Classification of Plant Disease from Leaf Images using Few-Shot Learning

Sabbir Ahmed

Department of Computer Science and Engineering
Islamic University of Technology (IUT)
July, 2022.

Classification of Plant Disease from Leaf Images using Few-Shot Learning

by

Sabbir Ahmed

Supervisor
Dr. Md. Hasanul Kabir
Professor
Department of Computer Science and Engineering
Islamic University of Technology

MASTER OF SCIENCE
IN
COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering
Islamic University of Technology (IUT)
Board Bazar, Gazipur-1704, Bangladesh.
July, 2022.

CERTIFICATE OF APPROVAL

The thesis titled,"**Classification of Plant Disease from Leaf Images using Few-Shot Learning**" submitted by Sabbir Ahmed, St. No. 171041017 of Academic Year 2017-18 has been found as satisfactory and accepted as partial fulfillment of the requirement for the degree Master of Science in Computer Science and Engineering on July 21, 2022.

Board of Examiners:

Dr. Md. Hasanul Kabir — Chairman
Professor, — (Supervisor)
Department of Computer Science and Engineering,
Islamic University of Technology (IUT), Gazipur.

Dr. Abu Raihan Mostafa Kamal — Member
Professor and Head, — (Ex-Officio)
Department of Computer Science and Engineering,
Islamic University of Technology (IUT), Gazipur.

Dr. Md. Azam Hossain — Member
Assistant Professor,
Department of Computer Science and Engineering,
Islamic University of Technology (IUT), Gazipur.

Dr. Mohammad Abu Yousuf — Member
Professor, — (External)
Institute of Information Technology (IIT),
Jahangirnagar University Savar, Dhaka, Bangladesh.

Declaration of Candidate

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by **Sabbir Ahmed** under the supervision of **Dr. Md. Hasanul Kabir**, Professor, Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither this thesis nor any part of it has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others have been acknowledged in the text and a list of references is given.

Dr. Md. Hasanul Kabir
Professor
Department of Computer Science and Engineering
Islamic University of Technology (IUT)
Date: July 21, 2022.

Sabbir Ahmed
Student No.: 171041017
Date: July 21, 2022.

Dedicated to Allah (SWT), who has bestowed upon me with this knowledge, verily I belong to HIM, and, verily to HIM I will return

Table of Contents

List of Figures

Page

List of Tables

Acknowledgment

First, I express my sincere gratitude to almighty Allah (SWT), for granting me this knowledge and enabling me to complete this thesis for the fulfillment of the degree of Master of Science in Engineering in due time by His grace, and for bestowing upon me good health and energy to carry out this research successfully.

I would like to express my sincere gratitude, appreciation, and gratitude to Dr. Md. Hasanul Kabir, professor in the Department of Computer Science and Engineering, the Islamic University of Technology, for his assistance, close supervision, academic guidance, instructions, encouragement, and valuable criticism throughout the research project.

Special thanks from the deepest core of my heart to Mr. Bakhtair Hasan and Mr. Tasnim Ahmed, without whose support, I would never reach this milestone. I am lucky to have seniors like Mr. Tahmid Rahman Laskar and Mr. Usama Islam who have constantly motivated me to complete this degree. I convey my profound respect and heartiest gratitude to all the faculty members and staff of the Department of Computer Science and Engineering, the Islamic University of Technology for their active cooperation and sincere help in carrying out the research work.

Finally, I would like to direct all my appreciation to my beloved parents and my spouse for their inspiration and endless encouragement.

Abstract

To ensure global food security and the overall profit of stakeholders, the importance of correctly detecting and classifying plant diseases is paramount. In this regard, the emergence of Deep Learning-based architectures has provided remarkable performance in plant disease classification in recent times. However, these solutions often require large-scale datasets and high computation resources to learn generalization and achieve state-of-the-art performance. The unavailability of large public datasets in the domain of plant disease classification and low resource constraints of the end-level devices makes the problem even harder. In this regard, we have proposed two solutions to tackle these existing limitations of the Deep Learning-based systems. At first, to ensure the applicability of these solutions in low-end devices we have proposed a lightweight transfer learning-based approach for detecting diseases from tomato leave images. The proposed pipeline utilizes an effective preprocessing method to enhance the leaf images with illumination correction for improved classification. The system extracts features using a combined model consisting of a pretrained MobileNetV2 architecture and a classifier network for effective prediction. Traditional augmentation approaches are replaced by runtime augmentation to avoid data leakage and address the class imbalance issue. Evaluation on tomato leaf images from the PlantVillage dataset shows that the proposed architecture achieves 99.30% accuracy with a model size of 9.60MB and 4.87M floating-point operations, making it a suitable choice for low-end devices. Afterward, to alleviate the dependency on large publicly available datasets, we proposed a pipeline incorporating the concept of Few-shot learning which can predict leaf diseases with only a few samples. The proposed pipeline produces highly general feature embeddings exploiting nine different state-of-the-art CNN-based feature extractors, which are concatenated and passed to a classifier block consisting of a Bi-LSTM layer for effective prediction. We have shown how the concept of 'Domain Adaptation' can be utilized in this regard to enhance the representation capability of the feature extractors on unseen classes. The model has been evaluated on the tomato leaf images from the PlantVillage dataset where it achieved promising accuracies of 89.06, 92.46, and 94.07 respectively for 5-shot, 10-shot, and 15-shot classification. Furthermore, an accuracy of 98.09 ± 0.77 has been achieved 80-shot classification, which is only 1.2% less than state-of-the-art providing 94.5% reduction in the requirement of training data. Experimental findings show that the proposed pipeline has outperformed all the existing works under single-domain, mixed-domain and cross-domain scenarios.

Chapter 1

Introduction

1.1 Motivation and Scope

The advent of smart systems in the Agricultural domain has been a demand of time to achieve the expected harvest since the overall production is on the decline due to the crops being prone to various diseases [1]. Traditional disease detection approaches require manual inspection of diseased leaves through visual cues or chemical analysis of infected areas, which can be susceptible to low detection efficiency and poor reliability due to human error. Adding to the problem, the lack of professional knowledge of the farmers and the unavailability of agricultural experts who can detect the diseases also hamper the overall harvest production. Negligence in this regard poses a significant threat to food security worldwide while causing great losses for the stakeholders. Early detection and classification of diseases implemented using tools and technologies available to the farmers can go a long way to alleviate all the issues discussed [2].

Several solutions have been proposed using the traditional machine learning approaches for plant disease classification [3]. However, these works are dependent of handcrafted feature extraction techniques, which have failed to generalize on larger datasets. The emergence of deep learning-based methods in the agricultural domain has opened a new door for researchers with outstanding generalization capability removing the dependencies extreme feature engineering [4]. Recently, Convolutional Neural Network (CNN) has become a powerful tool for any classification task as it automatically extracts important features from images without human supervision and has been adapted in different leaf disease classification problems. Moreover, the recent variations of CNN architectures such as AlexNet [5], DenseNets [6], EfficientNets [7], GoogLeNet [8], MobileNets [9,10], NASNets [11], Residual Networks (ResNets) [12], SqueezeNet [13], Visual Geometric Group (VGG) Networks [14], etc, have enabled the machines to understand complex patterns enabling even better performance than humans in many classification problems.

With the introduction of transfer learning where the reuse of a model efficient in solving one problem as the starting point of another problem in a relevant domain has significantly reduced the requirement of vast computational resources [15]. Consequently, the utilization of pretrained AlexNet and GoogLeNet architectures on the publicly available PlantVillage Dataset [16] has been one of the pioneer works of leaf disease classification using transfer learning and paved the way for numerous solutions in the existing literature [17]. These deep neural architectures have been found to be extremely helpful for leaf disease classification tasks for several plants such as, apple [18] , cassava [19], corn [20], cucumber [21], grape [22], maize [23], mango [24], rice [25], etc. However, most of these solutions propose deep and complex networks focusing on increasing the accuracy of detection; posing the requirement of a huge number of training images along with high computational resources. The task is made even more difficult by the lack of significant public datasets in the categorization of plant diseases and the resource limitations of end-level devices.

Real-life applications, such as agriculture, often require small and low latency models tailored explicitly for devices with small memory and low computational power while also having comparable, if not better, accuracy. Most of the systems focusing on lightweight models had to sacrifice accuracy and/or worked with a limited number of diseases/samples. On the other hand, the highly data-driven approaches obstruct the applicability of disease detection for many staple crops due to the unavailability of sufficient affected leaf images. Hence introducing efficient algorithms to accurately detect diseases from limited data can significantly improve the scenario. In this regard, Few-Shot Learning (FSL) is a branch of DL algorithms that have been recently introduced in the literature and aims to work with less amount of data. Despite a handful of FSL-based solutions that have been proposed for leaf disease classification, most of them are yet to achieve satisfactory performance and perform poorly in cross-domain applications.

1.2 Problem Statement

Based on the discussion above, this work aims to propose solutions to reduce the data and resource dependencies of Deep Learning-based solutions for leaf disease classification.

The specific objectives of this research are:

1. Proposing a lightweight model for detecting leaf diseases ensuring high performance along with applicability in low-end devices.

2. Create a robust pipeline with the ability to produce highly general feature embedding from a given sample.

3. Design an effective model to predict the appropriate diseases utilizing the feature representations with very few samples per class.

1.3 Research Challenges

The first research challenge to build any leaf disease classification system is to tackle the scarcity of publicly available datasets. Even if they are available, often the dataset carries a huge amount of class imbalance, which can be a difficult challenge for the model to overcome. Moreover, the samples may contain several challenges like illumination inconsistency, occlusion, deformation, etc. The high inter-class similarity within different leaf disease is hard to distinguish, and sometimes leaves gets highly affected by the disease which makes it almost impossible to correctly identify. To develop lightweight models for leaf disease classification, the system has to ensure that it is maintaining the low-resource constraints of the end-level devices. Satisfying this hardware constraint often makes it really hard to ensure state-of-the-art performance. On the other hand, to develop learning systems aimed to work with limited data, proposing a robust feature extractor with the ability to generate highly meaningful features is a difficult research challenge to consider. The task gets even harder when the model has to satisfy a cross-domain scenario which includes the additional challenge of generalizing features learned from one domain into another unseen domain.

1.4 Research Contributions

The key contributions of this work can be described in two folds:

1. We have proposed a lightweight and fast deep neural architecture for leaf disease classification for real-life applications in low-end devices ensuring state-of-the-art level accuracy. The system exploits a pretrained MobileNetV2 architecture as a feature extractor followed by a classifier network. The effect of poor lighting conditions has been tackled by utilizing Contrast Limited Adaptive Histogram Equalization Technique (CLAHE). Dataset imbalance, overfitting, and data leakage issues have been tackled using the runtime augmentation technique. Experimental results show that the proposed pipeline has achieved an accuracy of 99.30% on the tomato leaf disease subset of the PlantVillage dataset, which is a 2% performance improvement to the baseline MobileNetV2 architecture. Fur-

thermore, the nearest model producing a similar level of performance required $2.4\times$ heavier model size and $2.45\times$ additional FLOPs count requirements.

2. We have proposed a Few-shot learning-based pipeline that can effectively predict leaf diseases with only a few samples. The system can produce highly general feature representation exploiting nine different state-of-the-art CNN-based feature extractors. The concept of feature concatenation has been introduced along with a Bi-LSTM based classifier block to produce high performance with very limited samples. We introduced the idea of Domain Adaptation to improve the representation capability of the feature extractors. Experimental findings show the proposed pipeline has achieved state-of-the-art accuracy for several single-domain, mixed-domain and cross-domain classification tasks.

1.5 Organization

The rest of the dissertation is organized as follows. Chapter 2 discusses the background works related to leaf disease classification. It also identifies the problems persistent in the existing literature. Chapter 3 presents a lightweight deep neural architecture for tomato leaf disease classification. Chapter 4 introduces a few-shot learning approach for classifying leaf diseases with a very limited amount of samples. Chapter 5 concludes highlights the contributions and provides direction for future research scopes.

Chapter 2

Background Study

Earlier approaches in leaf disease classification involved different image-based handcrafted feature extraction techniques that were fed into machine learning-based classifiers. These works mainly focused on only a few diseases with extreme feature engineering and were often limited to constrained environments. To extract features, researchers focused on utilizing different image-level feature extraction techniques like Gray-Level Co-occurrence Matrices (GLCM) [26], Geometric and histogram-based features [27], Gabor Wavelet Transformation [28], Moth-Flame Optimization and Rough Set (MFORS) [29], and similar techniques. To segment the diseased portion of the leaves, several works have extracted the Region of Interest (RoI) using k-means clustering [27], Otsu's method [30], etc. To predict the class labels from the extracted features, Support Vector Machine (SVM) [26, 28], Decision Trees [30], and other classifiers were used. Due to their sensitivity to the surroundings of leaf images, machine learning approaches relied on rigorous preprocessing steps like manual cropping of RoI, color space transformation, resizing, background removal, and image filtering for successful feature extraction. This increased complexity due to preprocessing limited the traditional machine learning approaches to classify a handful of diseases from a small dataset, thus failing to generalize on larger ones.

With the remarkable ability to automatically learn features, Deep Learning (DL) techniques have been widely used in recent times in almost all branches of modern computer science. With no exception, these techniques have become a research hotspot in the agricultural domain as well. The emergence of DL in recognizing plant diseases can ensure global crop protection along with removing manual monitoring saving a significant amount of human labor and effort. In this chapter, we have thoroughly discussed the background literature under two major categories, which are, the lightweight models for leaf disease classification and Few-shot learning algorithms in agriculture.

2.1 Lightweight Models for Leaf Disease Classification

Current research trends on leaf disease classification tend to focus on developing solutions using Deep Neural Architectures, simplifying networks for faster computation targeting embedded systems, real-time disease detection, etc. The introduction of intelligent systems incorporating these solutions could go a long way to reduce crop yield loss, remove tedious manual monitoring tasks, and minimize human efforts.

The performances of a significant portion of the prior works were not comparable as they were mostly done on self-curated small datasets. This issue was alleviated to a great extent when the PlantVillage dataset was introduced containing 54,309 images of 14 different crop species and 26 diseases [16]. A subset of this dataset contains nine tomato leaf diseases and one healthy class which has been utilized by most of the recent deep learning-based works on tomato leaf disease classification. Several works on tomato leaf diseases also focused on segmenting leaves from complex backgrounds [31], real-time localization of diseases [32–34], detection of leaf disease in early-stage [35], visualizing the learned features of different layers of CNN model [36, 37], combining leaf segmentation and classification [38], and so on. These works mostly targeted removing the restrictions of lighting conditions and uniformity of complex backgrounds.

To alleviate the dependency on hand-crafted features along with achieving better classification accuracy with large datasets, recent transfer learning-based approaches to leaf disease classification have investigated the performance of different pretrained models using various hyperparameters. Based on their results, they recommended the use of GoogleNet [36, 39, 40], AlexNet [41], ResNet [42], DenseNet121 [43] in creating tomato leaf disease detection systems due to their superior performance compared to other models. Some of these works have also investigated the effect of different hyperparameter choices like optimizers, batch sizes, the number of epochs, and fine-tuning the model from different depths to see how they impact its performance [39,41]. These models were pretrained on massive datasets, making them the perfect choice for extracting relevant features outperforming shallow machine learning-based models. Although these systems achieved high accuracy going up to 99.39% [39], the models were huge and computationally expensive, often making them infeasible for low-end devices.

Several attempts were made to reduce the computational cost and model size. Durmucş *et al*. [44] utilized SqueezeNet to detect tomato leaf diseases. The base SqueezeNet architecture reduces the computational cost by minimizing the number of 3×3 filters, late downsampling, and deep compression. The authors conducted the

experiments on an Nvidia Jetson Tx1 device targeting real-time disease detection using robots. Tm *et al*. [45] proposed a variation LeNet, one of the earliest and smallest deep-learning architectures. The authors introduced an additional convolutional and pooling layer to the base architecture and increased the number of filters in different layers to extract complex features. However, the accuracies achieved by these two systems were not on par with the performance of the deeper models. Bir *et al*. [46] utilized pretrained EfficientNet-B0 to achieve a comparable accuracy with the state-of-the-art while keeping the model size and computation low. This architecture applies grid search to find coefficients for width, depth, and resolution scaling to reduce the size of the baseline model with a minimal impact on accuracy. However, when classifying the tomato leaves, the authors had to discard a significant number of samples to gain a comparable accuracy. Reduction of dataset size in this manner, even if balanced with augmentation, might result in discarding complex samples restricting the generalization capability of the models. All these issues impose the requirement of lightweight models that can achieve state-of-the-art performance with high generalization capability.

2.2 Few-Shot Learning in Agriculture

Deep learning is progressively being used in agriculture and plant science as artificial intelligence advances. However, the remarkable performance of these approaches requires a huge amount of data, which is always a difficult constraint to satisfy in this domain. The recent advancement of few-shot learning algorithms has shown promising results to solve this limitation. It mimics the human capacity to learn quickly with just a few labeled examples. There have been several branches of few-shot learning introduced in the literature such as methods based on data augmentation, parameter optimization, metric-learning approaches, etc [47]. The metric-learning approach classifies the unseen samples by learning the similarity between different categorizes, the method based on data augmentation expands the number of samples by reusing the originals, and the method based on parameter optimization solves the problem by learning how to optimize parameters.

The training strategy of few-shot learning algorithms is different from the traditional approaches of machine learning and deep learning methods. It is generally regarded as an N-way k-shot problem, where N refers to the number of categories and k refers to the number of samples available for each of them. For a particular classification task with N classes, where each class has only a small amount of samples to train the models, the goal is to learn a function f which will be able to produce highly

general feature representation for effective classification. To assist the task, often an auxiliary dataset is provided which has non-overlapping classes with target ones. The few-shot algorithm will utilize the knowledge from the auxiliary dataset and use the few samples of the target dataset to slightly fine-tune the weights.

The overall training process can be divided into two phases, meta-training and meta-testing. Both of the stages have their own Support set (S) and Query set Q. In the meta-training phase, the model is usually trained with the auxiliary dataset to learn the feature space. On the other hand, during meta-testing, the k-shots are picked from the Support set to fine-tune the model and the performance is evaluated on the Query set.

