataset) – Classification Report

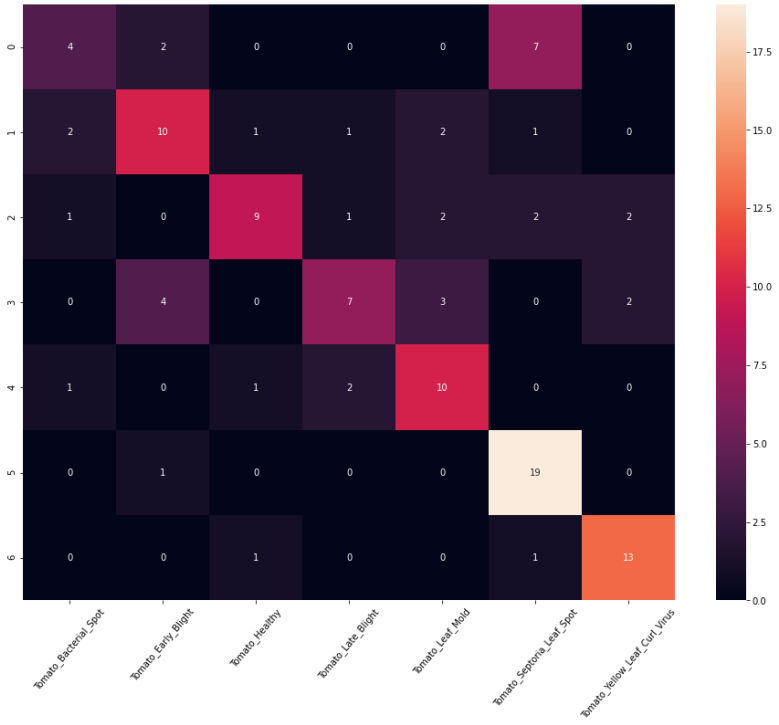
Model Evaluation Matrix

loss : 1.1005865335464478
tp : 63.0
fp : 21.0
tn : 651.0
fn : 49.0
accuracy : 0.6428571343421936
precision : 0.75
recall : 0.5625
auc : 0.904482901096344
prc : 0.7013537883758545

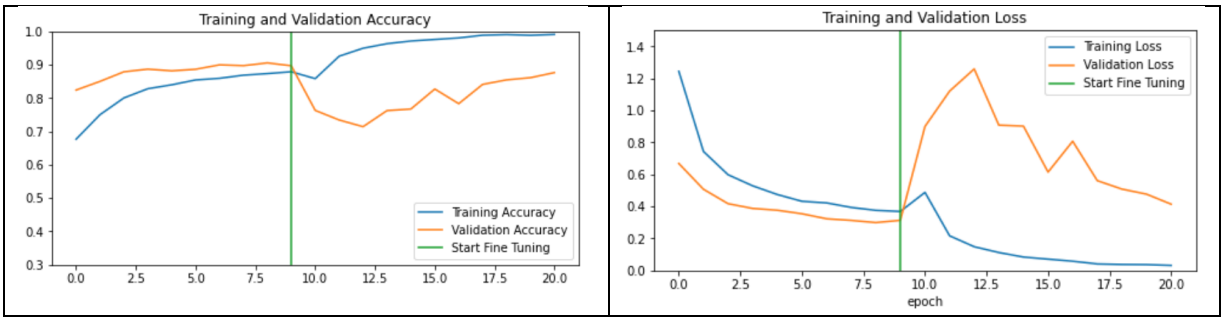
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.50	0.31	0.38	13
Tomato_Early_Blight	0.59	0.59	0.59	17
Tomato_Healthy	0.75	0.53	0.62	17
Tomato_Late_Blight	0.64	0.44	0.52	16
Tomato_Leaf_Mold	0.59	0.71	0.65	14
Tomato_Septoria_Leaf_Spot	0.63	0.95	0.76	20
Tomato_Yellow_Leaf_Curl_Virus	0.76	0.87	0.81	15
accuracy			0.64	112
macro avg	0.64	0.63	0.62	112
weighted avg	0.64	0.64	0.63	112

Fine-tuned model (PlantDoc Cropped Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantVillage Balanced Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantVillage Balanced Dataset) – Classification Report

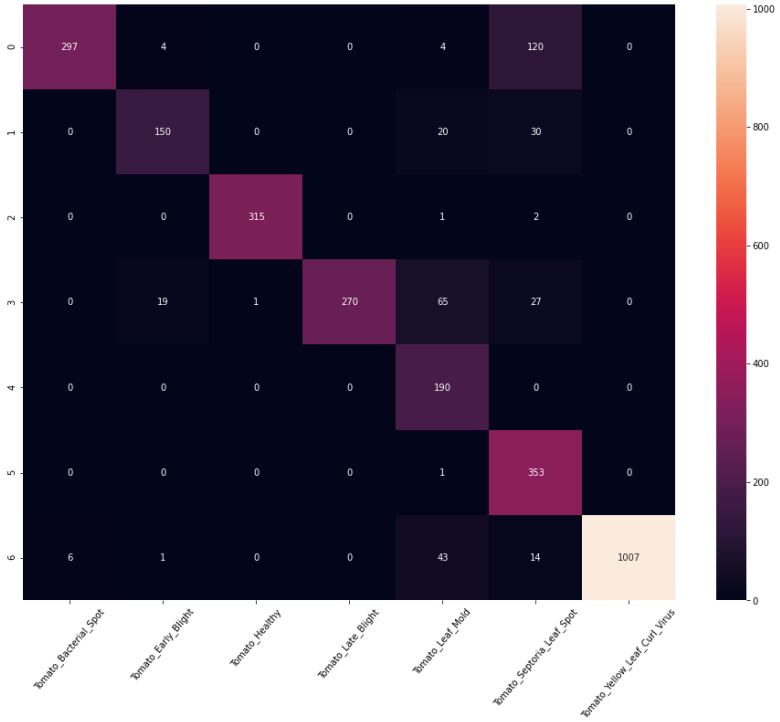
Model Evaluation Matrix

loss : 0.40935274958610535
tp : 2575.0
fp : 338.0
tn : 17302.0
fn : 365.0
accuracy : 0.8782312870025635
precision : 0.8839684128761292
recall : 0.8758503198623657
auc : 0.9828047752380371
prc : 0.9398378133773804

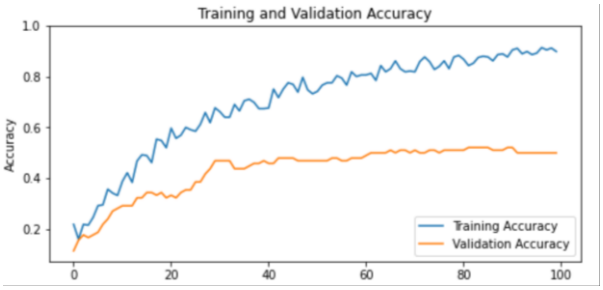
Classification Report

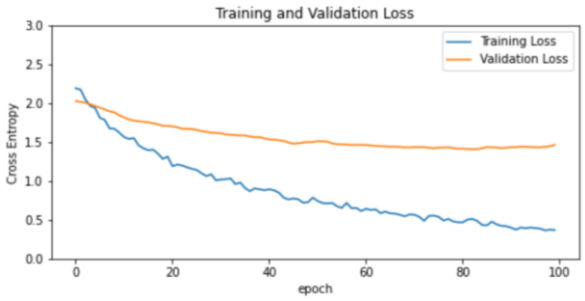
	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.98	0.70	0.82	425
Tomato_Early_Blight	0.86	0.75	0.80	200
Tomato_Healthy	1.00	0.99	0.99	318
Tomato_Late_Blight	1.00	0.71	0.83	382
Tomato_Leaf_Mold	0.59	1.00	0.74	190
Tomato_Septoria_Leaf_Spot	0.65	1.00	0.78	354
Tomato_Yellow_Leaf_Curl_Virus	1.00	0.94	0.97	1071
accuracy			0.88	2940
macro avg	0.87	0.87	0.85	2940
weighted avg	0.92	0.88	0.88	2940