2.2.1 Metric-learning based methods

The few-shot learning algorithms that are designed based on metric-learning work with the aim of learning a distance function to measure the similarity and dissimilarity between the sample of support set and query set. As illustrated in Figure 2.1, these algorithms learn to compare data samples and classify the query samples based on their similarity to the support samples. In the case of plant leaf disease classification, the model takes the input image to produce an embedding vector which is compared with the embeddings of other classes for prediction. Hence an effective similarity measurement method can remove the overfitting issues caused by the limited samples.

Among numerous variants of metric learning approaches, the Siamese network is one of the earliest one [48], which uses a weight-sharing network model to extract features of two different images, and predicts class labels based on the distance between obtained features. Another similar approach to this network is followed in the MatchingNet architecture [49], which adopts Long short-term memory (LSTM) [50] based approach in the network to improve the quality of extracted features and utilizes a cosine distance-based similarity function. There has been several architectures proposed in the literature with the idea of metric-learning based tasks such as Prototype network [51], RelationNet [52], RelationNetV2 [53], etc.

In the domain of agriculture, Argüeso *et al*. [54] proposed one of the most pioneering works introducing the concept of Few-shot learning algorithms in plant disease classification. The authors utilized the PlantVillage dataset and divided it into two subsections. At first, 32 classes were used to train an InceptionV3 architecture to learn general leaf characteristics. Then this fine-tuned architecture was used to build a Siamese Network with Triplet loss, which was further trained with the few samples of the target classes. Experimental findings showed how the proposed pipeline achieved high performance with almost 90% reduction in training data and outperformed the

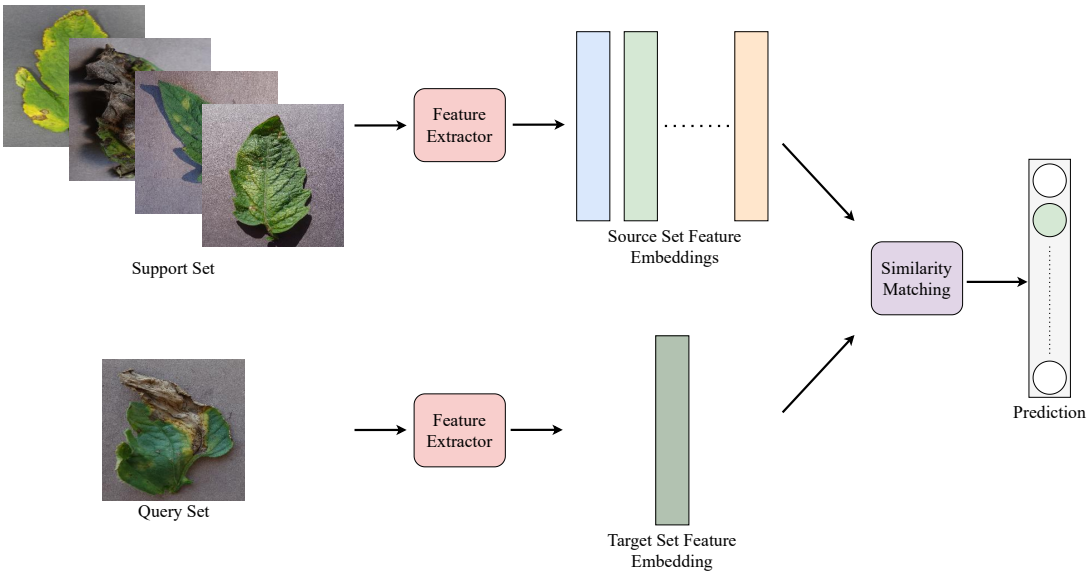


Figure 2.1: Metric-Learning based methods

classical approaches for small training sets.

Wang *et al*. [55] proposed an FSL-based method for plant leaf classification based on Siamese Networks [48] to achieve high accuracy with a limited number of samples. The network extracts features by passing two randomly picked images in the parallel branches of the baseline CNN architecture with weight sharing. Then the metric space is learned utilizing a loss function that focuses on minimizing the distance of images within similar classes and maximizing the distance of samples of different classes. Finally, the leaves are classified using the K-Nearest Neighbour classifier exploiting the learned metric space. Experiments were conducted on the open Flavia dataset [56], Swedish dataset [57] and Leafsnap dataset [58]. However, the datasets used by the authors are fairly simple with white/gray background and work only with plant classification. The inter-class similarity is very less, so the classification task becomes simpler. This method can be validated in plant leaf disease detection, where an additional challenge will be, that the model has to learn the disease patterns along with learning the structure of the leaf.

Egusquiza *et al*. [59] demonstrated a FSL-based approach where the latent space representation was learned using a model based on Siamese networks and metric learning. The work was evaluated on a dataset containing samples of 5 crops and 17 diseases, collected from real-life field images under challenging conditions. The authors showed, how a FSL-based method can outperform traditional CNN-based works trained with even less than 200 samples per class. On the other hand, to address the problem of leaf disease classification with limited data, Jadon *et al*. [60] proposed a metric-based Few-shot learning architecture named 'SSM-Net' consisting of stacked

Siamese [48] and Matching Network [49] components. Experiments showed that the proposed pipeline achieved 92.7% on the mini-leaves dataset [61] and 94.3% on their self-curated sugarcane dataset in a 5-way 5-shot setup, which was found to be significantly better than using a pretrained CNN-based pipeline with VGG16 backbone.

Li *et al*. [62] introduced the first task-driven meta-learning approach in the agricultural domain and provided baselines for different types of Few-shot crop and pest classification tasks. The authors curated a comprehensive dataset exploiting different publicly available resources, containing samples of both pests and plants to evaluate the few-shot algorithms under single-domain, mixed-domain, and cross-domain scenarios. Tassis *et al*. [63] evaluated the performance of different FSL algorithms such as Prototypical Networks [51], Triplet Networks [64], etc, in classification and severity estimation task. A dataset of biotic stress in coffee leaves was utilized where the authors achieved 96.03% and 96.72% accuracies respectively on the biotic stress classification in the leaf dataset and symptoms dataset. More importantly, in the severity estimation task, an accuracy of 93.25% was achieved which was 6.74% higher than the baseline. This signifies the promise in the use of FSL approaches in plant biotic stress recognition. Applying these architectures with the context of domain shift will be an interesting arena to explore.

In another work, Li *et al*. [65] proposed a method for cotton pest recognition using metric-based FSL, which generates feature vectors utilizing a CNN-based backbone with Triplet Loss. To verify the effectiveness and feasibility of the model, experiments were conducted on two datasets where high performance was achieved reflecting the generalization ability of the model. Furthermore, the model was transplanted into an embedded terminal and found to run smoothly with a FPGA-based circuit module and ARM-based control program, achieving a processing speed of 2 frames/second.

2.2.2 Data Augmentation based approaches

Since the main challenge of FSL tasks is the lack of labeled samples, the simplest way to tackle this issue is to expand the number of samples (Figure 2.2). If the number of samples can be expanded close to the amount required by traditional approaches, the overfitting issues faced by few-shot learning can be solved to some extent.

Hu *et al*. [66] presented a low-shot learning approach for tea leaf disease identification using Support Vector Machine (SVM) and Deep learning networks. At first, color and texture features were extracted to segment the disease spots using a SVM-based method. An improved Conditional Deep Convolutional Generative Adversarial Network (C-DCGAN) [67] is utilized to generate augmented samples using the extracted regions and used to train a deep VGG16 model [68] for tea leaf disease prediction with

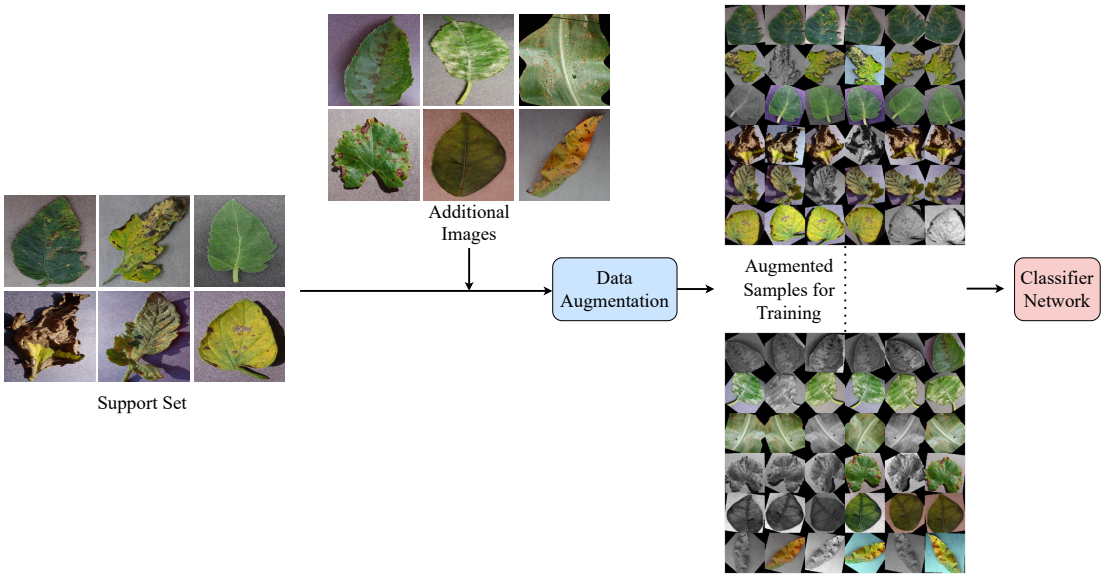


Figure 2.2: Data Augmentation based approaches

an accuracy reaching up to 90% on a self-curated dataset containing three tea leaf diseases.

Nesteruk *et al*. [69] proposed an image augmentation-based method for solving the FSL task for multitask application in agricultural use-cases. Given only a few examples, the proposed augmentation framework can enlarge the number of training samples while providing enough information for tasks like object detection, segmentation, classification, contouring, and denoising. The authors used the DeepLabV3 model for the semantic segmentation task, where around 9% performance improvement was achieved compared to applying the basic augmentation techniques.

2.2.3 Parameter optimization based methods

Traditional deep learning optimizes the network model's parameters using a vast number of samples as shown in Figure 2.3. The parameters are fitted to the best value by the network's gradient backpropagation. However, since the number of samples is small in FSL tasks, following the same procedure often results in overfitting. In this regard, the parameter optimization-based methods entail learning how to tune parameters using an optimizer.

Finn *et al*. [70] proposed on of the pioneering optimization-based method named 'Model-agnostic meta-learning (MAML)', which solves the few-shot task by learning an initialization parameter θ. Once the proper initialization can be learned, the new tasks can be solved with only a few gradient updates. The main benefit of this method is that it is designed to be independent of the meta-learner algorithm used. Therefore, a lot of machine learning algorithms that demand quick adaptation employ the MAML

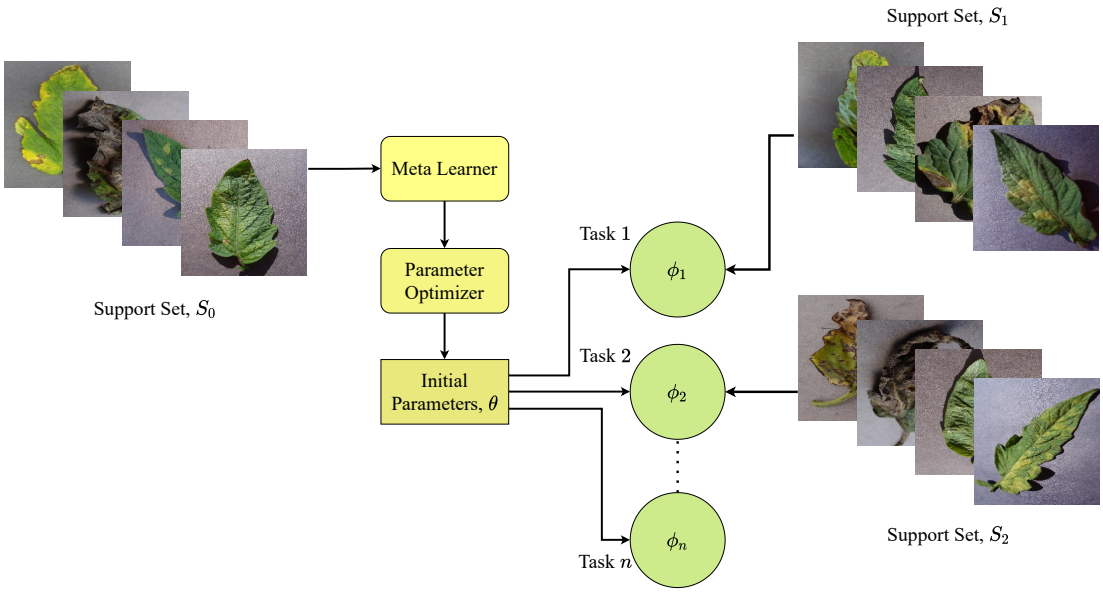


Figure 2.3: Parameter optimization based methods

technique. There has been several improvements over this approach such as 'Task-agnostic meta-learning (TAML)' [71], Reptile [72], etc.

Wang *et al*. [73] exploited the Model-agnostinc meta-learning approach [70] name 'IMAL' for plant leaf disease classification. The authors utilized the 'soft-center loss' function [74] to enhance the capability of the model to distinguish features information of different classes and Parametric Rectified Linear Unit (PReLU) [75] activation function was used to enhance the learning capability of the model with negligible additional impact on the overall computational cost along with reducing chances of overfitting. Experimental results showed that the proposed method achieved a higher result than some other FSL approaches such as fine-tuning approach, Siamese Network with contrastive loss, triplet loss, baseline model-agnostic model, etc. The proposed pipeline achieved 63.8%, 91.35%, and 96.0% accuracy respectively for 1-shot, 15-shot, and 80-shot classification. However, the IMAL approach requires to be trained on a huge number of tasks and consumes high computational resources. Future attempts can be made to further optimized to increase speed and reduce the computational overhead.

2.2.4 Pretrained Feature Extractors for Few-shot classification tasks

Most of the research works on few-shot learning have developed advanced meta-learning-based methods [76–78]. However, recent works show that transfer learning-based approaches using pretrained architectures can be utilized to produce highly general feature representations for FSL tasks.

Chen *et al*. [79] proposed one of the first research works to point out that a sim-

ple transfer-based approach consisting of a pretrained deep CNN and a simple linear classifier can outperform traditional few-shot learners. Authors in [80] argued that the best way to solve the FSL task is to use a high-quality feature extractor rather than focusing on meta-learning based methods. They proposed an approach which trains a huge model on a massive dataset to learn the feature space. In another work, the authors showed how a simple transfer-based approach can outperform advanced meta-learners [81]. Reference [82] utilized a set of feature extractors each trained on a different type of dataset to introduce diversity in the extracted features. On the other hand, [83] proposed an idea of ensembling to solve few-shot classification tasks. The authors simultaneously trained a series of deep CNN networks during the meta-training phase and added a penalty term to encourage diversity in the learned representations.

Nuthalapati *et al*. [84] proposed a method utilizing pretrained feature extractors to automatically classify plant, pests, and their diseases using FSL. The pipeline generates feature embeddings using a feature extractor exploiting a ResNet18 based backbone [85] pretrained on the ImageNet dataset [86]. Then it is passed to a Transformer block [87] to enrich the features embeddings of all samples of the support set in a given task even more. The similarity between the transformed embeddings (from the training phase) and embedding of the image to be classified was calculated using Mahalanobis distance [88]. The effectiveness of the model was shown using mixed-domain and cross-domain experiments using 'Plant and Pest' [62] and Tomato leaf subset of Plantvillage dataset [16]; where the proposed model achieved up to 14% and 24% performance gain compared to the earlier benchmarks. Moreover, to validate the performance of the model in real-world images, the authors curated a dataset named 'Plants in Wild' and provided a baseline.

Being inspired by the above-mentioned ideas, we proposed a transfer learning based solution to solve the FSL task utilizing a library of feature extractors pretrained on a large-scale dataset and combined the extracted features with a Bi-LSTM based classifier network to provide high performance with very limited data.

Chapter 3

Leaf Disease Classification with Lightweight Model

The amazing ability of Deep Neural Networks to automatically learn features have resulting into their wide usage in leaf disease classification tasks. However, the most of these architectures comes with huge computational requirements, which is not feasible to be deployed in agricultural low-end devices. With the goal to tackle this limitation, we have proposed a lightweight deep neural architecture which ensures state-of-the-art level performance along with applicability in low-end devices. In this chapter, we have discussed the proposed lightweight pipeline for lead disease identification. At first, we have discussed the overview of the entire architecture. Then we justified our intuition behind choosing each of the individual components of the proposed pipeline.

3.1 Overview

The proposed architecture takes tomato leaf images as input and outputs the class labels. At first, the input image is passed through a preprocessing step where it is enhanced using Adaptive Histogram Equalization. Then, the enhanced image is fed to a transfer learning block, where we utilize a pretrained deep CNN model for efficient feature extraction. To determine a suitable feature extractor, we experimented

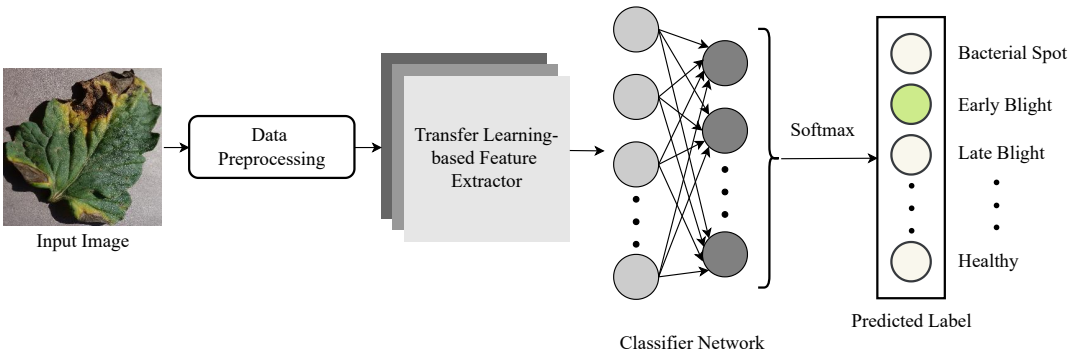


Figure 3.1: Overview of the tomato leaf disease classification architecture

with nine different pretrained architectures which are DenseNet121, DenseNet201, EfficientNet-B0, VGG19, MobileNet, MobileNetV2, NASNet-Mobile, ResNet50, and ResNet152V2. Based on the results, we have chosen MobileNetV2 due to its smaller size and faster inference while maintaining comparable accuracy. Then the features extracted by the pretrained model are fed through a shallow densely connected classifier network to get the Softmax probabilities for every class using which we predict the final label. The general pipeline of the proposed approach is depicted in Figure 3.1.

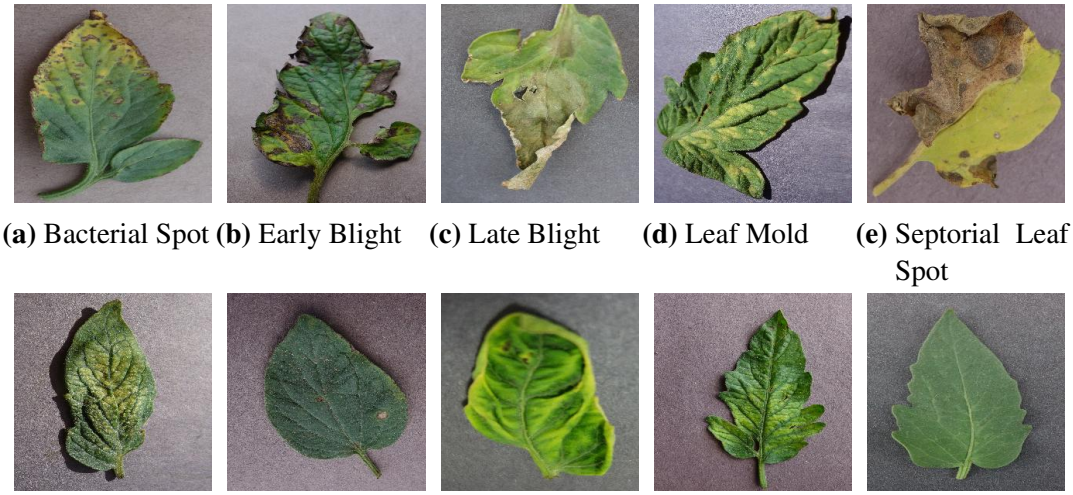
3.2 Materials and Methods

In this section, we have discussed each of the components of the proposed pipeline in details. We started the discussion by describing the dataset, followed by the preprocessing and augmentation techniques that have been incorporated. Then we provided the detailed analysis on the architectural detail of the CNN-based feature extractor and classifier network. Afterwards, we discussed the different policies adopted throughout the experiment along with the evaluation metrics.

3.2.1 Dataset

As of today, the PlantVillage Dataset is the largest open-access repository of expertly curated leaf images for disease diagnosis. The dataset comprises 54,309 images of healthy and infected leaves belonging to 14 crops, labeled by plant pathology experts. Among them, 18,160 images are of tomato leaves, divided into one healthy and nine disease classes. This dataset offers a wide variety of diseases and contains samples of leaves being infected by various diseases to different extents. One sample image from each class can be seen in Figure 3.2.

From the distribution of the number of samples in different classes shown in Table 3.1, it is evident that the dataset contains imbalance as different classes have a significantly varying number of samples. The maximum number of samples is 5357, belonging to Yellow Leaf Curl Virus disease, whereas the number of samples corresponding to Mosaic Virus disease is as low as 373. Few problems arise because of this class imbalance. First, the model does not get a good look at the images of classes with a lower number of samples, leading to less generalization [89]. Moreover, the overall accuracy might still be high even if the model is ignoring these small-sized classes, as they do not contribute much to the overall accuracy [90]. Different techniques involving undersampling and oversampling can be employed to tackle this issue, ensuring that the model is equally capable of identifying all diseases.



(a) Bacterial Spot **(b)** Early Blight **(c)** Late Blight **(d)** Leaf Mold **(e)** Septorial Leaf Spot

(f) Two-spotted Spider Mites **(g)** Target Spot **(h)** Yellow Leaf Curl Virus **(i)** Tomato Mosaic Virus **(j)** Healthy

Figure 3.2: Sample tomato leaf images of the 10 classes from the PlantVillage dataset

3.2.2 Data Preprocessing

Disease spots often have close intensity values with the surroundings due to the poor lighting condition of the images provided in the dataset. Moreover, in real-world applications, images captured by the end-users might not always be adequately illuminated, and this might fail to provide the model with enough details to identify the disease, and hence affect the classification result [91]. Contrast enhancement techniques like histogram equalization can be applied to enhance the details and correct the illumination problem. Generally, histogram-based approaches work globally throughout the image. However, the intensity distribution of the leaf regions can be different from that of the background. So, the same transformation function cannot be applied to the entire image. To tackle the illumination problem addressing the uneven distribution of intensity, we opted for Contrast Limited Adaptive Histogram Equalization [92].

Furthermore, as mentioned earlier, there exists a class imbalance in the original dataset. This issue has been tackled in various ways in the existing literature. The most common way of dealing with this has been to undersample and/or oversample certain classes [40,42,43,46]. Although it makes the dataset balanced to some extent, it has its own drawbacks. Undersampling may drop some of the challenging images for certain classes that can contain important information for the model to learn, which eventually hinders the generalizing capability of the model. Oversampling utilizes different data augmentation techniques to produce multiple copies of the original images, each having slight variations. But if we perform augmentation before splitting the dataset into train, validation, and test sets, it might inject slight variations of the training set into the test set. As the model learns to classify one variation of the image while training, it is

Table 3.1: Distribution of samples in the dataset

Class Label	Sample Count
Bacterial Spot	2127
Early Blight	1000
Late Blight	1909
Leaf Mold	952
Septoria Leaf Spot	1771
Two-spotted Spider Mites	1676
Target Spot	1404
Yellow Leaf Curl Virus	5357
Tomato Mosaic Virus	373
Healthy	1591
Total	18160

(a) Original Image **(b)** Enhanced Image

Figure 3.3: Illumination correction using Contrast Limited Adaptive Histogram Equalization.

highly likely to correctly classify the other variations in the test set, overestimating the accuracy of the system. This problem is known as data leakage [93]. As each choice has its pros and cons, we decided to perform data augmentation during runtime.

Contrast Limited Adaptive Histogram Equalization (CLAHE)

CLAHE increases the contrast between diseased spots and the leaf by dividing the image into multiple small regions and applying a transformation function that is proportional to the cumulative distribution function. This function is calculated based on the histogram of the intensity distribution of the pixels inside each region. CLAHE also limits the amplification of the noise, which is prevalent in low light images, near regions with constant intensity by clipping the histogram value beyond a threshold. Figure 3.3 shows the sample output after applying CLAHE on an original image.

Before applying CLAHE, the leaf image was converted from RGB color space to Hunter Lab color space. Here, L denotes the channel with the intensity value of the

image, a and b denotes the color components. CLAHE was applied on the L channel. The image was then divided into $P \times Q$ regions, where P denotes the number of contextual regions in the x-axis, and Q denotes the number of contextual regions in the y-axis. Our empirical results demonstrated that a value of 7 for both P and Q provided the best results.

Suppose each of the divided regions contains M pixels having intensity value ranging from 0 to $(N-1)$. That means, there were N discrete intensity levels in the leaf image. Then for each region, the histogram $H_{i,j}$ was calculated, where $0 \leq i < P$ and $0 \leq j < Q$. Each of the N histogram bins $H_{i,j}(k)$ contained the number of pixels in the region (i, j) with intensity k. Here, $0 \leq k \leq N-1$. Then each histogram was clipped based on a threshold β, which was set to be 3 upon experimentation. To do that, the total number of excess pixels per histogram bin E was calculated.

$$E = \sum_{k=0}^{N-1} \begin{cases} (H_{i,j}(k) - \beta), & \text{if } H_{i,j}(k) > \beta \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

Then the average pixel increment per bin, A was calculated:

$$A = \frac{E}{N} \tag{3.2}$$

Then for each histogram bin, pixels were redistributed.

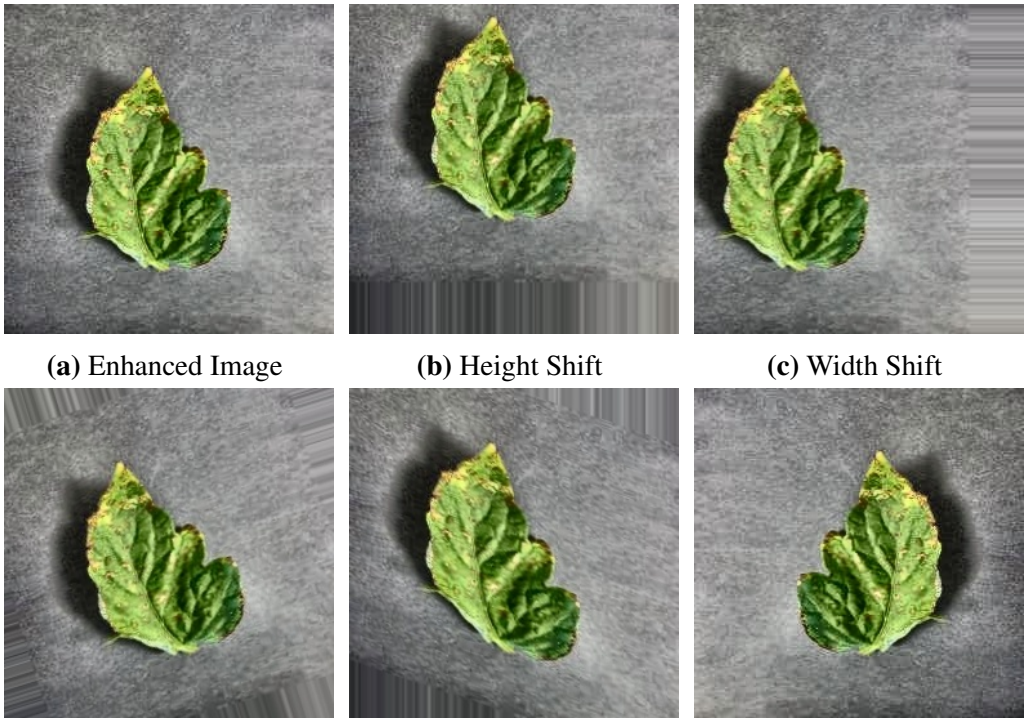
$$H_{i,j}(k) = \begin{cases} \beta, & \text{if } H_{i,j}(k) > \beta \\ \beta, & \text{if } H_{i,j}(k) + A > \beta \\ H_{i,j}(k) + A, & \text{otherwise} \end{cases} \tag{3.3}$$

At the same time, each increment was subtracted from E to keep track of the total number of remaining excess pixels. After the initial distribution, if there were any other remaining excess pixels, they were distributed equally to all the bins.

From the clipped histogram, the Cumulative Distribution Function $C_{i,j}$ was calculated.

$$C_{i,j}(k) = \sum_{j=0}^{k} \frac{n_j}{M} \tag{3.4}$$

Here, n_j is the number of pixels with intensity value j, M is the number of pixels in the region (i, j), and $0 \leq k < N$. $C_{i,j}$ is used to calculated the mapping function $F(k)$, where $0 \leq k < N$ is calculated.



(a) Enhanced Image **(b)** Height Shift **(c)** Width Shift

(d) Rotation **(e)** Shearing **(f)** Horizontal Flip

Figure 3.4: Data augmentations. A combination of these augmentations were applied randomly during run-time.

The mapping function was used to calculate the desired intensity by performing Bilinear Interpolation of the four nearby regions to reduce the blocking effect. The output intensity values were scaled within the range $[0, N-1]$. Then using $F(k)$, the intensity of the L channel was mapped to the desired intensity value. Finally, the image was converted from Hunter Lab color space to RGB color space.

To maintain consistency, we have preprocessed all the tomato leaf images of the dataset using CLAHE before feeding them to the model.

Data Augmentation

To reflect real-life scenarios, we have picked height and width shifting, clockwise and counterclockwise rotation, shearing, and horizontal flipping to augment the leaf images during runtime.

Height and Width Shifting is performed by translating each pixel of the image respectively in the horizontal and vertical direction by a constant factor. Height shift is performed using:

$$y_{\text{new}} = y_{\text{old}} + \alpha n_y \tag{3.5}$$

where,

y_{new} = The new y-coordinate of the shifted pixel

y_{old} = The old y-coordinate of the shifted pixel

α = Constant factor

n_y = Number of pixels in the y-axis of the image

In our case, the constant factor, α was chosen randomly within the range $[0, 0.2]$. While shifting, the pixels going outside the boundary ($y_{\text{new}} \geq n_y$) are discarded, and the empty regions ($y_{\text{old}} < \alpha n_y$) are filled with the RGB values of the nearest pixels. During height shift, the values of the x-coordinate remain unchanged.

Width shift is performed using:

$$x_{\text{new}} = x_{\text{old}} + \beta n_x \tag{3.6}$$

where,

x_{new} = The new x-coordinate of the shifted pixel

x_{old} = The old x-coordinate of the shifted pixel

β = Constant factor

n_x = Number of pixels in the x-axis of the image

In our case, the constant factor, β was chosen randomly within the range $[0, 0.2]$. While shifting, the pixels going outside the boundary ($x_{\text{new}} \geq n_x$) are discarded, and the empty regions ($x_{\text{old}} < \beta n_x$) are filled with the RGB values of the nearest pixels.

Figure 3.4b and 3.4c shows the effect of performing height and width shift, respectively.

Rotation is performed with respect to the centre pixel of the image. It is performed using:

$$x_{\text{new}} = \cos(\theta) \times (x_{\text{old}} - x_c) - \sin(\theta) \times (y_{\text{old}} - y_c) + x_c \tag{3.7}$$

$$y_{\text{new}} = \sin(\theta) \times (x_{\text{old}} - x_c) - \cos(\theta) \times (y_{\text{old}} - y_c) + y_c \tag{3.8}$$

where,

x_{new} = The new x-coordinate of the rotated pixel

y_{new} = The new y-coordinate of the rotated pixel

x_{old} = The old x-coordinate of the rotated pixel

y_{old} = The old y-coordinate of the rotated pixel

x_c = The x-coordinate of the center pixel

y_c = The y-coordinate of the center pixel

θ = The rotation angle

For our case, the rotation angle, θ was chosen randomly within the range [-20, 20]

(a) **(b)** **(c)**

Figure 3.5: Sample augmentations performed on the images during training, validation, and testing phase

degrees. Figure 3.4d shows the effect of performing rotation.

Shearing is performed by moving each pixel towards a fixed direction by an amount proportional to the pixel's distance from the bottom-most pixels of the image based on a shearing factor.

$$\begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \end{bmatrix} = \begin{bmatrix} 1 & m \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \end{bmatrix} \tag{3.9}$$

where,

x_{new} = The new x-coordinate of the sheared pixel

y_{new} = The new y-coordinate of the sheared pixel

x_{old} = The old x-coordinate of the sheared pixel

y_{old} = The old y-coordinate of the sheared pixel

m = Shearing factor

We randomly picked the shearing factor, m within the range $[0, 0.2]$. Figure 3.4e shows the effect of performing shearing.

Flipping an image horizontally requires mirroring the pixels with respect to the centerline parallel to the x-axis.

$$x_{\text{new}} = x_{\text{old}} \times (-1) + n_x \tag{3.10}$$

where,

x_{new} = The new x-coordinate of the mirrored pixel

x_{old} = The old x-coordinate of the mirrored pixel

n_x = Number of pixels in the x-axis of the image

Figure 3.4f shows the effect of performing horizontal flipping.

Multiple random augmentations are applied to the same image to ensure that the

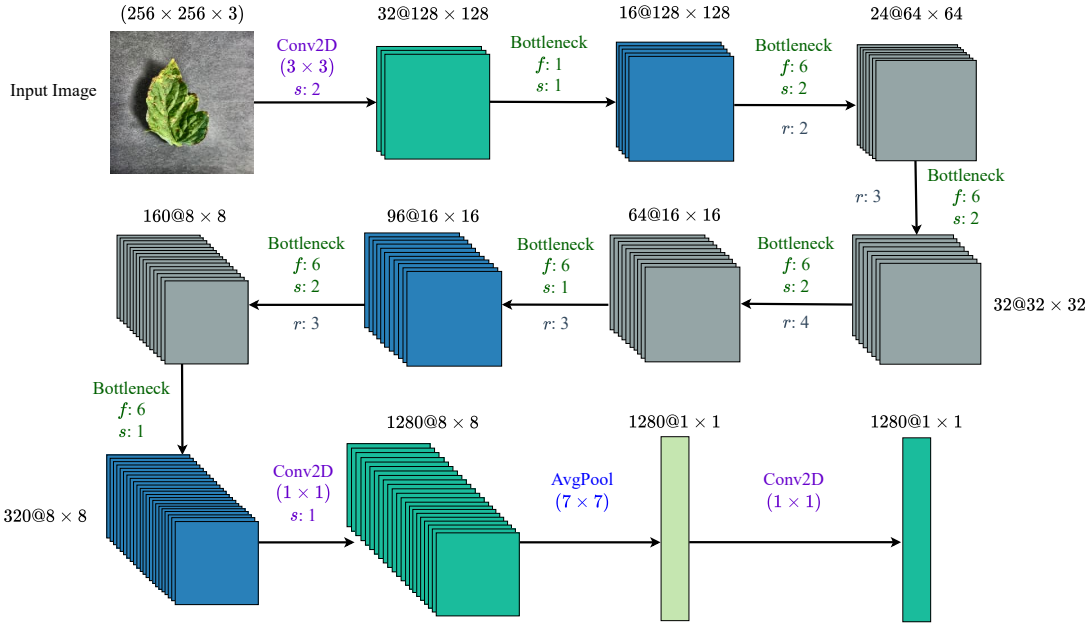


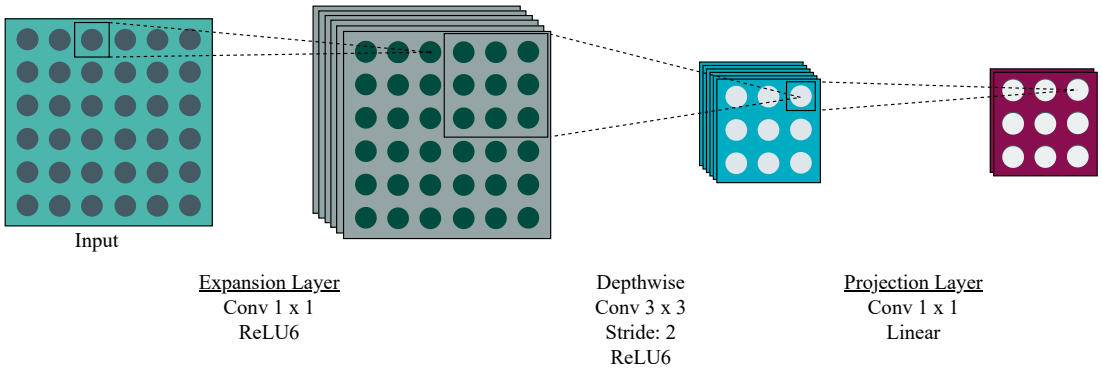
Figure 3.6: MobileNetV2 architecture adopted from [10] and modified for extracting features from $256 \times 256 \times 3$ tomato leaf images. Each box represents the feature maps (not to scale) after going through different layers. Here, f denotes the expansion factor of each Bottleneck Layer. The first layer of each sequence has a stride value of s, and the remaining use stride 1. r denotes the number of times a layer is repeated to produce the next feature map.

model sees a new variation on every epoch and thus learns to recognize a variety of images. Figure 3.5 shows the effect of combining different augmentations that are used during the training, validation, and testing phase.