Fine-tuned model (PlantVillage Balanced Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Balanced Dataset) – Train and Validation Accuracy & Loss





Fine-tuned model (PlantDoc Balanced Dataset) – Classification Report

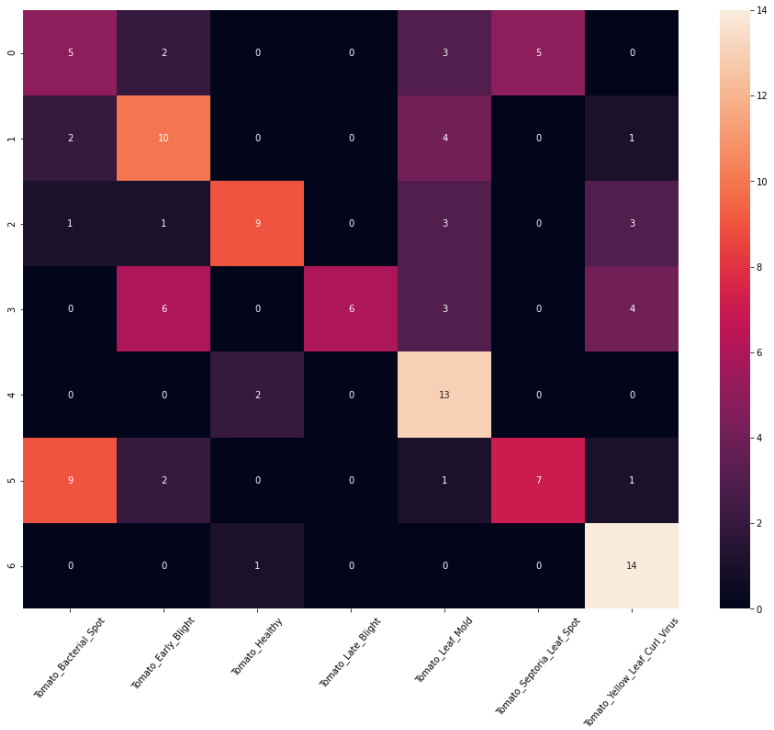
Model Evaluation Matrix

```
loss :  1.2993786334991455
tp :  55.0
fp :  31.0
tn :  677.0
fn :  63.0
accuracy :  0.5423728823661804
precision :  0.6395348906517029
recall :  0.4661017060279846
auc :  0.8709542751312256
prc :  0.6064140796661377
```

Classification Report

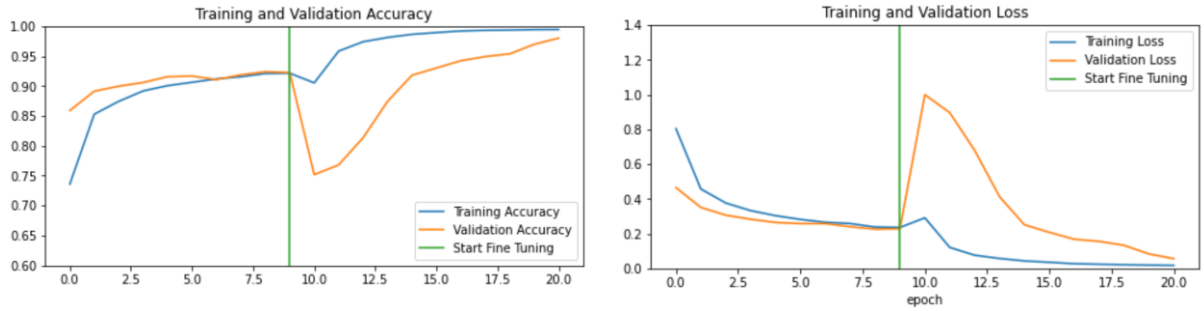
	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.29	0.33	0.31	15
Tomato_Early_Blight	0.48	0.59	0.53	17
Tomato_Healthy	0.75	0.53	0.62	17
Tomato_Late_Blight	1.00	0.32	0.48	19
Tomato_Leaf_Mold	0.48	0.87	0.62	15
Tomato_Septoria_Leaf_Spot	0.58	0.35	0.44	20
Tomato_Yellow_Leaf_Curl_Virus	0.61	0.93	0.74	15
accuracy			0.54	118
macro avg	0.60	0.56	0.53	118
weighted avg	0.61	0.54	0.53	118