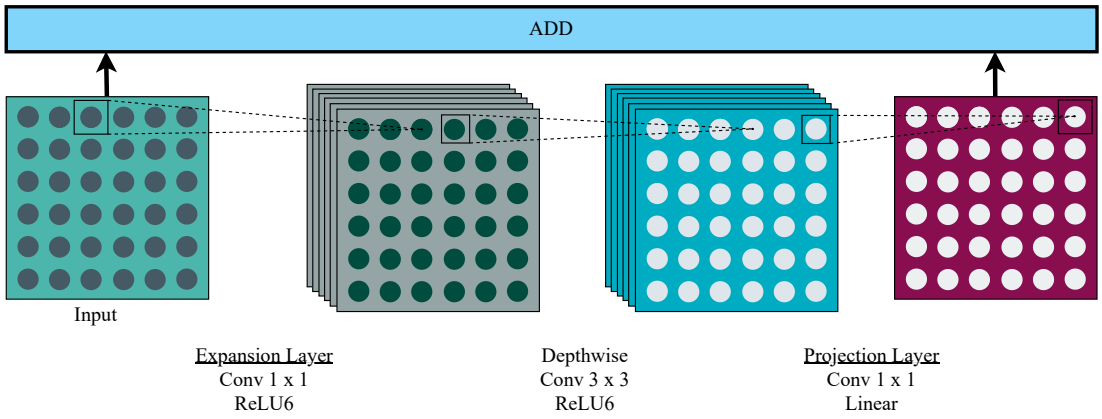
Unlike traditional approaches in the existing literature of tomato leaf disease classification, we decided not to use data augmentations to increase the number of samples before training. Instead, these augmentations were performed randomly on different images during runtime in different splits, ensuring that the model sees different variations of the same image separately in different epochs. This reduces the possibility of overfitting, as it cannot see the same image in every epoch. On the other hand, this ensures that the different variations of the same image do not appear in both training and test set, thus eliminating the data leaking problem persistent in the existing literature.

3.2.3 Transfer Learning-based Feature Extractor

Earlier machine learning approaches assumed that the training and test data must be in the same feature space. However, recent advances in deep learning approaches have facilitated the use of an architecture trained to extract features on the training data of one domain to be used as a feature extractor for another domain. As the feature extractors in deep learning-based tasks became more and more generalized, this method of knowledge transfer, also known as Transfer Learning, has significantly im-



(a) Depthwise Separable Convolution with Stride-2 Block



(b) Inverted Residual Connection with Stride-1 Block

Figure 3.7: Bottleneck Residual Block. Here, each block represents the feature map output by different layers.

proved the performance of learning, reducing a considerable amount of computational complexity. In this connection, the MobileNetV2 architecture has enabled real-time applications across multiple tasks and benchmarks using low computational resources. As shown in Figure 3.6, MobileNetV2 consists of a regular 3×3 convolution with 32 filters, followed by 17 Bottleneck Residual Blocks, a Pointwise convolution layer, a global average pooling layer, and a classification layer. The classification layer usually corresponds to the number of classes of the original dataset. For our system, the classification layer was replaced with a classifier network to classify tomato diseases.

At the heart of the MobileNetV2 architecture resides Bottleneck Residual Block containing three convolutional layers (Figure 3.7a). The Expansion Layer increases the number of channels in the input data by performing Pointwise convolution based on an expansion factor. The feature map output by this layer is then fed to a 3×3 Depthwise Convolution layer which works as a filter by applying convolution per channel. The Projection Layer takes these filtered values to generate salient features. Besides, this layer projects the higher dimensional data into a much lower number

of dimensions, reducing the number of channels. The Depthwise Convolution layer combined with the Pointwise Convolution performs Depthwise Separable Convolution, reducing the computation by a factor of $O(k^2)$ compared to regular convolutions. Here, k is the size of the Depthwise convolution kernel. Like most modern architectures, each of the three convolution layers is followed by batch normalization to stabilize the learning process. The activation function used by these layers is ReLU6. It bounds the activation within $[0, 6]$, making it more robust than the well-known ReLU function in fixed-point arithmetic. However, the Projection Layer does not contain any activation function due to the low dimensionality of the data produced by this layer. The non-linearity of the ReLU activation function can destroy valuable features. In addition, to reduce the effect of diminishing gradients, inverted residual connections are introduced through the network, which connects the bottleneck blocks with the same number of channels (Figure 3.7b).

In this work, to compare the performance of MobileNetV2, other transfer-learning architectures that are popular in leaf disease detection: DenseNet121, DenseNet201, EfficientNet-B0, MobileNet, NASNet-Mobile, ResNet50, ResNet152V2, and VGG19 were used.

3.2.4 Classifier Network

Instead of directly using the extracted features from pretrained models for final prediction, we employed a combination of dense, dropout, and batch normalization blocks to fine-tune the extracted traits further. As shown in Figure 3.8, dense blocks were added before the final output layer. The pretrained MobileNetV2 architecture we used was trained on a large and generalized dataset, making it perfect for feature extraction. The features extracted from the leaf images by MobileNetV2 architecture are then fed into the dense blocks trained from scratch to extract further the relevant features required to classify the diseases. Upon experimentation with different numbers of layers consisting of a varying number of nodes, two dense blocks with 128 and 64 nodes helped us achieve global minima in terms of loss.

A Batch Normalization [94] block was added between the output of the MobileNetV2 and the first densely connected block and one between the second densely connected block and the output layer. A batch normalization block is used to standardize the inputs for the final layer for each mini-batch and stabilize the whole learning process, reducing the epochs needed to train the network. Rectified Linear Unit (ReLU) [95] was used as the activation function of the two densely connected blocks. This activation function makes the models easier to optimize and more generalizable. A dropout layer [96] in-between these two dense blocks work as a regularizer, ensuring

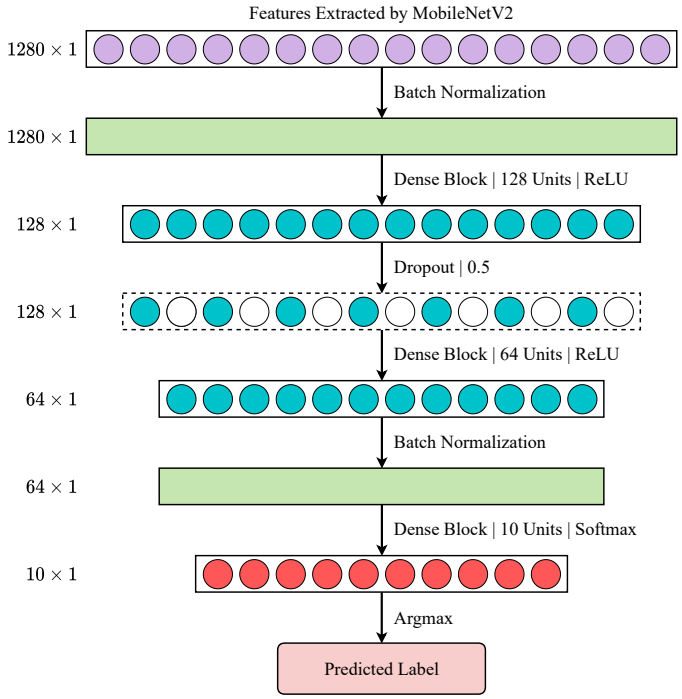


Figure 3.8: Classifier network

that the model does not overfit. The final output layer of the classifier network is also a densely connected block with a Softmax activation function [97]. The output layer of the classifier network contains ten nodes corresponding to each class label. The value of each node represents the probability of the input sample being in that class. Applying Argmax on this layer provides us with the predicted class label.

3.2.5 Experimental Setup

The proposed architecture was trained under a Python environment with TensorFlow, Keras, and other necessary libraries in Google Colab[1]. All experiments were conducted using an Intel Xeon CPU with a base clock speed of 2.3 GHz and an NVIDIA Tesla T4 GPU with a VRAM of 15 GB. The total usable memory of the machine was 13 GB.

From each class, the sample images were randomly split into 60% for training, 20% for validation, and 20% for the test set. Following the mini-batch gradient descent technique [98], the batch size was selected as 16. Since smaller batch sizes are often noisy, they help create a regularization effect and reduce the generalization error. They also help fit training data into memory.

While working with mini-batches, there is always a possibility of choosing batches that are not representative of the entire dataset, resulting in an inaccurate estimate of the gradient. Thus the training images were shuffled after each epoch throughout the

[1] `https://colab.research.google.com/`

experiments. This increases the probability that the model will converge and not be trapped with too many inferior batches. For experimenting with mini-batches, it is crucial to shuffle the data after each epoch. The goal of shuffling data is to reduce variance and ensure that models generalize well and do not overfit.

The model was trained for at most 1000 epochs with early stopping. Early stopping helps reduce overfitting and improves the generalization of neural networks. Validation accuracy was selected as the scheme for evaluating the model so that early stopping can be triggered. In our proposed approach, a change in validation accuracy between epochs was considered significant if it was greater than 10^{-4}. Otherwise, it was considered a patient epoch. The training was stopped early if there are ten consecutive patient epochs. Consequently, it was observed that our proposed architecture was able to converge after 70 epochs on average.

To ensure the rapid learning of salient features, we have used Adam optimizer [99] for training our model. Compared to other optimizers, Adam can help multilayer deep learning networks converge faster for computer vision problems. Since this is a multiclass classification task, we have used categorical cross-entropy loss. The initial learning rate was set to 10^{-5}. For every four consecutive patient epochs, the learning rate was decreased by a factor of 0.1 to help the model learn a set of globally optimal weights that leads to better optimization of the loss function.

The models can be initialized with different weights during the training phase, e.g., 0, random values, or pretrained weight values. In our work, we initialized the feature extractor part of the network with the respective pretrained weights from ImageNet Dataset [100] for the models and the classifier network with random weights. Model Checkpoints were used to save the model with the best validation accuracy so that they can be loaded later to continue the training from the saved state if required.

3.2.6 Evaluation Metrics

Accuracy

Accuracy is the ratio between the total number of predictions that were correct and the total number of predictions. To get a better estimation of the generalization capability of a model, the accuracy is calculated using the samples from the test set, which is unseen to the model during training.

$$\text{Accuracy} = \frac{M}{N} \times 100\% \tag{3.11}$$

Here, N is the number of samples in the test set and M is the number of samples for which the class labels were correctly predicted by the model.

Parameter Count

Parameters are the model's learnable weights, which are changed during the backward propagation phase based on the chosen optimization algorithm. The number of parameters can not only provide us with an idea regarding the training time of the model but also helps determine the model size and inference time.

$$\text{Parameter Count} = \sum_{i=1}^{L} p_i \tag{3.12}$$

Here, p_i is the number of parameters in the ith layer and L is the total number of layers in the model.

Model Size

Trained models can be stored as a Hierarchical Data Format version 5 (HDF5) file. The saved model contains the model's configuration, trained weights, and optimizer state. The model, along with its saved weights, can be loaded again to run inference. The size of the saved model is called the model size. Model size can be measured in MB (Megabyte) or GB (Gigabyte).

FLOPs Count

FLOPs Count is the theoretical maximum number of floating-point operations that a model requires to perform inference. Since the time taken for inference can vary from device to device, FLOPs Count is a better measurement to compare the relative inference time of deep learning models. It is usually measured in megaFLOPs (MFLOPs), gigaFLOPs (GFLOPs), or teraFLOPs (TFLOPs). The higher the value, the larger the number of computations required for a model to perform inference.

Precision

Precision is the ratio of the sum of the number of true positive predictions among all classes and the sum of the number of true positive predictions and false positive predictions among all classes. In a multiclass problem, for each class, precision is used to evaluate the correctly classified samples of that class among all the samples that were classified as of that class. Precision is also called Positive Predictive Value (PPV).

Precision for each class c can be calculated considering the one-vs-all strategy.

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c} \tag{3.13}$$

Here, TP_c is the number of samples correctly classified as c, and FP_c is the number of samples wrongly classified as c.

For imbalanced classes, macro-average precision is calculated where the precision for each class is calculated separately, and their average is taken. This ensures that the model gets equally penalized for each false positive instance of any class.

For a set of classes C,

$$\text{Macro Average Precision} = \frac{\sum_{c \in C} \text{Precision}_c}{|C|} \tag{3.14}$$

Here, Precision_c is the precision value for class c, and $|C|$ is the total number of classes.

Recall

Recall is the ratio of the sum of the number of true positive predictions among all classes and the sum of the number of true positive predictions and false negative predictions among all classes. In a multiclass problem, recall is used to evaluate how many samples are correctly classified among all the samples that should have been classified as of that class. Recall is also called Sensitivity.

Recall for each class c can be calculated considering the one-vs-all strategy.

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c} \tag{3.15}$$

Here, TP_c is the number of samples correctly classified as c, and FN_c is the number of samples of c that are wrongly classified as other classes.

For imbalanced classes, macro average recall is calculated where the recall for each class is calculated separately and their average is taken. This ensures that the model gets equally penalized for each false negative instances of any class.

For a set of classes C,

$$\text{Macro Average Recall} = \frac{\sum_{c \in C} \text{Recall}_c}{|C|} \tag{3.16}$$

Here, Recall$_c$ is the Recall value for class c, and N = Total number classes.

F1-Score

F1-Score is the weighted average of precision and recall that considers both the number of false positive predictions and false negative predictions. While working on an imbalanced dataset, having a high F1-Score is crucial to reduce the number of false positive and false negative predictions.

F1-Score for each class c can be calculated using the following formula:

$$\text{F1-Score}_c = \frac{2 \times P_c \times R_c}{P_c + R_c} \tag{3.17}$$

Here, P_c is the precision value for class c, and R_c is the recall value for class c.

For imbalanced classes, macro average F1-score is calculated where the F1-score for each class is calculated separately and their average is taken. This ensures that each class gets equal priority in classification.

For a set of classes C,

$$\text{Macro Average F1-Score} = \frac{\sum_{c \in C} F_c}{|C|} \tag{3.18}$$

Here, F_c is the F1-Score for class c, and $|C|$ is the total number of classes.

AUC-ROC Score

Precision, recall, and the majority of the commonly utilized metrics have their individual set of restrictions. Precision is a measurement that determines how accurate a classification task is and it is only based on true positive and false positive predictions; a score of 1.0 for precision can be achieved with just one true positive prediction. On the other hand, recall is all about the completeness and is solely based on true positive and false negative responses. As a result, predicting all the samples as positive will result in a recall of 1.0, but the precision will be very low. The Receiver Operating Characteristic (ROC) curve and the area under the ROC curve (AUC-ROC) are utilized as evaluation methods in this regard by combining the True Positive Rate (TPR) and False Positive Rate (FPR).

These methods allow models to be evaluated according to how well they separate classes from one another. The FPR and TPR for a series of predictions generated by the model at various thresholds are calculated to summarize the behavior of the model and can also be used to examine its ability to differentiate classes. Each probability

threshold is represented by a point, linked to form a curve in the ROC graph. A model with no discriminating power is depicted by a diagonal line from FPR 0 and TPR 0 (coordinates: 0,0) to FPR 1 and TPR 1. (co-ordinate: 1,1). A perfect model is represented as a point in the upper-left corner of the plot.

3.3 Result and Discussion

For our experiments, we first investigated the performance of different baseline Deep CNN architectures to choose the best fit for our requirements. After that, an ablation study was conducted to justify how the different considerations in our proposed pipeline and modifications over the baseline contributed to improving our model's performance. Next, we inspected per-class precision, recall, and F1-score to evaluate how the proposed architecture addresses the class imbalance issue. Then, we compared the performance of our model with the existing state-of-the-artwork of tomato leaf disease classifications to establish its superiority. Finally, an error analysis was conducted to figure out where to invest the future improvement efforts.

3.3.1 Performance of Different Baseline Architectures

To choose the baseline model, several state-of-the-art Deep CNN architectures were implemented to perform tomato leaf disease classification. A comparison of their performance is shown in Table 3.2.

The models were initialized with their pretrained weights on the ImageNet dataset and fine-tuned using the original tomato leaf samples from the PlantVillage dataset. The benefit of this initialization was that the models were already capable of learning complex patterns leading to faster convergence. Since our goal was to pick the best-suited baseline for the proposed system, we only changed the final softmax layer with the number of classes of our dataset and trained without any enhancement or augmentations.

While choosing the appropriate architecture, we have considered the accuracy, number of trainable parameters, estimation of the number of floating-point operations (FLOPs), and the model's size. The VGG19 and DenseNet201 architectures achieved an accuracy higher than 99% percent, and the performance of the ResNets also came close. These models are superior in terms of accuracy but have a significant disadvantage considering the other metrics. For example, the VGG19 model has achieved 99.48% accuracy, which is 2.2% higher than the accuracy of MobileNetV2 architecture. However, this improvement is costly in terms of memory and inference time. The model consumed 8.5 times the storage space and 8.8 times higher FLOPs count

Table 3.2: Comparison of the performance and characteristics among the baseline architectures on the original dataset

Architecture	Accuracy (%)	Parameters Count (Millions)	Model Size (MB)	FLOPs Count (MFLOPs)
DenseNet121	97.96	7.1	27.58	14.1
DenseNet201	99.36	18.35	71.11	36.69
EfficientNet-B0	96.94	4.1	15.89	8.1
MobileNet	96.53	3.2	12.51	6.5
MobileNetV2	97.27	**2.28**	**8.98**	**4.54**
NASNet-Mobile	97.21	4.3	17.53	8.6
ResNet50	98.70	23.62	98.29	51.11
ResNet152V2	98.62	58.36	223.52	116.61
VGG19	**99.48**	20.02	76.48	40.05

than MobileNetV2. Similar can be said for DenseNet201 as well. On the other hand, the relatively lighter models such as EfficientNet-B0, MobileNet, and NASNet-Mobile had lower accuracy than MobileNetV2 despite having higher values in terms of other metrics.