Fine-tuned model (PlantDoc Balanced Dataset) – Confusion Matrix Heatmap



MobileNet V2 Alpha 0.5 (50%) Models

Fine-tuned model (PlantVillage Dataset) – Train and Validation Accuracy & Loss

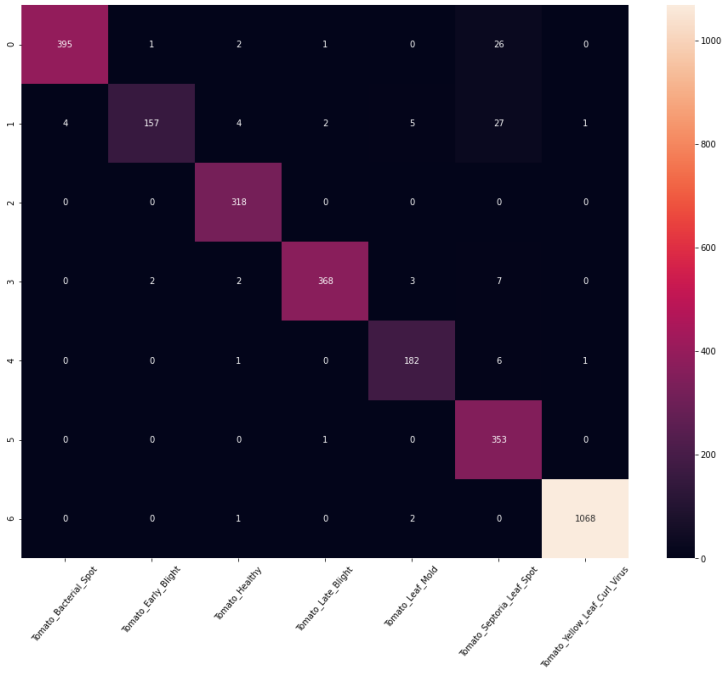


Fine-tuned model (PlantVillage Dataset) – Classification Report

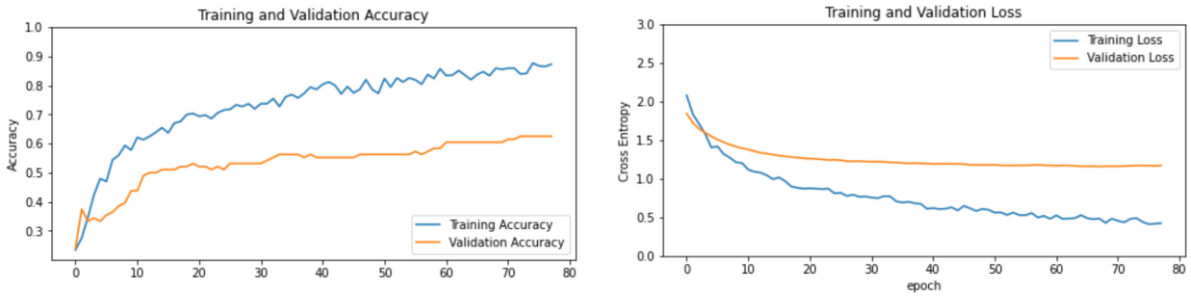
```
Model Evaluation Matrix

loss :  0.08517277240753174
tp :  2834.0
fp :  97.0
tn :  17543.0
fn :  106.0
accuracy :  0.9663265347480774
precision :  0.9669054746627808
recall :  0.9639455676078796
auc :  0.9983768463134766
prc :  0.9949300289154053
```

```
Classification Report
                                precision    recall  f1-score   support

          Tomato_Bacterial_Spot       0.99      0.93      0.96       425
            Tomato_Early_Blight       0.98      0.79      0.87       200
                 Tomato_Healthy       0.97      1.00      0.98       318
            Tomato_Late_Blight       0.99      0.96      0.98       382
              Tomato_Leaf_Mold       0.95      0.96      0.95       190
     Tomato_Septoria_Leaf_Spot       0.84      1.00      0.91       354
Tomato_Yellow_Leaf_Curl_Virus        1.00      1.00      1.00      1071

                      accuracy                           0.97      2940
                     macro avg       0.96      0.95      0.95      2940
                  weighted avg       0.97      0.97      0.97      2940
```

Fine-tuned model (PlantVillage Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Original Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Original Dataset) – Classification Report

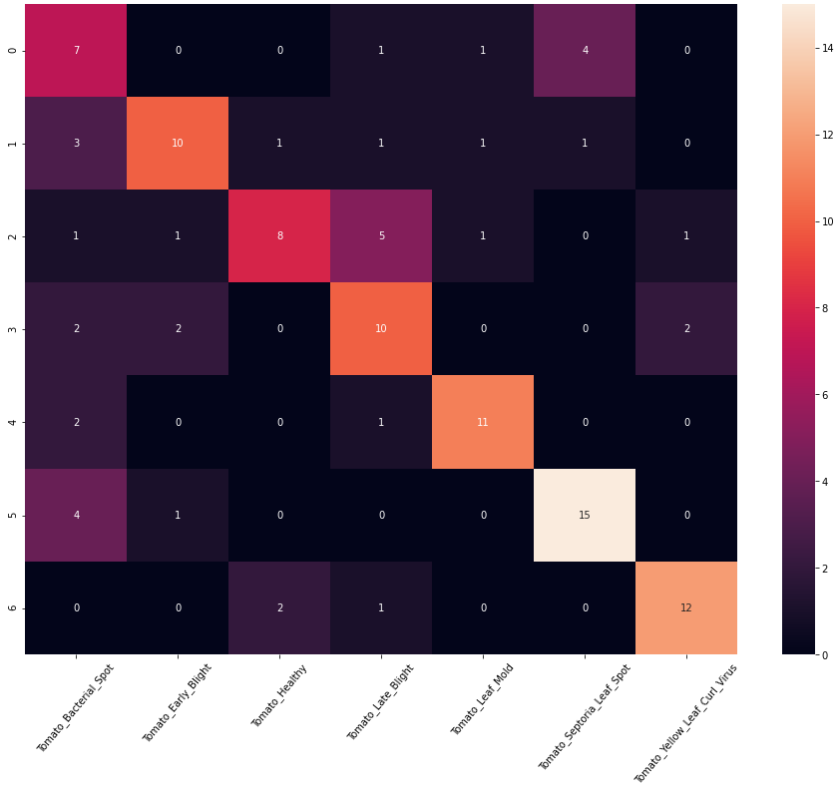
Model Evaluation Matrix

```
loss :  1.0924501419067383
tp :  62.0
fp :  29.0
tn :  643.0
fn :  50.0
accuracy :  0.6517857313156128
precision :  0.6813187003135681
recall :  0.5535714030265808
auc :  0.9040178060531616
prc :  0.6737455725669861
```

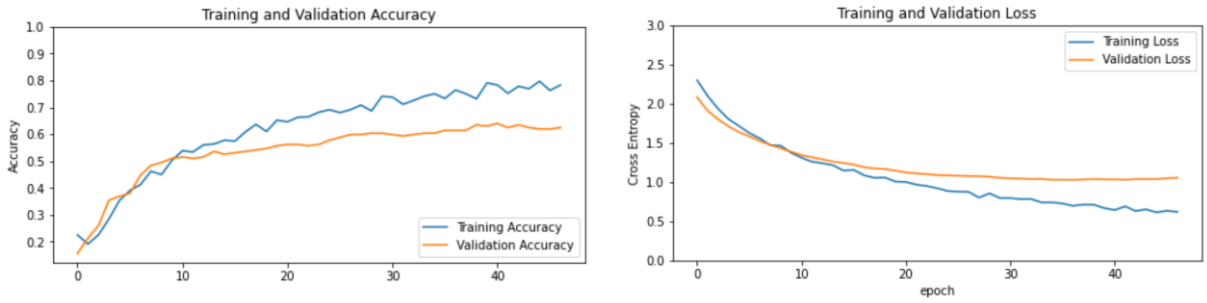
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.33	0.23	0.27	13
Tomato_Early_Blight	0.56	0.53	0.55	17
Tomato_Healthy	0.70	0.41	0.52	17
Tomato_Late_Blight	0.44	0.50	0.47	16
Tomato_Leaf_Mold	0.58	0.79	0.67	14
Tomato_Septoria_Leaf_Spot	0.70	0.80	0.74	20
Tomato_Yellow_Leaf_Curl_Virus	0.71	0.80	0.75	15
accuracy			0.59	112
macro avg	0.57	0.58	0.57	112
weighted avg	0.58	0.59	0.58	112

Fine-tuned model (PlantDoc Original Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Cropped Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Cropped Dataset) – Classification Report

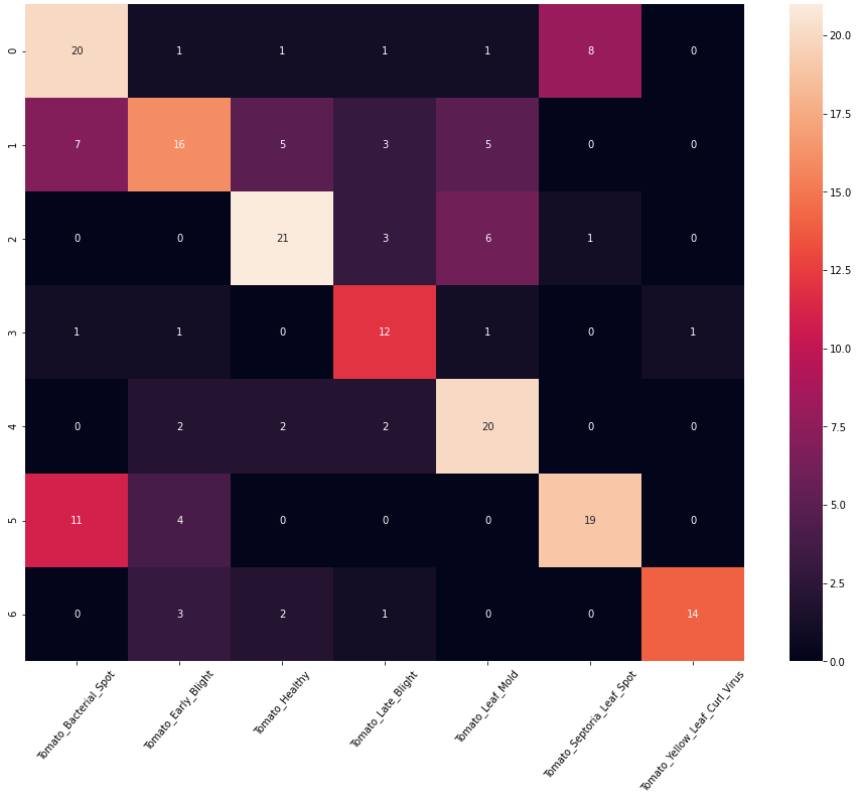
Model Evaluation Matrix

loss : 1.0628308057785034
tp : 105.0
fp : 43.0
tn : 1127.0
fn : 90.0
accuracy : 0.6256410479545593
precision : 0.7094594836235046
recall : 0.5384615659713745
auc : 0.9066798686981201
prc : 0.6961241960525513