The MobileNetV2 architecture has the smallest model size and the lowest FLOPs count, making it ideal for real-time disease detection in devices with memory constraints. In addition to that, the fewer parameters of MobileNetV2 architecture result in faster training and inference. For these reasons, we chose MobileNetV2 as our base transfer learning architecture. We further aimed to improve the baseline performance by utilizing preprocessing techniques and an additional classifier network.

3.3.2 Ablation Study

An ablation study was conducted to understand the contribution of different components of the proposed pipeline to the overall performance. We considered several combinations of the design choices like the preprocessing steps, such as CLAHE, data augmentation, and the introduction of a classifier network to analyze their effects. A summary of the result in different settings can be found in Table 3.3.

A positive impact can be seen on the results when the images are preprocessed using CLAHE. This can be attributed to CLAHE for enhancing the leaf images' disease spots, making them more prominent and easier to identify for the models. For example, the baseline performance of 97.27% was improved to 97.71% after we introduced CLAHE. We noticed a further improvement in the results when data augmentation was introduced. The runtime augmentations allow the model to learn from different repre-

Table 3.3: Ablation study of different components of the proposed pipeline

CLAHE	Augmentation	Classifier Network	Accuracy
×	×	×	97.27
×	×	✓	98.29
×	✓	×	98.46
×	✓	✓	99.03
✓	×	×	97.71
✓	×	✓	98.60
✓	✓	×	98.84
✓	✓	✓	**99.30**

sentations of the images in every epoch, allowing the model to focus on the features highlighted by CLAHE.

Experiments were performed to find out how data augmentation in different splits affects the overall performance. We found that performing data augmentation on all three splits resulted in the best accuracy. As a result of the augmentation, the model learns to recognize different variations of the original image during the training phase. However, without augmenting the test set, no such variations are to be found. This violates the key assumption in dataset splitting for general classification tasks, that the distribution of images found in the training and validation set should be similar to the distribution in the test set.

One key factor here is that, as all our samples are being augmented with random probability during run-time, the model never sees the same version of an image twice. Augmentation in training and validation splits ensures that the model hardly gets any chance to overfit and learns generic feature representations. In addition, augmentations performed on the test set ensure that those samples represent real-life scenarios, making the classification task even more challenging. However, this begets a problem. Since the augmentation is performed randomly, the model sees different images in each test run. As a result, the accuracy for each run might not be the same; instead, it gives us a value within a range. To resolve this issue, we tested the trained model 100 times whenever we used augmentation and reported the average accuracy. The benefits of doing this are two-fold. First, as the test set is randomly augmented, the average accuracy is a better descriptor of the model's performance, preventing any chance of getting lucky. Moreover, these trials are testing the model with a variety of samples, more than what could be done using a static test set. So a model being able to do well in this setup will be robust and can be expected to achieve similar accuracy in real-life scenarios. It is worth mentioning that the maximum accuracy achieved by

our best model was 99.53%.

Our hypothesis of introducing the classifier network was that the model would be able to consider further combinations of the extracted features from the MobileNetV2 network, leading to improved overall performance. Since this network was trained from scratch on the provided information from the feature extractor network, it extracted even more meaningful features for leaf disease classification. Thus we found an improvement in the overall accuracy every time the classifier network was introduced in different setups.

Initially, the performance of the baseline MobileNetV2 model was only 97.27%. The combination of the preprocessing techniques increased it up to 98.84% showing how these choices improve the generalizing capability of the model. Finally, the model's competence was further enhanced with the classifier network leading to a mean accuracy of 99.30% (Standard Deviation: 0.00095) over 100 runs.

3.3.3 Addressing the Class Imbalance

As mentioned earlier, there exists a class imbalance in the PlantVillage dataset. Thus drawing a conclusion on a model's performance solely based on the accuracy metric might be unwise as the accuracy might still be in the 90^{th} percentile even if the model is incapable of classifying half of the samples of the least populated classes. To tackle this issue, macro-averaged precision, recall, and F1-score values were taken under consideration, which gives equal importance to all the classes regardless of the number of samples. Our proposed model achieves 99.18 precision, 99.07 recall, and 99.12 F1-score. The high values of precision and recall signify that our model does a great job identifying the True Positives. At the same time, it penalizes the accidental False Positive and False Negative cases. Taking a harmonic mean of these two metrics, the 99.12 value of the F1-score proves the robustness of the proposed architecture even in imbalanced datasets.

Furthermore, Table 3.4 shows the precision, recall, and F1-score for each class. From the table, it is evident that our data augmentation technique solved the class imbalance problem as these values are high even for the classes with a low number of samples.

3.3.4 Class Separability

The AUC-ROC curve can assess the performance of a predictive model by describing the trade-off between the True Positive Rate and False Positive Rate by employing multiple probability thresholds. A perfect classifier will have a ROC where the graph

Table 3.4: Per class precision, recall, F1-Score for the test set

Class Label	Sample Count	Precision	Recall	F1-Score
Bacterial Spot	425	0.9976	0.9953	0.9965
Early Blight	200	0.9745	0.9550	0.9646
Late Blight	381	0.9794	0.9974	0.9883
Leaf Mold	190	1.000	0.9895	0.9947
Septoria Leaf Spot	354	0.9915	0.9915	0.9915
Two-spotted Spider Mite	335	0.9852	0.9940	0.9896
Target Spot	281	0.9928	0.9858	0.9893
Yellow Leaf Curl Virus	1071	1.0000	0.9981	0.9991
Tomato Mosaic Virus	74	1.0000	1.0000	1.0000
Healthy	318	0.9969	1.0000	0.9984

reaches 100% true positives and zero false positives. We generally measure the number of positive classifications with an increment in the rate of false positives.

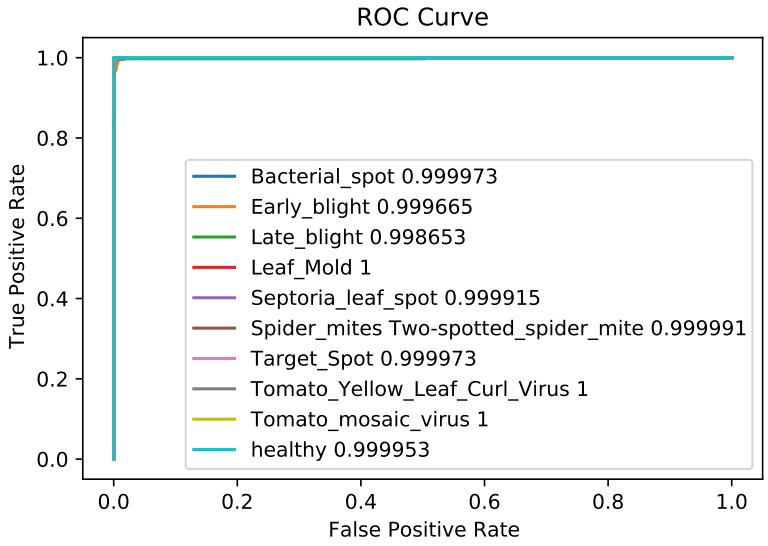


Figure 3.9: ROC curve for the tomato leaf diseases

As shown in Figure 3.9, the ROC curves overall each other at the top-left corner. That means, our proposed architecture achieves a commendable performance for all 10 classes. Among the classes, our model achieved AUC score of 1 for Leaf Mold, Yellow Leaf Curl Virus, and Mosaic Virus. Scores for the other classes are also fairly high, indicating a satisfactory class separability.

3.3.5 Comparison with State-of-the-art Methods

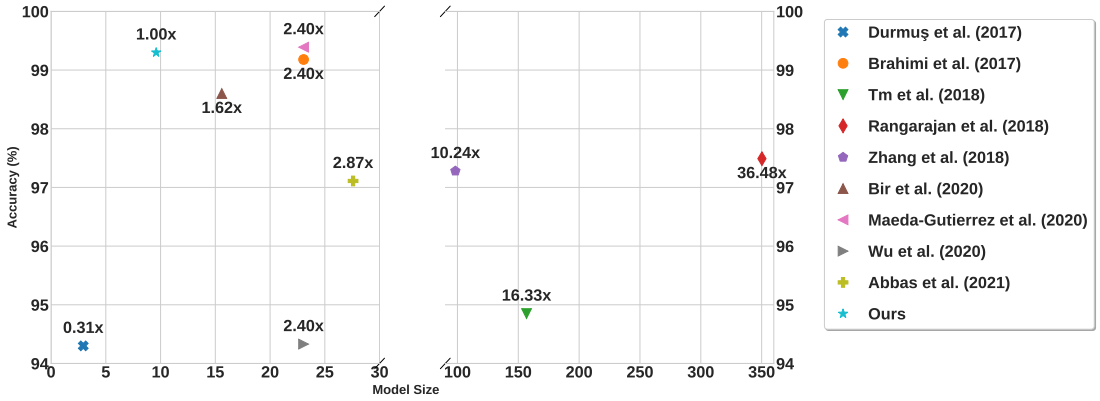
Table 3.5 presents a comparison of our proposed architecture with the state-of-the-art models of tomato leaf disease classification. Our model achieves a commendable accuracy of 99.30% while keeping the model size and the number of operations low. Comparing it to the state-of-the-art models, we can notice that only [39] achieved a mere 0.09% increase in accuracy, having 2.4 times the model size and 59.27% increase in FLOPs count. Our model's smaller size and low computational cost without sacrificing performance make it suitable for low-end devices.

Table 3.5: Performance comparison with the State-of-the-Art models for tomato leaf disease classification

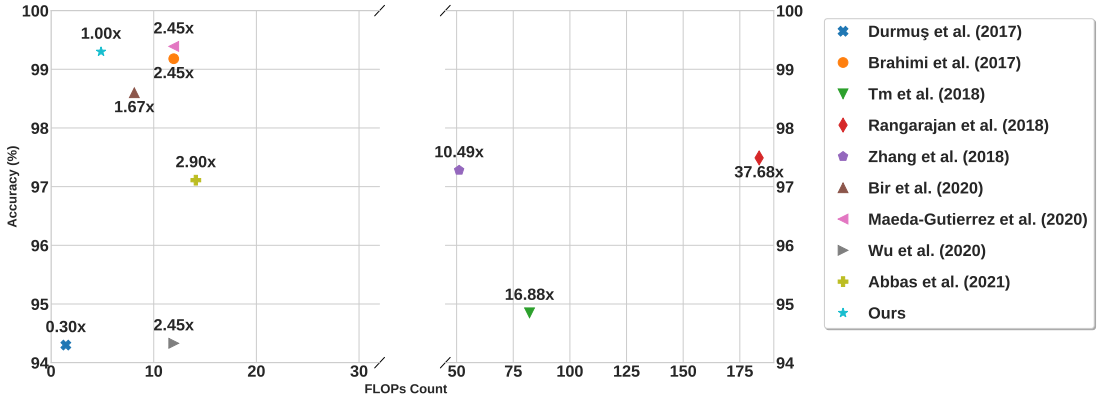
Reference	Class Count	Image Count	Accuracy (%)	Model Size (MB)	FLOPs Count (MFLOPS)
Durmuş *et al*. [44]	10	N/A	94.30	2.94	1.44
Brahimi *et al*. [36]	9	14828	99.18	23.06	11.95
Tm *et al*. [45]	10	18160	94.85	156.78	82.18
Rangarajan *et al*. [41]	7	13262	97.49	350.25	183.53
Zhang *et al*. [42]	9	5550	97.28	98.29	51.1
Bir *et al*. [46]	10	15000	98.60	15.59	8.11
Maeda-Gutierrez *et al*. [39]	10	18160	99.39	23.06	11.95
Wu *et al*. [40]	5	5300	94.33	23.06	11.95
Abbas *et al*. [43]	10	16012	97.11	27.58	28.09
Proposed Method	10	18160	99.30	9.60	4.87

Some of the works mentioned in the table did not utilize all the samples from the subset of the PlantVillage dataset, which leaves a possibility of accidentally missing some critical samples [40, 42, 43, 46]. Additionally, some of the models did not consider all the classes, which might lead to the misclassification of unseen samples. For example, [36] achieved an accuracy of 99.18%, but the experiment did not include any healthy samples of tomato leaves. This results in labeling a healthy leaf sample to any of the disease classes.

Further analysis shows that the space requirement of our proposed architecture is only 9.6MB. In contrast, different works in the existing literature required at least twice of this storage space, if not more, to produce similar accuracy (Figure 3.10a). Although [44] has a smaller model size than that of ours, the accuracy is far less. Figure 3.10b shows that our model significantly reduced the FLOPs requirement without compromising the accuracy. Hence it removes the requirement for high-performance hardware along with reducing the inference time of the model. It can be observed that



(a) Accuracy vs. Model Size



(b) Accuracy vs. FLOPs Count

Figure 3.10: Performance comparison with state-of-the-art tomato leaf disease classification architectures based on model size and FLOPs count

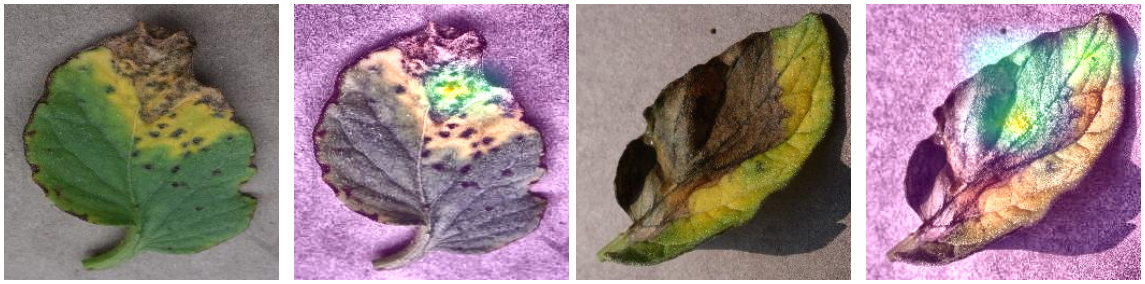
despite using deeper models, some works could not achieve comparable performance to the state-of-the-art. This further justifies the usefulness of the different components of our proposed architecture.

3.3.6 Qualitative Analysis

Focusing on the Diseased Portion

To get an intuitive understanding of whether our model is learning to correctly predict considering the relevant features or not, we examine the Gradient-weighted Class Activation Mapping (GradCAM) output for the correctly classified samples [101]. (Figure 3.11). This visualization can show us how our model classifies a diseased or healthy sample by highlighting the region in which our model focuses while making the class label decision.

As shown in Figure 3.11a and 3.11b, our model focused on the diseased portion of the leaf image to provide the class decision. Conversely, in the case of healthy leaf (Figure 3.11c), the model focuses on the entire leaf image only find no diseased



(a) GradCam output for Bacterial Spot **(b)** GradCam output for Late Blight

(c) GradCam output for Healthy Leaf

Figure 3.11: Qualitative results showing GradCam output for correctly classified samples

portion, and then classifies the image as healthy. This goes to show the capability of our model in understanding what to look for in a leaf image to make the correct class decision.

Error Analysis

According to the confusion matrix of our best performing model (Figure 3.12), for half of the classes, our model was able to predict all the unseen test samples correctly. For the rest, the accuracy is comparable to other state-of-the-art methods. However, the most misclassified samples were from the 'Early Blight' class. A few of the misclassified samples from this class were predicted as 'Late Blight'. Upon reviewing the misclassified samples, we identified visually similar leaves from both classes. For example, in the original dataset, the class label for Figure 3.13a is 'Early Blight', which was misclassified to 'Late Blight' class during inference. The GradCAM output for the sample shows that our model correctly focuses on the diseased region of the leaf image (Figure 3.13a).

However, in the training set of the Late Blight class, there are several images (e.g. Figure 3.13b) that are similar to Figure 3.13a. Since during training, the model learns to classify these images as of the class 'Late Blight', it is expected that similar images from the test set will also be classified in the same class.