Classification Report

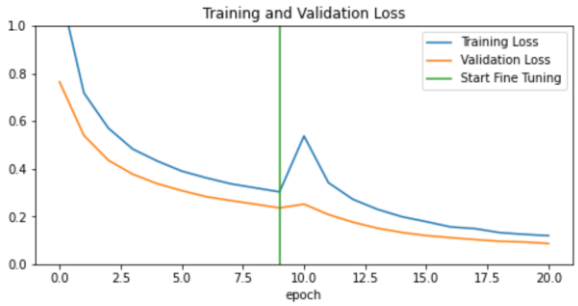
	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.51	0.62	0.56	32
Tomato_Early_Blight	0.59	0.44	0.51	36
Tomato_Healthy	0.68	0.68	0.68	31
Tomato_Late_Blight	0.55	0.75	0.63	16
Tomato_Leaf_Mold	0.61	0.77	0.68	26
Tomato_Septoria_Leaf_Spot	0.68	0.56	0.61	34
Tomato_Yellow_Leaf_Curl_Virus	0.93	0.70	0.80	20
accuracy			0.63	195
macro avg	0.65	0.65	0.64	195
weighted avg	0.64	0.63	0.63	195

Fine-tuned model (PlantDoc Cropped Dataset) – Confusion Matrix Heatmap



EfficientNet B0 Models

Fine-tuned model (PlantVillage Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantVillage Dataset) – Classification Report

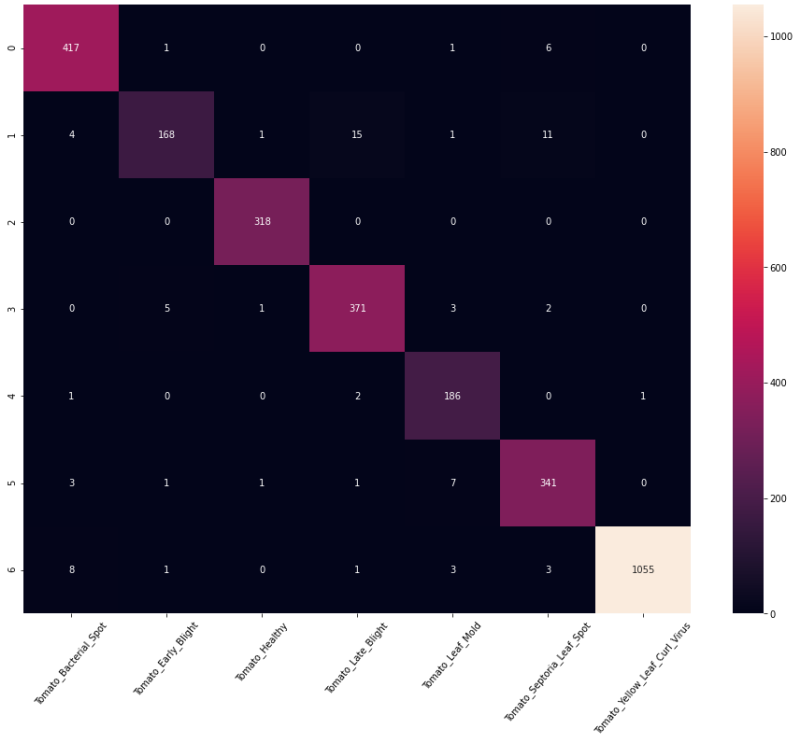
Model Evaluation Matrix

```
loss :  0.08531027287244797
tp :  2844.0
fp :  69.0
tn :  17571.0
fn :  96.0
accuracy :  0.9714285731315613
precision :  0.976313054561615
recall :  0.9673469662666321
auc :  0.9991397261619568
prc :  0.9957382678985596
```

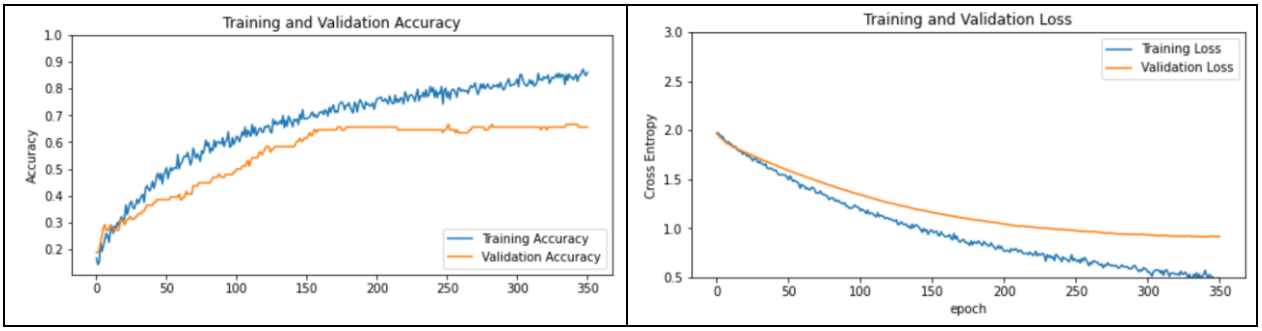
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.96	0.98	0.97	425
Tomato_Early_Blight	0.95	0.84	0.89	200
Tomato_Healthy	0.99	1.00	1.00	318
Tomato_Late_Blight	0.95	0.97	0.96	382
Tomato_Leaf_Mold	0.93	0.98	0.95	190
Tomato_Septoria_Leaf_Spot	0.94	0.96	0.95	354
Tomato_Yellow_Leaf_Curl_Virus	1.00	0.99	0.99	1071
accuracy			0.97	2940
macro avg	0.96	0.96	0.96	2940
weighted avg	0.97	0.97	0.97	2940

Fine-tuned model (PlantVillage Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Original Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Original Dataset) – Classification Report

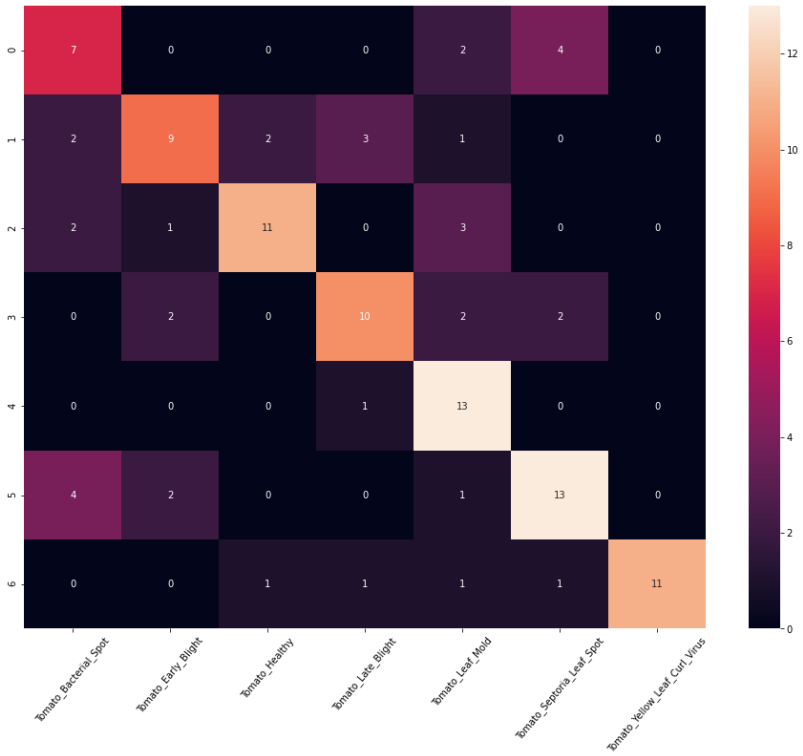
Model Evaluation Matrix

loss : 0.9371363520622253
tp : 64.0
fp : 16.0
tn : 656.0
fn : 48.0
accuracy : 0.6607142686843872
precision : 0.800000011920929
recall : 0.5714285969734192
auc : 0.9246983528137207
prc : 0.761119544506073

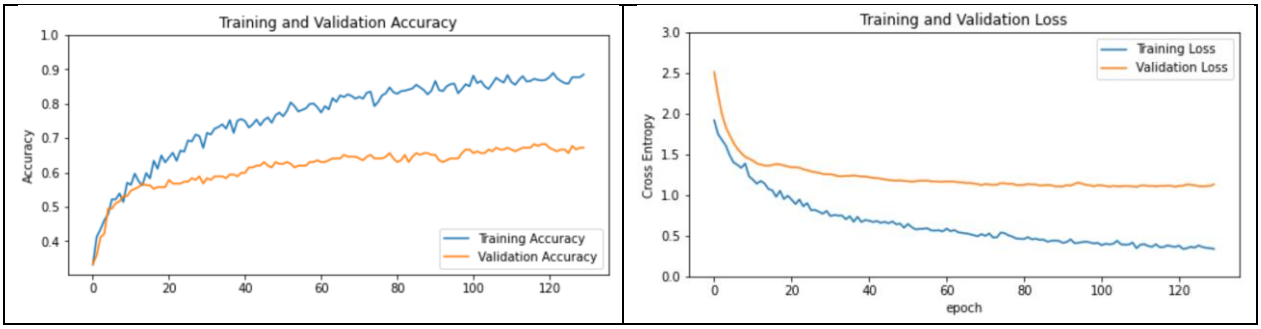
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.47	0.54	0.50	13
Tomato_Early_Blight	0.64	0.53	0.58	17
Tomato_Healthy	0.79	0.65	0.71	17
Tomato_Late_Blight	0.67	0.62	0.65	16
Tomato_Leaf_Mold	0.57	0.93	0.70	14
Tomato_Septoria_Leaf_Spot	0.65	0.65	0.65	20
Tomato_Yellow_Leaf_Curl_Virus	1.00	0.73	0.85	15
accuracy			0.66	112
macro avg	0.68	0.66	0.66	112
weighted avg	0.69	0.66	0.66	112

Fine-tuned model (PlantDoc Original Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Cropped Dataset) – Train and Validation Accuracy & Loss

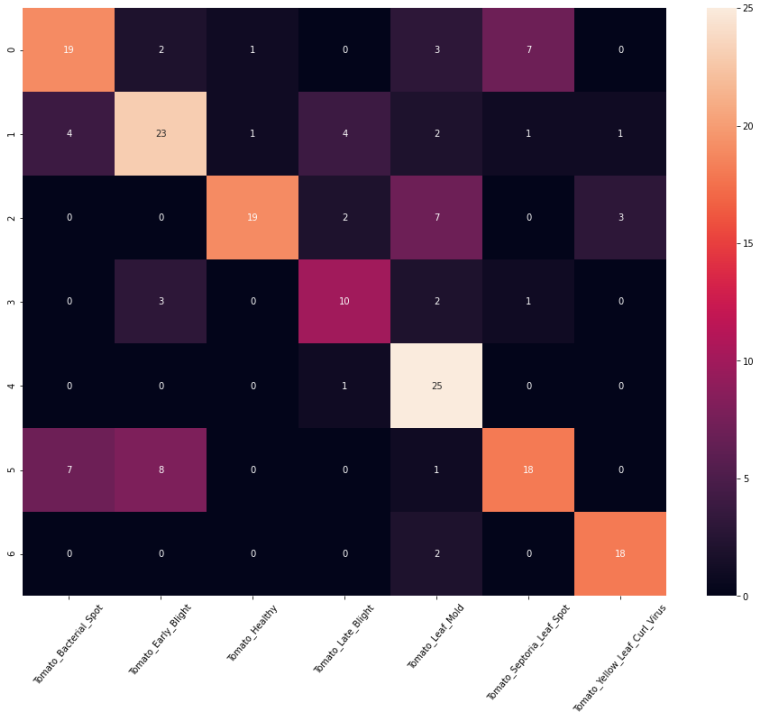


Fine-tuned model (PlantDoc Cropped Dataset) – Classification Report

```
Model Evaluation Matrix

loss :  1.1161253452301025
tp :  120.0
fp :  49.0
tn :  1121.0
fn :  75.0
accuracy :  0.6769230961799622
precision :  0.7100591659545898
recall :  0.6153846383094788
auc :  0.9148542284965515
prc :  0.7207858562469482
```

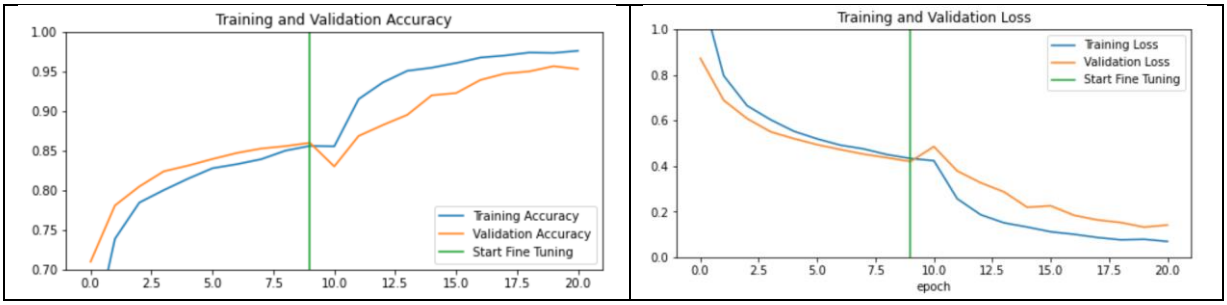
```
Classification Report
                                precision    recall  f1-score   support

          Tomato_Bacterial_Spot       0.63      0.59      0.61        32
            Tomato_Early_Blight       0.64      0.64      0.64        36
                 Tomato_Healthy       0.90      0.61      0.73        31
              Tomato_Late_Blight       0.59      0.62      0.61        16
                Tomato_Leaf_Mold       0.60      0.96      0.74        26
       Tomato_Septoria_Leaf_Spot       0.67      0.53      0.59        34
Tomato_Yellow_Leaf_Curl_Virus          0.82      0.90      0.86        20

                        accuracy                           0.68       195
                       macro avg       0.69      0.69      0.68       195
                    weighted avg       0.69      0.68      0.67       195
```

Fine-tuned model (PlantDoc Cropped Dataset) – Confusion Matrix Heatmap



NasNetMobile Models

Fine-tuned model (PlantVillage Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantVillage Dataset) – Classification Report