To conclude, after analyzing the misclassified samples, we found some inter-class similarities in the infected regions among some of the diseases. A few of the leaves

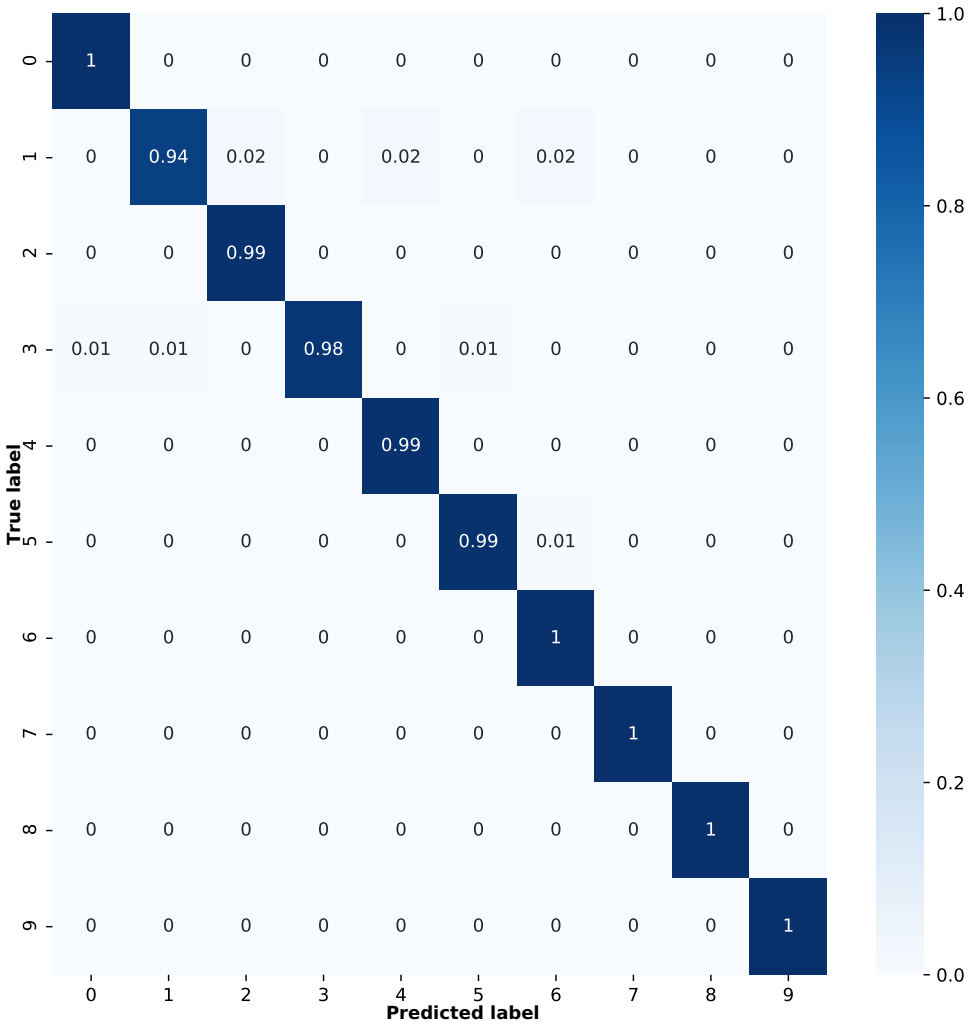
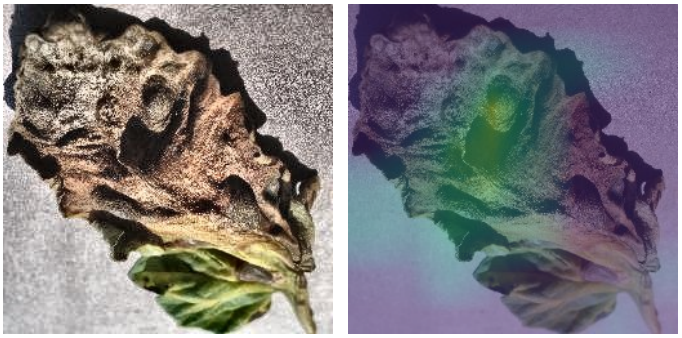


Figure 3.12: Confusion matrix. Here, 0 = Bacterial spot, 1 = Early blight, 2 = Late Blight, 3 = Leaf Mold, 4 = Septoria Leaf Spot, 5 = Two-spotted Spider Mite, 6 = Target Spot, 7 = Yellow Leaf Curl Virus, 8 = Tomato Mosaic Virus, 9 = Healthy

were severely damaged by the virus, which eventually restricted the model from extracting meaningful features leading to misclassification.



(a) Misclassified Early Blight Sample and its GradCAM output

(b) Similar Late Blight Samples

Figure 3.13: Misclassified sample with visually similar samples of the predicted class

Chapter 4

Leaf Disease Classification with Limited Data

Deep learning based architectures have achieved state-of-the-art performance in different leaf disease classification tasks. However, this high performance poses the requirement of huge amount of training data, which is a very difficult constraint to satisfy. With the goal of alleviating this limitation, we have proposed a pipeline which can produce high performance even with very limited samples. In this chapter, we have discussed the proposed pipeline for plant disease classification using Few-shot learning. First, we have provided an overview of the overall architecture. Then we have discussed each of the individual components and the justifications behind our choice.

4.1 Overview

The pipeline is built with the objective of classifying leaf diseases with very limited samples. It consists of two major components, the CNN-based feature extractor, and the classifier network. At first, the image is passed to the feature extractor block, which consists of nine state-of-the-art Deep CNN architectures. The image is passed through each of the individual CNN blocks and a feature vector is extracted from each of them. In this way, for each image, the feature extraction block produces nine 'observations' describing the different characteristics. Then all the nine features are concatenated to produce a combined feature vector which is then passed through a Classifier block for the final prediction. We have explored a variety of choices for the classifier block and found the best result by implementing a Bi-LSTM layer with 1024 nodes. Finally, the classifier block is densely connected with a softmax layer which provides the final prediction. A pictorial view of the proposed pipeline is shown in Figure 4.1.

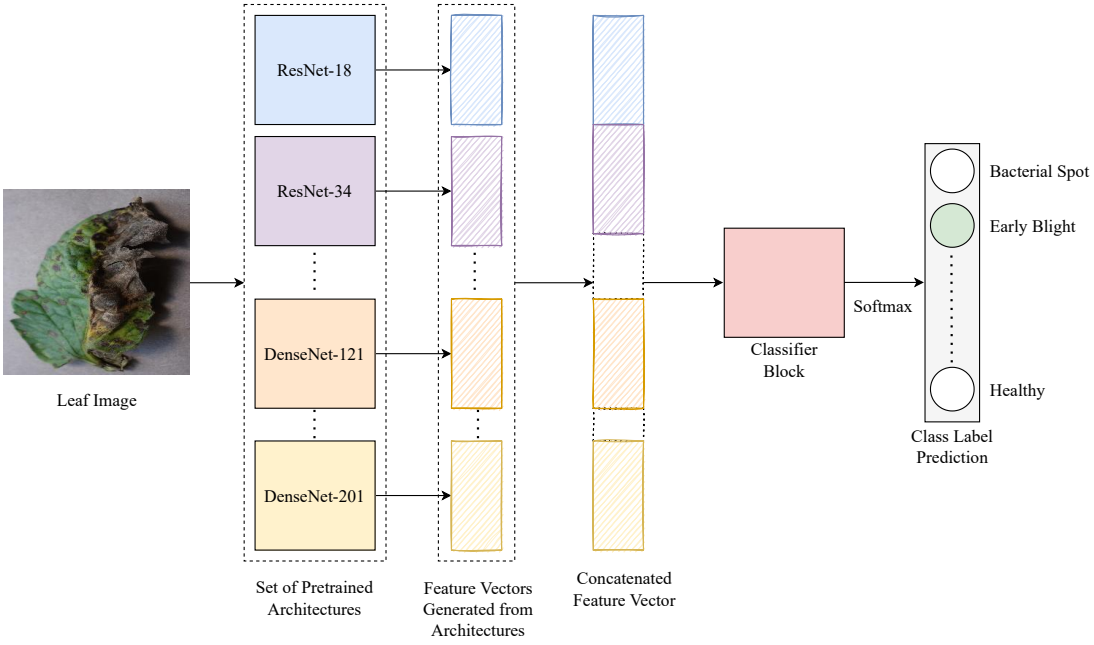


Figure 4.1: Overview of the Few-Shot Learning Pipeline

4.2 Materials and Methods

In the proposed pipeline, the CNN-based feature extractor produces a concatenated feature vector for each input. It is then passed to the classifier block to find deeper meaning and produce class predictions. To enhance the feature learning ability of the extractors, we have introduced the concept of domain adaptation. In the following discussions, we have explained each of these components and provided justification behind each design choice.

4.2.1 Few-shot Learning Tasks Formulation

Few-shot learning experiments usually have two phases; meta-training and meta-testing. The datasets used in these two phases have non-overlapping classes. Each of them is divided into Support set (S) and Query sets (Q) similar to the train-test splits of the traditional Deep Learning experiments.

The meta-training dataset is used to train the feature extractors to learn to produce good feature embedding from limited samples. The number of classes and samples in this portion is not limited as long as it does not overlap with the meta-testing portion. On the other hand, the meta-testing phase consists of the classes to be used for the Few-shot Classification problem. The number of classes in this portion is termed as N. The Support set of the meta-testing dataset consists of very 'few' samples, from which, k samples are taken for training the classifiers to effectively predict the labels N

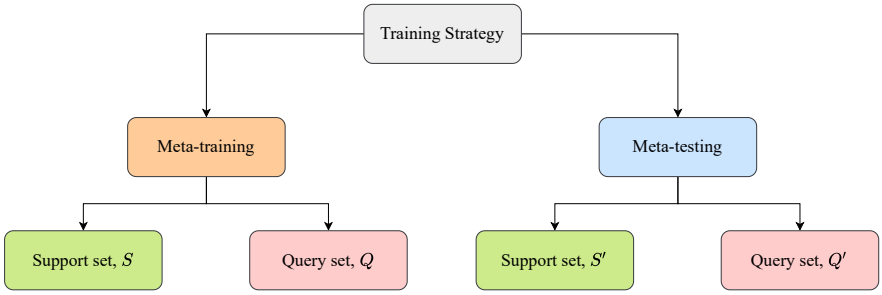


Figure 4.2: Training Strategy for Few-Shot Learning Experiment

classes. Hence, the Few-shot experiments are termed as N-way K-shot problems. The classification task can be termed as 1-shot, 5-shot, 10-shot, etc, respectively based on the values of $k = 1, 5, 10....$ Once the classifier is trained with the k-shots, Q-samples are taken from the Query set of the meta-testing dataset and the overall performance is calculated. This is termed a Few-shot Learning 'Task'. The models are usually tested for multiple such tasks with randomly chosen Support and Query samples for statistical significance.

4.2.2 Dataset for Single-Domain & Cross-Domain Experiments

The proposed pipeline has been tested under single-domain, mixed-domain and cross-domain conditions. In the single-domain experiments, the meta-training and meta-testing come from a similar types of domains with non-overlapping classes. On the contrary, for the cross-domain experiments, the datasets of meta-training and meta-testing phases come from two different domains. This task is even more challenging since it is very difficult to produce feature distribution by tackling the high amount of dissimilarity of the domains of interest. In mixed-domain FSL, both of the sets contain classes from multiple domains.

Single-Domain Experiment

For this experiment, we have used the PlantVillage Dataset, which is at present the largest open-access repository of expertly curated samples for leaf disease classification tasks. The dataset consists of 38 classes belonging to 14 crops. Out of these classes, we have picked the 10 classes of tomato leaf images for meta-testing phase and used the rest of the 28 classes during meta-training. Figure 4.3 shows the split of the PlantVillage Dataset used in our experiment.

In the meta-training phase, the 28 classes have been used to fine-tune the feature extractors to enhance their capabilities specially for leaf disease identification. On the contrary, the 10 classes of tomato leaf samples have been divided into Support

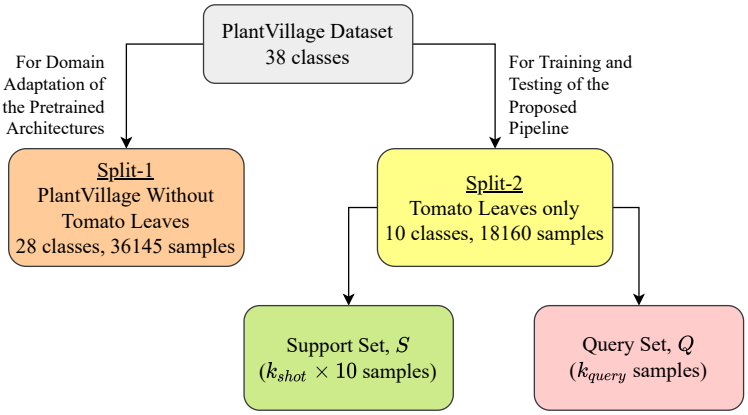


Figure 4.3: Split of the PlantVillage Dataset for Few-Shot Learning Experiment

and Query sets with an 80:20 ratio. Hence, out of the 18160 samples of tomato leaf images, 3629 samples have been considered as the available samples for the Query set, from which Q number of samples will be picked to test the model. The rest of the tomato leaf images will be available for the Support set, out of which k-shots will be randomly picked for training the classifier. For each task, k Support samples and Q query samples are randomly picked from the available images. To remove biases, this process has been repeated 100 times and the average performance has been reported.

Mixed-domain & Cross-Domain Experiment

For the Mixed-domain and Cross-domain experiments, we have utilized the 'Plants & Pest' dataset proposed by Li *et al.* [62]. The dataset consists of 20 classes in total, out of which, 10 classes belong to different kinds of plants such as apple, blueberry, cherry, grape, etc; and the other 10 classes contain images of different kinds of pests such as cydia pomonella, fruit fly, gryllotalpa, snail, etc. Each of the classes contains 300 images.

We have followed the recommended splits by the original authors for the mixed and cross-domain experiments. In the mixed-domain tasks, we have taken five plants and five pest classes for meta-training and the rest of the classes for meta-testing. For the Cross-domain experiments, we have conducted two kinds of experiments. In the first one, the meta-training set consists of the 10 pest classes and in another cross-domain scenario, we have considered all the plant classes for meta-training and the remaining classes for meta-testing.

4.2.3 CNN Based Feature Extractor

The general objective of any Few-shot Learning (FSL) pipeline is to produce very good feature embeddings from the provided 'few' samples. The different variants of the FSL

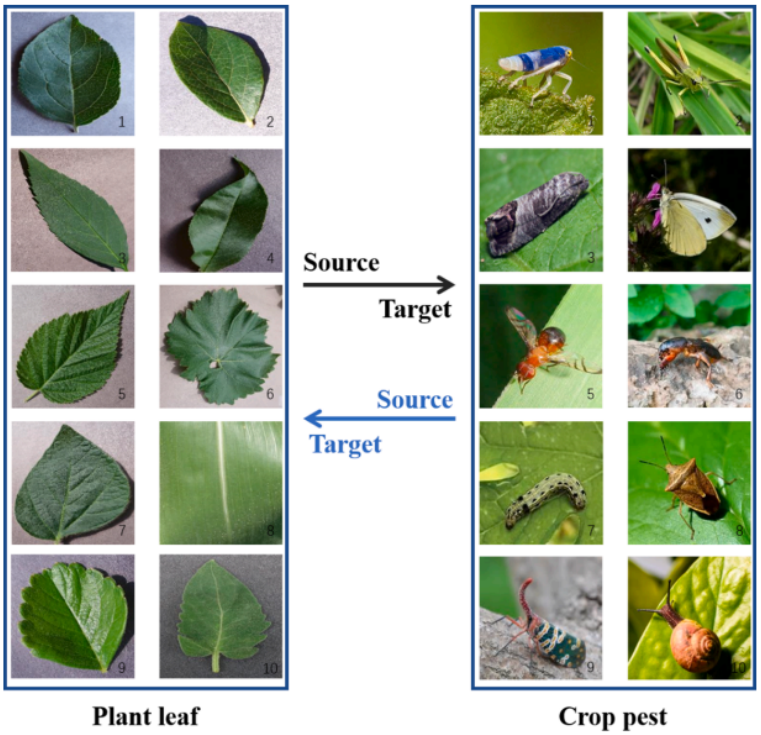


Figure 4.4: Dataset for Cross-Domain Experiment (courtesy to [62])

algorithm have evolved keeping this goal in mind. One of the possible solutions to this problem can be to use a pretrained CNN-based network to extract features and then pass them through a classifier block. Since the CNN architecture for feature extraction is usually pretrained with a very large dataset with a wide variety of classes under challenging conditions, it is supposed to be able to extract meaningful features from the provided samples. Usually, such features are passed to the softmax layer for final prediction. But in the FSL applications, the extracted features are expected to be passed through a classifier block. The classifier block is trained with the 'Support samples' of the 'meta-testing' set.

Although this idea has worked tremendously in many applications, the process is data hungry. Since the number of samples per class is very less here, so it is not enough to train the CNN architectures, which results in poor performance. As a solution to this issue, we have thought about the problem in a different manner.

Each of the CNN architecture produces a feature vector utilizing the spatial information of the images. It can be thought as, the feature extractor is the 'critic' and the provided features can be compared to its 'observation' on the image. In traditional Deep Learning approaches, each model is fed with many samples, so it can be trained with the goal of improving its ability to produce highly general observations from the given samples. However, in Few-shot learning problems, the models cannot be fed

with such a huge amount of data. So our hypothesis says, combining the observation of multiple Deep CNN architectures can be a possible solution to tackle this scenario. Each of the variants of the state-of-the-art CNN architecture has its ability to generate features in a different manner. So, as we are unable to provide enough samples to the model for getting expert observation, we aim to show the 'few samples' to multiple observers, take their observations, and combine them to produce a highly general observation for each sample.

The CNN-based feature extractor block consists of nine state-of-the-art CNN architectures, which are, ResNet18, ResNet34, ResNet50, ResNet101, ResNet152, DenseNet121, DenseNet161, DenseNet169, DenseNet201. These models have been chosen considering their remarkable performance in large-scale image classification tasks [102]. Each of the individual models is pretrained on the ILSVRC2012 dataset. We discarded the last layer and took only the feature embeddings. In the following discussions, we have discussed the characteristics of the pretrained CNN architectures under two categories; Residual CNN Architectures and Densely-Connected CNN architectures.

Residual CNN Architectures

Residual Network (ResNet) [12] is one of the most popular deep learning architectures. The architecture improves upon the status quo set by previous architectures by helping to train a large number of stacked layers. Traditionally, increasing the number of layers in a deep learning architecture improves the performance by a significant margin [8, 14, 94]. It is expected that deeper architectures are able to combine activation maps of many layers resulting in a deeper understanding of the inherent features of the input. However, even when the number of layers of architectures is increased, the performance of the networks gets saturated, and even degrades rapidly [103, 104]. This problem is caused by the notorious vanishing/exploding gradients problem. With the increase in the number of layers, the flow of gradients which are used to optimize the network is reduced by a constant factor in each layer, reducing the improvement of performance.

ResNets solve the vanishing gradient problem by introducing residual connections in-between layers. This connection facilitates the flow of information from shallow layers to deeper layers. At the same time, it enables gradients to flow back skipping a few layers. For this reason, these residual connections are also known as skip connections.

As seen in Figure 4.5, generally, the network takes x as input and produces $\mathcal{F}(x)$ as output. In the case of ResNets, the output is changed to $\mathcal{F}(x) + x$. That means the

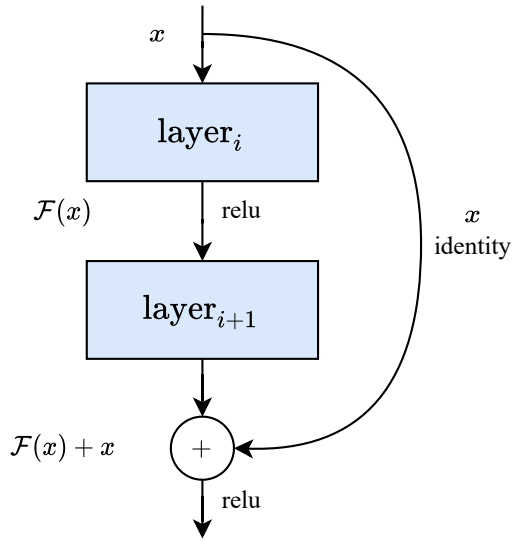


Figure 4.5: Residual Connection Block [Adapted from [12]]

features are combined via summation operation. Considering the change in dimensions due to convolution and/or pooling layers, one of the following two approaches can be adopted:

- Padding zeros with x in the skip connection to increase the dimension
- Adding 1×1 convolution layers to match the output dimension

These residual connections help solve the vanishing gradient problem by introducing 'shortcut' paths for gradient flow throughout the network. Again, due to regularization, any layer hurting the performance of the model can be skipped.

The authors proposed 5 variants of ResNets, namely ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. A ResNet-X architecture consists of X layers. But all of them contain the same residual connection in-between layers. The layer configurations are shown in Table 4.1.

Table 4.1: Layer configurations of the variants of ResNet architectures for input size 224×224. The input and output of the blocks in square brackets are connected with residual connections.

Layers	Output	ResNet-18	ResNet-34	ResNet-50	ResNet-101	ResNet-152
$conv1$	112×112	7×7, 64, stride 2				
$conv2\_x$	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
$conv3\_x$	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
$conv4\_x$	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
$conv5\_x$	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
Model Size (MB)		23	35	103	155	219
FLOPs		2×10^9	4×10^9	4×10^9	8×10^9	11×10^9

Densely Connected CNN Architectures

Expanding on the ideas of ResNet's skip connections, Dense Convolutional Network (DenseNet) [6] connects all layers with matching feature-map sizes directly with each other. Here, each layer receives input from all the previous layers and sends its output to all the subsequent layers. Additionally, instead of combining features through summation like ResNets, DenseNet architectures concatenate features. This alleviates the problem of dimension matching. That means the layer l of an L layer network receives l inputs combining the activation maps of all the previous layers. Again, the activation map produced by layer l is passed on to all $L-l$ subsequent layers. This creates $\frac{L(L+1)}{2}$ connections between different layers. Creating a dense connection between all the layers ensures the maximum flow of information and gradient between layers. In addition to that, it reduces the number of parameters compared to that of ResNets, since the layers in the DenseNets are narrow.

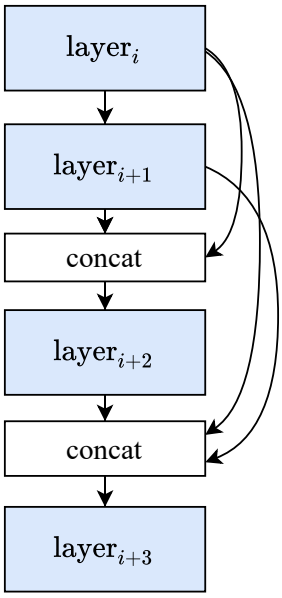


Figure 4.6: Dense Convolutional Block [Adapted from [6]]

As seen in Figure 4.6, the layer l receives the activation maps of all the previous layers as input:

$$x_l = \mathcal{F}([x_0, x_1, \ldots, x_{l-1}]) \tag{4.1}$$

where, $[x_0, x_1, \ldots, x_{l-1}]$ refers to concatenation operation.

Based on the number of layers, the authors proposed 4 variants of the DenseNet architecture, namely DenseNet-121, DenseNet-161, DenseNet-169, and DenseNet-201. The layer configurations are shown in Table 4.2.

Table 4.2: Layer configurations of the variants of ResNet architectures for input size 224×224

Layers	Output	DenseNet-121	DenseNet-161	DenseNet-169	DenseNet-201
$conv1$	112×112	$7\times$ conv, stride 2			
$pooling$	56×56	3×3 max pool, stride 2			
$dense1$	56×56	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 6$
$trans.1$	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
$dense2$	28×28	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 12$
$trans.2$	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
$dense3$	14×14	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 36$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 64$
$trans.3$	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
$dense4$	7×7	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \end{bmatrix} \times 32$
$classi.$	1×1	7×7 global average pool			
		$1000 - d$ fc, softmax			
Model Size (MB)		31	110	55	77
FLOPs		3×10^9	8×10^9	3×10^9	4×10^9

4.2.4 Combining the Extracted Features

In the feature extraction block, we utilized nine state-of-the-art CNN-based networks from which we get nine features for each input image. These features can also be termed as the 'observations' from the 'critics' i.e. the models. Since each critic has its unique characteristics, each observation carries some unique description of the sample. Hence, we combine the features before passing them to the classifier and expect it to find out even more meaningful features. We explored different ways to utilize the extracted observations and empirically found that concatenating all the observations to produce a single feature vector is the best way to pass the information to the classifier network.

4.2.5 Classifier Network

The classifier network takes the output obtained from the pretrained features extractors and uses a bidirectional LSTM-based [105] architecture to classify the features. As an addition to standard LSTMs, bidirectional LSTMs can improve the model's performance on a variety of classification tasks. Instead of training just one LSTM on the input features, Bi-LSTMs train two LSTMs. The first one is on the input features as-is and the other on a reversed copy of the input features. A basic Bi-LSTM network architecture is depicted in Figure 4.7. The basic building block of a Bi-LSTM network is a simple LSTM block which is represented in Figure 4.8.

Intentionally, LSTMs are created to prevent the long-term dependency issue. They do not strain to learn; rather, remembering knowledge for extended periods is their default habit. All recurrent neural networks have the form of a series of neural network modules that repeat. This recurring module in typical RNNs will be made up of just one tanh layer, for example. Although the repeating module of LSTMs also has a chain-like topology, it is structured differently. There are four neural network layers instead of just one, and they interact in a very unique way.

The horizontal line that runs across the top of the Figure 4.8 represents the cell state and is the key to LSTMs. The cell state resembles a conveyor belt in certain ways. With only a few tiny linear interactions, it proceeds directly down the whole chain. Information can very easily continue to travel along it unmodified. The LSTM may modify the cell state by removing or adding information, which is carefully controlled via gates. Information can pass via gates on a purely voluntary basis. They consist of a pointwise multiplication process and a layer of sigmoid neural networks. Indicating how much of each component should be allowed through, the sigmoid layer generates integers between zero and one.

Choosing whatever information from the cell state to discard is the first stage in

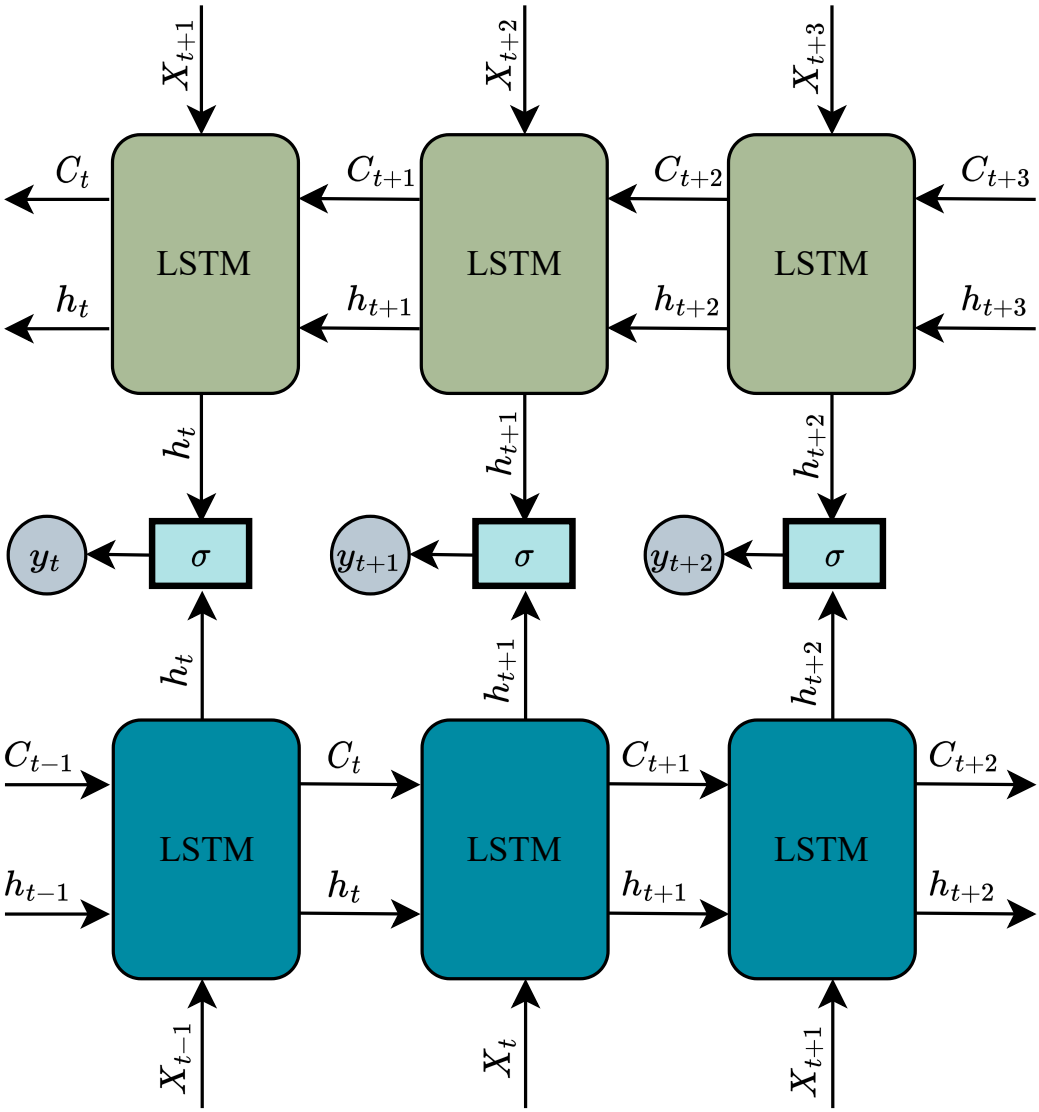


Figure 4.7: Bidirectional LSTM Architecture

our LSTM. The "forget gate layer," a sigmoid layer, decides on this. It examines h_{t-1} and x_t, and for each number in the cell state C_{t-1}, it produces a number between 0 and 1. The next step is to choose the new data that will be kept in the cell state. Two parts make up this. The "input gate layer", a sigmoid layer, first determines which values will be updated. The state is then updated with a vector of potential new values, C_t, created by a $tanh$ layer. These two will be combined in the subsequent phase to provide an update on the state.

Finally, we have to choose what we will produce as output. This output, however filtered, will be based on the status of our cell. We first run a sigmoid layer to determine which portions of the cell state will be output. Then, in order to output just the portions

we chose to, we multiply the cell state by the output of the sigmoid gate after passing the cell state through $tanh$ (to push the values to be between -1 and 1).

In the case of Bi-LSTM, two separate LSTM network is run concurrently one from the left to right and the other one from the right to left. In each step. output from both networks are concatenated and passed through a sigmoid layer to produce the output which will, in turn, be used for calculating the loss and backpropagation. In traditional LSTM architectures, features that come before the current feature are only responsible for the current output. The latter features actually could affect the present output being generated. Because of this, taking into account the features from both sides tends to significantly increase the classification performance.

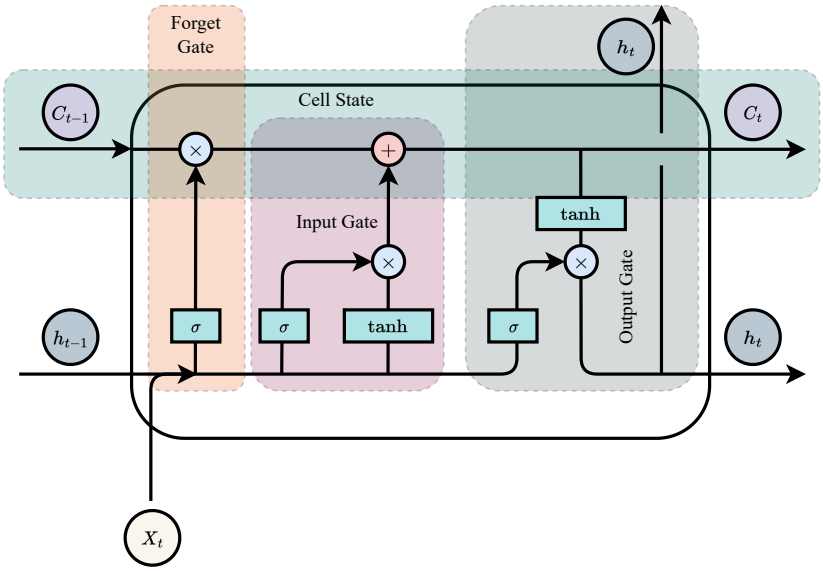


Figure 4.8: The Repeating Module in a Basic LSTM Network

4.2.6 Domain Adaptation

The feature extractor block consists of a total of nine variants of the ResNet and DenseNet architectures pretrained on the large-scale ImageNet dataset containing a wide variety of samples from diversified classes. Hence they are capable to extract very highly sophisticated features that can be utilized for any classification problem. When these networks are used for any classification task, a common practice is to fine-tune the architectures based on the available training images. However, in the few-shot classification problems, it is not possible to fine-tune such deep models with very limited samples. An alternative solution to this problem is to provide knowledge of a similar domain so that, despite the model having never seen the target classes, it has some idea about their general characteristics. In this work, we have named this approach 'Domain Adaptation'.

To apply domain adaptation in the proposed pipeline, we have trained the CNN-based architectures with a good amount of leaf images containing healthy and diseased samples. For this, we used the 28 classes of the PlantVillage dataset except for the samples of the tomato leaves. Our intuition behind this experiment was that as the models were already capable of extracting useful features, this knowledge base can be further enriched by showing them a lot of leaf samples. Although at this point, the model has never seen any 'tomato leaf samples', it knows what a healthy or diseased leaf looks like, what are the features to look for in a leaf image, and how to differentiate the different classes of leaves, etc. Thus when the limited samples are provided, the 'domain adapted models' can extract even better features compared to their 'pretrained' states.

4.3 Results & Discussions

In this section, we have evaluated and discussed the performance of the model under different circumstances. At first, we explained the experimental setup for the experiments. Then we thoroughly investigated the design choices of the proposed pipeline such as the appropriate way to combine features, the choice of classifier, and the importance of domain adaptation. Then we tested the pipeline in single-domain, mixed-domain and cross-domain scenarios.

4.3.1 Experimental Setup

The proposed pipeline has been trained under a python environment with PyTorch and other necessary libraries in Google Colab. We conducted the experiments on the provided Intex Xeon CPU with a base clock speed of 2,3 GHz. The available GPU was the NVIDIA Tesla T4 with a Virtual RAM of 15 gigabytes.

For the single-domain experiments, the meta-testing phase was conducted with 10 classes of tomato leaf images, which were divided into Support and Query sets with a split ratio of 80:20. The remaining 28 classes of the PlantVillage dataset were used for the meta-training phase. For each evaluation task, k training samples and Q testing samples were randomly picked respectively from the support and query set of the meta-testing portion of the dataset. Hence, it was a N-way k-shot problem, with the value of N fixed to 10, and the value of k was taken as 1, 5, 10, 15, 20, 40, 80, etc. in different experiments.

The batch size was selected as 32 following the mini-batch gradient descent technique [98]. This was chosen first to fit the data into the available memory. Moreover, the smaller batch size is often noisy, and thus they help to create a regularization effect in the overall training process. Since there were many samples available for the support

and query set, each time the k and Q samples were randomly chosen and the process was evaluated for 100 tasks to incorporate statistical significance. Since the samples are randomly picked, the average accuracy is shown for each experiment along with the 95% confidence interval score to show the deviation.

While fine-tuning the CNN-based feature extractor with the meta-training set, we allowed the individual networks to be trained till convergence and used early stopping. It helps to reduce the overfitting issue and improves generalization. For each task, the classifier network, was trained with the k-shots for 100 epochs with L2 regularization. The learning rate used to train the classifier was 0.0005.

4.3.2 Selection of Feature Combination

Our first hypothesis is that as the number of samples per class is very less, the capability of pretrained architectures can be utilized to build the feature space. For this reason, we used the extracted features of nine pretrained architectures and concatenated them to produce a combined 'observation' of the image, and provide it to the classifier.

Single vs Ensembled Feature Extractor

At first, the performance of each of the feature extractors was analyzed thoroughly for different values of k. We explored five variants of ResNet and four variants of the DenseNet architectures since they have been proven to provide state-of-the-art performance in different large-scale image classification tasks. Along with finding the performance of the individual networks, we also considered the ensemble of the ResNets, an ensemble of DenseNets, and finally the ensemble of all nine architectures. Since we focused on finding the best combination of feature extractors, we used only a Dense layer of 1024 hidden units followed by a softmax layer. The findings of this experiment are shown in Table 4.3.

We conducted the experiment for four combinations of $k = 1, 5, 10, \&15$. For the test samples, $Q = 50$ images per class were taken at random. To remove any kind of bias, this random sampling of k, Q was conducted 100 times and the average value was reported. The objective was to find the feature extractor which works best with the provided 'few-shots'. Among the residual architectures, ResNet18 was found to be the best performing one. The deeper versions of the residual architectures are designed for classifying large datasets, so due to the very limited number of samples, the shallower layer performed better. The DenseNet architectures performed similar to or better than the ResNets in general. Among the single feature extractors, the DenseNet201 was the best. Compared to the ResNet18 architecture, it provided a similar performance

Table 4.3: Single vs Ensembled Feature Extractor

Feature Extractor	$\boldsymbol{K=1}$	$\boldsymbol{K=5}$	$\boldsymbol{K=10}$	$\boldsymbol{K=15}$
ResNet18	34.36 ± 0.82	55.42 ± 0.77	62.56 ± 0.67	66.76 ± 0.62
ResNet34	29.65 ± 0.91	49.93 ± 0.81	57.93 ± 0.56	63.69 ± 0.54
ResNet50	30.62 ± 0.88	54.24 ± 0.92	61.79 ± 0.65	65.99 ± 0.64
ResNet101	29.64 ± 0.93	52.73 ± 0.79	61.61 ± 0.74	65.79 ± 0.68
ResNet152	29.75 ± 0.84	52.61 ± 0.86	61.39 ± 0.74	66.44 ± 0.68
DenseNet121	32.48 ± 0.99	54.93 ± 0.89	64.09 ± 0.72	68.21 ± 0.56
DenseNet161	32.29 ± 1.07	56.58 ± 0.88	65.59 ± 0.71	70.56 ± 0.66
DenseNet169	32.89 ± 0.97	55.39 ± 0.78	64.09 ± 0.72	69.39 ± 0.7
DenseNet201	33.51 ± 0.94	57.21 ± 0.87	65.71 ± 0.86	70.77 ± 0.73
Ensemble of ResNets	36.34 ± 1.01	60.04 ± 0.91	68.74 ± 0.71	73.84 ± 0.68
Ensemble of DenseNets	35.94 ± 1.03	61.47 ± 0.96	70.26 ± 0.79	75.42 ± 0.65
Ensemble of Nine Extractors	$\mathbf{37.89 \pm 0.99}$	$\mathbf{62.07 \pm 0.98}$	$\mathbf{72.33 \pm 0.81}$	$\mathbf{76.93 \pm 0.66}$

for one-shot classification and provided $2-3\%$ improvement for 5-shot, 10-shot, and 15-shot classification.