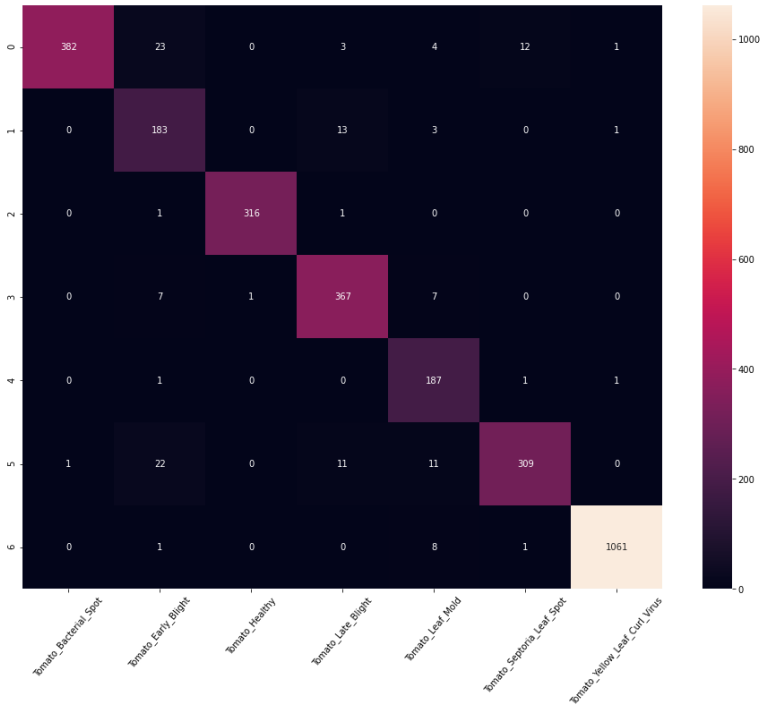
Model Evaluation Matrix

```
loss :  0.13814228773117065
tp :  2780.0
fp :  114.0
tn :  17526.0
fn :  160.0
accuracy :  0.954081654548645
precision :  0.9606081843337616
recall :  0.9455782175064087
auc :  0.9977254867553711
prc :  0.9908806085586548
```

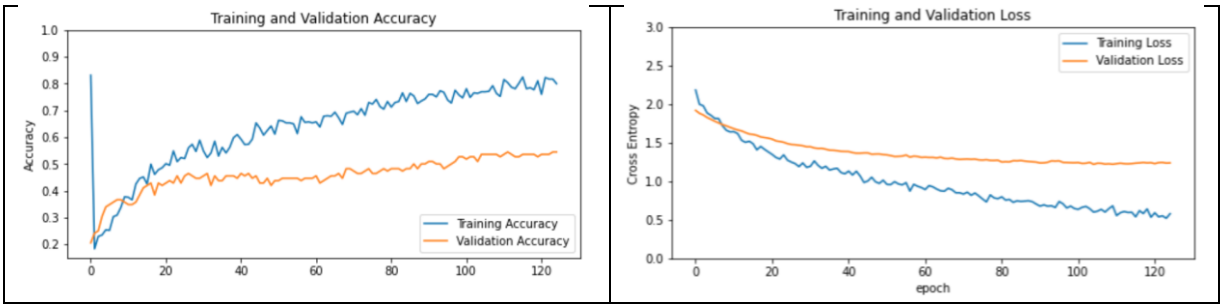
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	1.00	0.90	0.95	425
Tomato_Early_Blight	0.77	0.92	0.84	200
Tomato_Healthy	1.00	0.99	1.00	318
Tomato_Late_Blight	0.93	0.96	0.94	382
Tomato_Leaf_Mold	0.85	0.98	0.91	190
Tomato_Septoria_Leaf_Spot	0.96	0.87	0.91	354
Tomato_Yellow_Leaf_Curl_Virus	1.00	0.99	0.99	1071
accuracy			0.95	2940
macro avg	0.93	0.95	0.93	2940
weighted avg	0.96	0.95	0.95	2940

Fine-tuned model (PlantVillage Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Original Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Original Dataset) – Classification Report

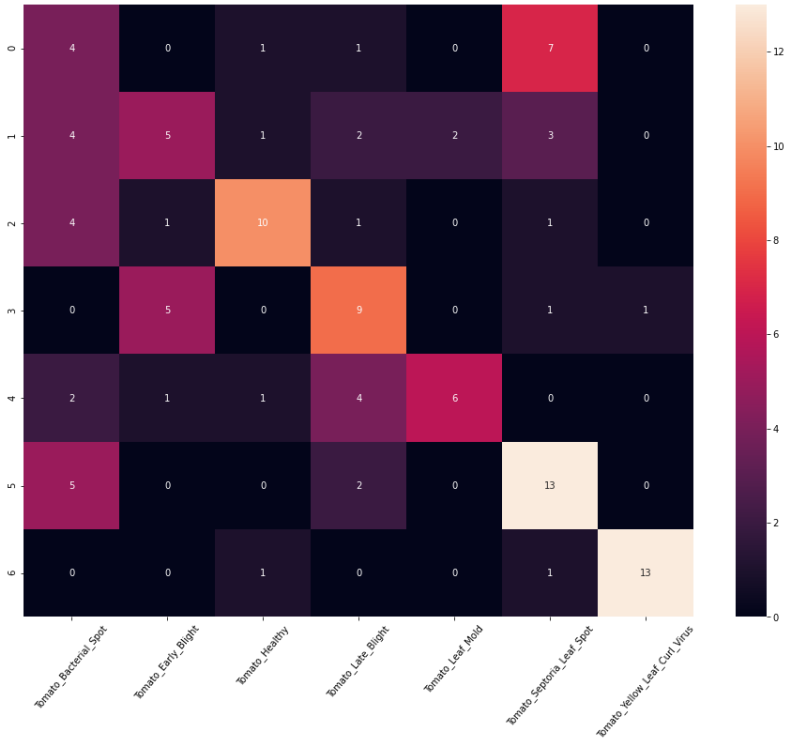
Model Evaluation Matrix

loss : 1.2196178436279297
tp : 45.0
fp : 25.0
tn : 647.0
fn : 67.0
accuracy : 0.5357142686843872
precision : 0.6428571343421936
recall : 0.4017857015132904
auc : 0.8760430812835693
prc : 0.5919050574302673

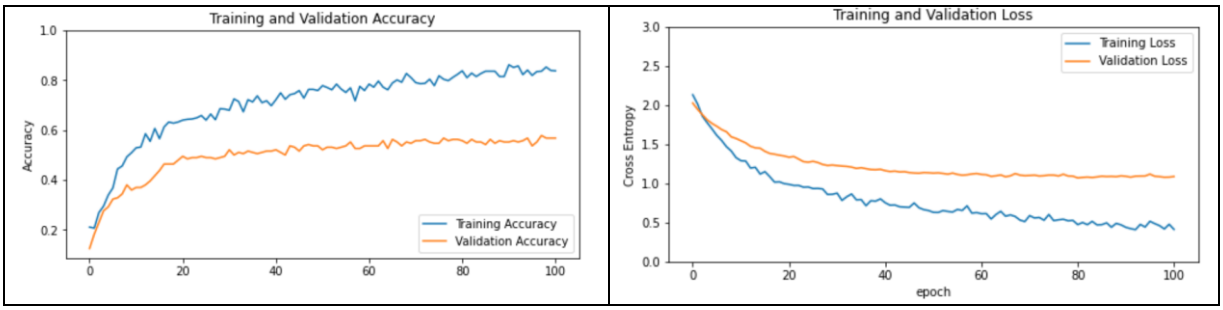
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.21	0.31	0.25	13
Tomato_Early_Blight	0.42	0.29	0.34	17
Tomato_Healthy	0.71	0.59	0.65	17
Tomato_Late_Blight	0.47	0.56	0.51	16
Tomato_Leaf_Mold	0.75	0.43	0.55	14
Tomato_Septoria_Leaf_Spot	0.50	0.65	0.57	20
Tomato_Yellow_Leaf_Curl_Virus	0.93	0.87	0.90	15
accuracy			0.54	112
macro avg	0.57	0.53	0.54	112
weighted avg	0.57	0.54	0.54	112

Fine-tuned model (PlantDoc Original Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Cropped Dataset) – Train and Validation Accuracy & Loss

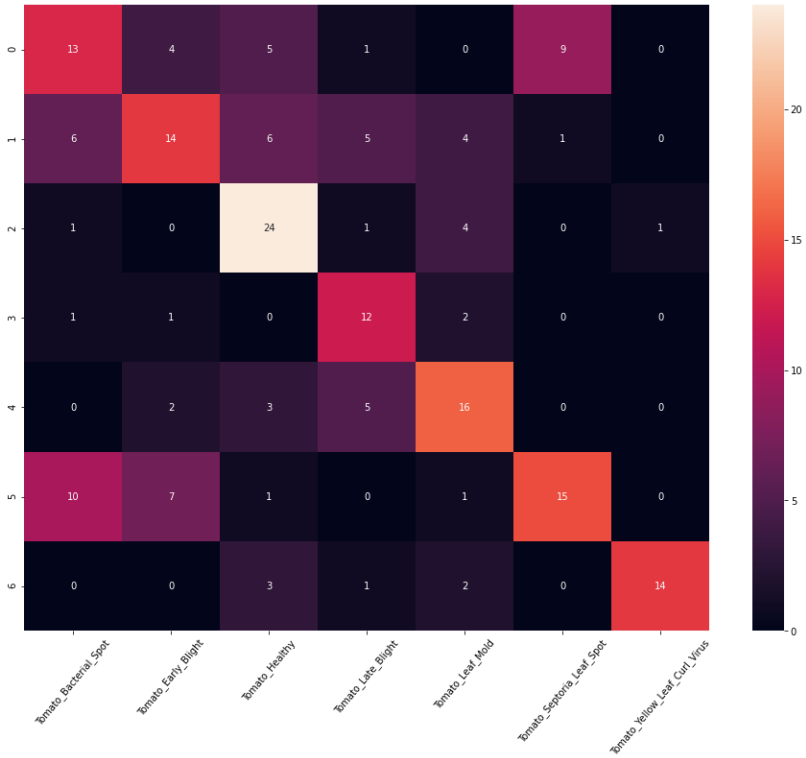


Fine-tuned model (PlantDoc Cropped Dataset) – Classification Report

```
Model Evaluation Matrix

loss :  1.0724668502807617
tp :  87.0
fp :  44.0
tn :  1126.0
fn :  108.0
accuracy :  0.5538461804389954
precision :  0.6641221642494202
recall :  0.446153849363327
auc :  0.9037386775016785
prc :  0.6432443857192993
```

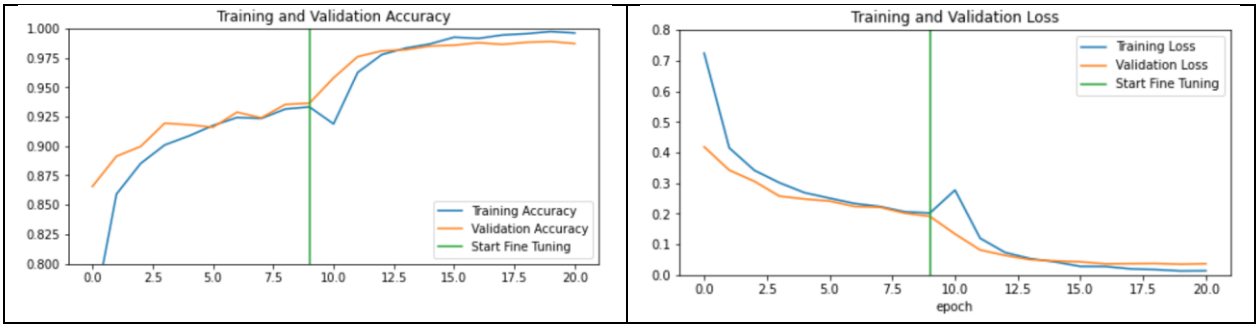
```
Classification Report
                                precision    recall  f1-score   support

          Tomato_Bacterial_Spot       0.42      0.41      0.41        32
            Tomato_Early_Blight       0.50      0.39      0.44        36
                 Tomato_Healthy       0.57      0.77      0.66        31
             Tomato_Late_Blight       0.48      0.75      0.59        16
              Tomato_Leaf_Mold        0.55      0.62      0.58        26
        Tomato_Septoria_Leaf_Spot       0.60      0.44      0.51        34
Tomato_Yellow_Leaf_Curl_Virus       0.93      0.70      0.80        20

                       accuracy                           0.55       195
                      macro avg       0.58      0.58      0.57       195
                   weighted avg       0.57      0.55      0.55       195
```

Fine-tuned model (PlantDoc Cropped Dataset) – Confusion Matrix Heatmap



ResNet50 V2 Models

Fine-tuned model (PlantVillage Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantVillage Dataset) – Classification Report

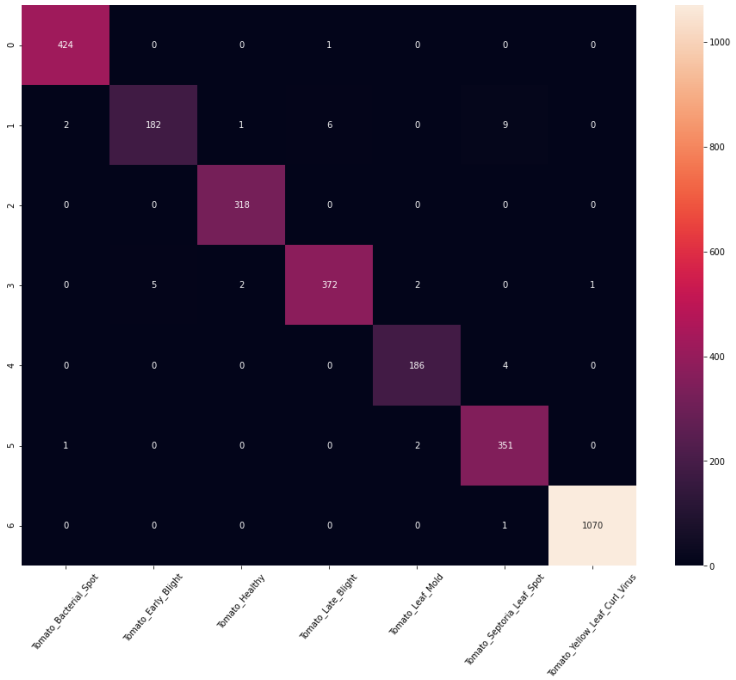
Model Evaluation Matrix

```
loss :  0.036006759852170944
tp :  2901.0
fp :  35.0
tn :  17605.0
fn :  39.0
accuracy :  0.9874149560928345
precision :  0.9880790114402771
recall :  0.9867346882820129
auc :  0.9994919896125793
prc :  0.9982022643089294
```

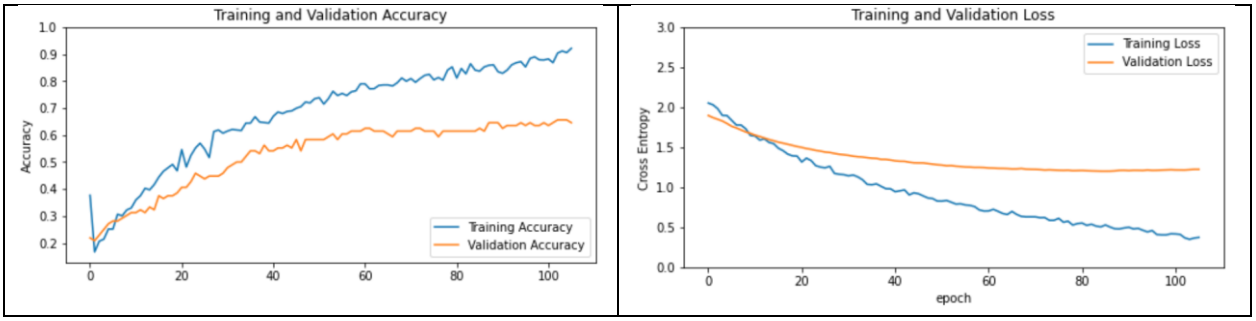
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.99	1.00	1.00	425
Tomato_Early_Blight	0.97	0.91	0.94	200
Tomato_Healthy	0.99	1.00	1.00	318
Tomato_Late_Blight	0.98	0.97	0.98	382
Tomato_Leaf_Mold	0.98	0.98	0.98	190
Tomato_Septoria_Leaf_Spot	0.96	0.99	0.98	354
Tomato_Yellow_Leaf_Curl_Virus	1.00	1.00	1.00	1071
accuracy			0.99	2940
macro avg	0.98	0.98	0.98	2940
weighted avg	0.99	0.99	0.99	2940

Fine-tuned model (PlantVillage Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Original Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Original Dataset) – Classification Report

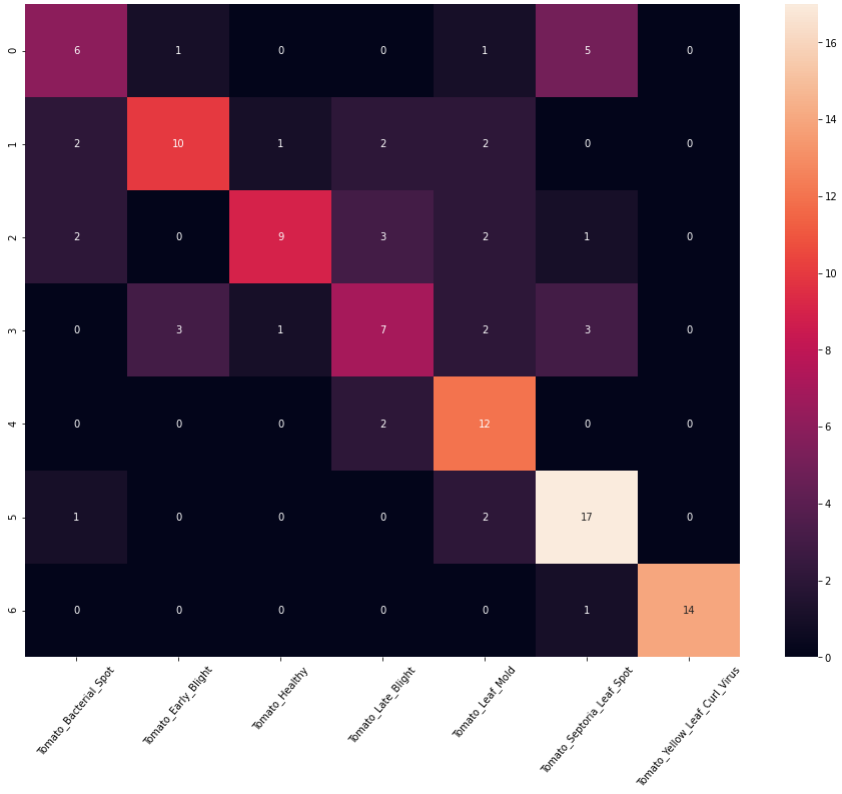
Model Evaluation Matrix

loss : 1.1013076305389404
tp : 59.0
fp : 16.0
tn : 656.0
fn : 53.0
accuracy : 0.6696428656578064
precision : 0.7866666913032532
recall : 0.5267857313156128
auc : 0.897002637386322
prc : 0.7049161195755005

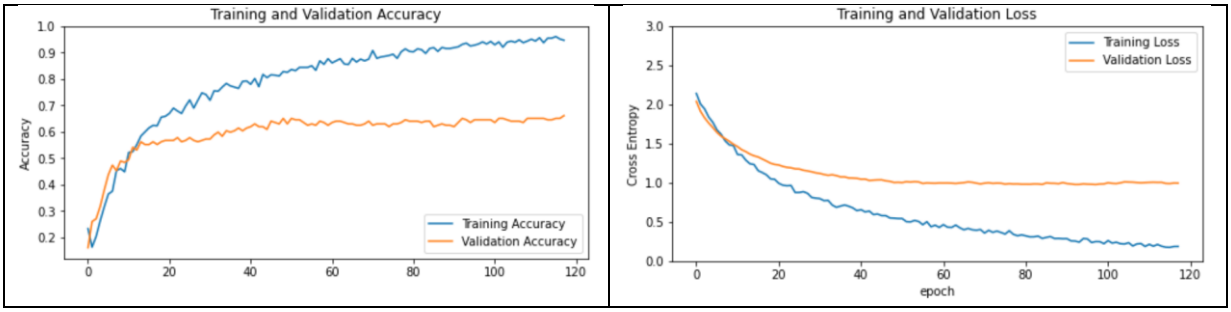
Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.55	0.46	0.50	13
Tomato_Early_Blight	0.71	0.59	0.65	17
Tomato_Healthy	0.82	0.53	0.64	17
Tomato_Late_Blight	0.50	0.44	0.47	16
Tomato_Leaf_Mold	0.57	0.86	0.69	14
Tomato_Septoria_Leaf_Spot	0.63	0.85	0.72	20
Tomato_Yellow_Leaf_Curl_Virus	1.00	0.93	0.97	15
accuracy			0.67	112
macro avg	0.68	0.67	0.66	112
weighted avg	0.69	0.67	0.66	112

Fine-tuned model (PlantDoc Original Dataset) – Confusion Matrix Heatmap



Fine-tuned model (PlantDoc Cropped Dataset) – Train and Validation Accuracy & Loss



Fine-tuned model (PlantDoc Cropped Dataset) – Classification Report

Model Evaluation Matrix

```
loss :  0.9787305593490601
tp :  116.0
fp :  47.0
tn :  1123.0
fn :  79.0
accuracy :  0.6461538672447205
precision :  0.7116564512252808
recall :  0.5948718190193176
auc :  0.9221125841140747
prc :  0.7283641695976257
```

Classification Report

	precision	recall	f1-score	support
Tomato_Bacterial_Spot	0.53	0.50	0.52	32
Tomato_Early_Blight	0.78	0.58	0.67	36
Tomato_Healthy	0.66	0.74	0.70	31
Tomato_Late_Blight	0.43	0.75	0.55	16
Tomato_Leaf_Mold	0.72	0.69	0.71	26
Tomato_Septoria_Leaf_Spot	0.59	0.59	0.59	34
Tomato_Yellow_Leaf_Curl_Virus	1.00	0.80	0.89	20
accuracy			0.65	195
macro avg	0.67	0.67	0.66	195
weighted avg	0.67	0.65	0.65	195

Fine-tuned model (PlantDoc Cropped Dataset) – Confusion Matrix Heatmap