Since the performance of the network was close to each other, our intuition said that ensembling them can provide better results because each of them might have unique strengths in extracting certain types of features useful for detecting specific types of diseases. We applied three types of ensembles of the CNN-based feature extractors. By ensembling similar types of networks, we achieved better performance than the individual ones. The best performance was achieved by combining all the nine feature extractors. This justifies our hypothesis; 'combined observation of expert critics are few-shot learners'.

Table 4.4: Finding the Best Way to Combine Features

Observation Type	**Classifier**	**Domain Adopt**	$k = 1$	$k = 5$	$k = 10$	$k = 15$
Parallel	Dense	×	40.2 ± 1.46	72.95 ± 0.97	82.6 ± 0.97	85.71 ± 0.78
Parallel	Dense	✓	50.38 ± 1.68	83.78 ± 1.16	89.07 ± 0.91	92.79 ± 0.69
Parallel	Bi-LSTM	×	39.84 ± 1.65	48.27 ± 2.20	52.66 ± 1.75	63.42 ± 1.78
Parallel	Bi-LSTM	✓	53.4 ± 1.74	74.75 ± 1.68	79.3 ± 1.90	81.99 ± 1.81
Concatenated	Dense	×	37.89 ± 0.99	62.07 ± 0.98	72.33 ± 0.81	76.93 ± 0.66
Concatenated	Dense	✓	46.56 ± 1.07	72.17 ± 1.04	81.18 ± 0.68	85.12 ± 1.02
Concatenated	Bi-LSTM	×	42.53 ± 1.25	81.9 ± 1.37	87.80 ± 1.23	89.06 ± 1.22
Concatenated	Bi-LSTM	✓	$\mathbf{51.36 \pm 1.54}$	$\mathbf{89.19 \pm 1.20}$	$\mathbf{90.75 \pm 1.09}$	$\mathbf{94.02 \pm 0.86}$

Combining the Extracted Features

Once the leaf images are given into the proposed pipeline, the feature extraction block provides nine unique observations. These features are then combined and sent to the classifier block to find out deeper meanings. To combine the features, we have attempted two approaches; which we have named, 'Parallel Observation' and 'Concatenated Observation'.

In the Parallel Observation approach, the nine observations, i.e, the nine different features provided by the CNN-based extractors are passed to the classifier network separately. This can be interpreted as if, nine observers are describing the image to the classifier network. Since the inherent architecture of each of the feature extractors has a difference, each of the observations carries some 'unique' information about the sample. It provides the model with an opportunity to create an effect of nine times more samples than what is provided.

On the other hand, the Concatenated Observation approach combines the nine features into one observation by concatenating with one another, resulting in a $13,984$ dimensional feature vector per sample. The objective of the classifier is to pick the most important information from the different observations of the feature extractors and combine them to produce even more meaningful one.

To compare both of the approaches, we conducted a thorough ablation of the dif-

ferent design choices. For each approach, two types of classifiers have been used. The first classifier is a simple densely connected layer of 1024 nodes, which has been termed as 'Dense Classifier'. The other classifier consists of a Bi-directional LSTM layer of 1024 nodes, termed as 'Bi-LSTM Classifier'. We have also checked the impact of domain adaptation for each of the combinations. For this experiment, we have considered the values of k as $1, 5, 10\&15$ and $Q = 50$ applied for 100 tasks. The findings are mentioned in Table 4.4.

For each combination, the domain adapted state has always provided $5 - 10\%$ performance improvement compared to the feature extractors which have only been trained on the ImageNet dataset. The Dense classifier has been found to be a better fit for the parallel observation approach. On the contrary, the Bi-LSTM classifier is more suitable for the feature concatenation approach. Out of all the experiments, the Domain Adapted Concatenated feature extraction approach with Bi-LSTM classifier is the best performing one, having an accuracy as high as 90.97% and 94.02% respectively for 10-shot and 15-shot classification.

4.3.3 Selection of Classifier Network

After finding the appropriate way to combine the feature, we explored different choices of classifiers intending to extract even deeper meanings. For this, we considered the Dense, LSTM, GRU, Bi-GRU, and Bi-LSTM classifiers. Each of the classifiers consisted of 1024 nodes followed by a softmax layer of 10 nodes. Experiments have been conducted on the base feature extractors without applying domain adaptation, with $k = 1, 5, 10\&15$. For this analysis, we took the entire test set into account with a total of 3629 images. To check the statistical significance of the results, we investigated them 100 times. The experimental findings have been reported in Table 4.5.

In dense classifier networks, the current output is based on the current input features which have no impact on the future or the past feature representations. Incorporating both past and future features help the classifier to generalize on the current input thus, obtaining a better classification result. In general, recurrent layers performed better than dense layers. One intuition might be that the dense layers are forced to have at least some weight on each of the connections, whereas, the recurrent layer has the opportunity to completely forget some of the components of the observations.

To incorporate left-to-right and right-to-left features in the current input, we have used RNN-based classifiers namely, LSTM [106] and GRU [107]. From the obtained results, it can be observed that the performance difference between GRU and LSTM is minimal. The reason behind this is, LSTM was proposed to eradicate the vanishing gradient problem of traditional RNN-based architectures. LSTM-based architectures

successfully do so, with the help of 4 gates in every LSTM unit that is repeated throughout the network. The increased number of gates in each block increased the training time as well as the required resources. To optimize the training and inference, Gated Recurrent Unit (GRU) was proposed which comprised of a reset gate and an update gate. Our experiments show that LSTM performs better with less training data due to its more powerful unit block whereas GRU performs better with more training samples.

The input features obtained from our pre-trained feature extractors are significant to the final output or the classification results. The features are fed to the classifier network in the form of a one-dimensional vector in a sequential manner. Considering the features that only the network has seen before might not be sufficient for the current output prediction and that is why the features that will come after it are also considered. Our experimental results also prove this hypothesis as the bidirectional classifiers, both LSTM and GRU, performs better than their unidirectional counterpart. The performance of Bi-LSTM [105] is significantly better than Bi-GRU for lower values of K and the performance, the gap minimizes as the value of K increases. In our experiments, the Bi-LSTM-based network provides the best performance for each value of K while taking slightly more training time than Bi-GRU.

Table 4.5: Different Choices of Classifier Network

Classifier	$\boldsymbol{K = 1}$	$\boldsymbol{k = 5}$	$\boldsymbol{K = 10}$	$\boldsymbol{K = 15}$
Dense	37.89 ± 0.99	62.07 ± 0.98	72.33 ± 0.81	76.93 ± 0.66
LSTM	43.24 ± 1.72	69.16 ± 1.75	74.15 ± 1.52	72.84 ± 1.02
GRU	40.21 ± 1.55	68.15 ± 1.54	78.97 ± 1.54	81.08 ± 1.56
Bi-GRU	43.03 ± 0.51	72.69 ± 1.28	85.53 ± 1.26	90.94 ± 1.03
Bi-LSTM	$\mathbf{48.02 \pm 1.63}$	$\mathbf{84.36 \pm 1.62}$	$\mathbf{88.97 \pm 1.80}$	$\mathbf{91.22 \pm 1.67}$

4.3.4 Impact of Domain Adaptation

The baseline CNN-based feature extractors are pretrained on larges scale ImageNet datasets with the capability of extracting useful feature features from any input samples. However, to make the feature extractors even more specialized in finding appropriate leaf features, we applied the domain adaptation concept. This significantly improved the overall performance of the pipeline. We have used the Bi-LSTM clas-

sifier since it has been found to be the most useful one on the feature concatenation approach. The experimental findings are available in Table 4.6.

Table 4.6: Impact of Domain Adaptation

k-shot	Domain Adapt	$Q = Full$
$k = 1$	$\times$	48.02 ± 1.63
	✓	56.19 ± 1.72
$k = 5$	$\times$	84.36 ± 1.62
	✓	89.06 ± 1.30
$k = 10$	$\times$	88.97 ± 1.80
	✓	92.46 ± 1.75
$k = 15$	$\times$	91.22 ± 1.67
	✓	94.07 ± 1.40
$k = 20$	$\times$	92.66 ± 1.87
	✓	95.68 ± 1.20
$k = 40$	$\times$	93.7 ± 1.66
	✓	97.04 ± 0.98
$k = 80$	$\times$	97.38 ± 1.48
	✓	98.09 ± 0.77

In this experiment, we have also considered larger values of k such as 20, 40, and 80. The trained models have been tested using all the samples from the meta-testing set. After the domain adaptation, the models were already capable of distinguishing different kinds of leaf diseases. Hence, even though it never saw any tomato leaf samples, it could achieve high accuracy like 89.06%, 92.46%, 94.07% respectively for the 5-shot, 10-shot, and 15-shot classification task. The results were even more promising as the number of samples increased. We achieved 98.09% accuracy with only 80 samples per class which is almost $20\times$ less data dependency compared to the other works. One obvious observation was that, as the value of k goes higher, the significance of Domain adaptation becomes lower. Finally, to find the maximum capacity of the pipeline, we provided the model with all the 14531 training samples and evaluated it with the full test set; which achieved a remarkable accuracy of 99.31 ± 0.12. Our conclusion from this experiment is that domain adaptation is such a powerful tool that it has enabled the model to allow $20\times$ less data compromising only 1.22% accuracy.

4.3.5 Single Domain vs Cross Domain Experiments

To conduct the robustness of our proposed pipeline in different circumstances, we conducted several experiments in single-domain and cross-domain environments. In the first experiment, we used the dataset of the few-shot task proposed by Argueso *et*

Table 4.7: Performance Comparison on PlantVillage Dataset

Approach	$k = 1$	$k = 10$	$k = 15$	$k = 80$
Argüeso *et al*. [54]	55.5	77	80	90
Wang *et al*. [73]	63.8	×	91.3	96
Ours	$\mathbf{74.7 \pm 3.12}$	$\mathbf{99.73 \pm 0.24}$	$\mathbf{99.92 \pm 0.09}$	$\mathbf{99.9 \pm 0.07}$

Table 4.8: Single-Domain and Cross-Domain Experiments

k-shot	**Approach**	Single-Mixed Domain	Cross-Domain 1	Cross-Domain 2
$K = 1$	Li *et al*. [62]	81.1	72.1	44.1
	Nuthalapati *et al*. [84]	84.1	79.2	52
	Ours	$\mathbf{95.56 \pm 0.91}$	$\mathbf{90.29 \pm 0.91}$	$\mathbf{90.05 \pm 1.06}$
$K = 5$	Li *et al*. [62]	87	84.9	53.1
	Nuthalapati *et al*. [84]	91.2	93.7	66.5
	Ours	$\mathbf{99.97 \pm 0.03}$	$\mathbf{99.8 \pm 0.1}$	$\mathbf{99.51 \pm 0.14}$
$K = 10$	Li *et al*. [62]	90.4	87.1	55.8
	Nuthalapati *et al*. [84]	92.9	95.5	71.6
	Ours	$\mathbf{99.97 \pm 0.03}$	$\mathbf{99.89 \pm 0.05}$	$\mathbf{99.41 \pm 0.28}$

al. [54]. In this task, the target classes contained samples from apple, blueberry, and cherry leaves. The experiments were conducted on $Q = 50$ for 20 randomly sampled tasks. We applied the model under similar experimental setups and it significantly outperformed the state-of-the-art. The results are mentioned in Table 4.7.

Afterward, the robustness of the pipeline was shown with the single-domain and cross-domain tasks as proposed in [62]. For this task, the Plants & Pest dataset has been utilized. This experiment has been conducted keeping three tasks in mind named as, single-mixed domain, cross-domain1, and cross-domain2. In the single-mixed domain experiment, the source class contained samples of both leaves and pests. Hence, although the target classes contained other kinds of leaves or pests, the classification task was simpler. In the cross-domain1 experiment, the pest classes were chosen as the source class and the plant classes as the target class. The cross-domain 2 experiment was the other way around. The merit of these experiments is that, if the model can learn leaves after only being trained on pest samples, it signifies that it has learned very good feature representation. Regardless of the single or cross-domain tasks, the pipeline has outperformed the works of [62, 73] by a good margin. This shows the generalizing ability of the model under different circumstances with just a few shots.

Chapter 5

Conclusion

5.1 Summary

Fast and accurate recognition of leaf diseases can go a long way to meeting the ever-increasing demand in food production. In this regard, we have proposed a lightweight deep neural network by combining a fine-tuned pretrained model and a classifier network. The utilization of the adaptive contrast enhancement technique has eliminated the illumination problem persistent in the dataset. Runtime data augmentation techniques have been applied to address the class imbalance issue while avoiding data leakage. All these components of the pipeline enabled the model to focus on the disease spots and extract high-level features leading to an accuracy of 99.30%. We achieved this performance with significantly smaller model size and FLOPs count compared to the state-of-the-art models. The nearest model producing a similar level performance required $2.4\times$ heavier model size and $2.45\times$ additional FLOPs count requirements. This makes the proposed pipeline a suitable choice for building applications for low-end devices.

Furthermore, the scarcity of quality datasets has brought the interest of the researchers to propose systems that can produce high accuracy even with a limited number of samples. In this regard, we incorporated the concept of few-shot learning to propose a pipeline for classifying leaf diseases with very limited data. The challenge of building a good feature representation has been tackled by utilizing a CNN-based feature extraction block consisting of different state-of-the-art architectures. Each component of the feature extractor has provided its observation on the provided sample which has been then combined and passed to a classifier block for effective prediction. The concept of domain adaptation has improved the feature extractors by providing domain-specific knowledge so that it performs well for unseen classes. After thoroughly analyzing the pipeline, we have achieved promising accuracy of 89.06%, 92.46%, and 94.07% respectively for 5, 10, and 15-shot classification, which is excep-

tionally higher than using other types of FSL algorithms. We achieved an accuracy of 98.09% providing only 80 samples per class, which is a 94.5% reduction in the requirement of training data. We also investigated the robustness of the model under mixed-domain and cross-domain tasks, where it produced state-of-the-art performance compared to the existing methods.

5.2 Future Works

Despite that, there are several scopes for improvement. One of the limitations of the PlantVillage dataset is that the samples are taken in laboratory conditions. Further experiments can be performed with leaf images having varying backgrounds taken from the field. Such images might contain occlusion and background clutter. Advanced segmentation techniques can be taken into account to locate the infected regions before classification. Moreover, only a single disease can be found in each of the samples used in our experiment. Identifying multiple diseases within a single leaf will be another challenging task to solve. The classification goal can also include detection of the severity of infection on leaves, which intelligent systems can utilize to decide the amount of pesticide to be used. In the few-shot learning task, an attention mechanism could have been employed on the concatenated features. The effectiveness of the proposed pipelines can also be explored in other domains. Finally, we have proposed separate solutions to tackle the data and resource constraints. Proposing lightweight few-shot learning algorithms along with ensuring state-of-the-art performance can be an interesting research challenge to consider in future endeavors.

REFERENCES

[1] S. Panno, S. Davino, A. G. Caruso, S. Bertacca, A. Crnogorac, A. Mandić, E. Noris, and S. Matić, "A Review of the Most Common and Economically Important Diseases That Undermine the Cultivation of Tomato Crop in the Mediterranean Basin," *Agronomy*, vol. 11, no. 11, 2021. [Online]. Available: https://www.mdpi.com/2073-4395/11/11/2188

[2] L. Li, S. Zhang, and B. Wang, "Plant Disease Detection and Classification by Deep Learning—A Review," *IEEE Access*, vol. 9, pp. 56 683–56 698, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9399342

[3] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, "Machine Learning in Agriculture: A Review," *Sensors*, vol. 18, no. 8, p. 2674, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/8/2674

[4] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0168169917308803

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269. [Online]. Available: https://ieeexplore.ieee.org/document/8099726

[7] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri

and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 6105–6114. [Online]. Available: http://proceedings.mlr.press/v97/tan19a.html

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/document/7298594/

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv - Computing Research Repository*, 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520. [Online]. Available: https://ieeexplore.ieee.org/document/8578572

[11] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/html/Zoph_Learning_Transferable_Architectures_CVPR_2018_paper.html

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. [Online]. Available: https://ieeexplore.ieee.org/document/7780459

[13] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size," *arXiv - Computing Research Repository*, 2016. [Online]. Available: http://arxiv.org/abs/1412.6980

[14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[15] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264. [Online]. Available: https://www.igi-global.com/chapter/transfer-learning/36988

[16] D. P. Hughes and M. Salathé, "An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing," *arXiv - Computing Research Repository*, 2015. [Online]. Available: http://arxiv.org/abs/1511.08060

[17] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detectio," *Frontiers in Plant Science*, vol. 7, p. 1419, 2016. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fpls.2016.01419/full

[18] Z. ur Rehman, M. A. Khan, F. Ahmed, R. Damasevicius, S. R. Naqvi, M. W. Nisar, and K. Javed, "Recognizing apple leaf diseases using a novel parallel real-time processing framework based on MASK RCNN and transfer learning: An application for smart agriculture," *IET Image Process.*, vol. 15, no. 10, pp. 2157–2168, 2021. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/ipr2.12183

[19] O. O. Abayomi-Alli, R. Damaševičius, S. Misra, and R. Maskeliūnas, "Cassava disease recognition from low-quality images using enhanced data augmentation model and deep learning," *Expert Systems*, vol. 38, no. 7, p. e12746, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.12746

[20] A. Waheed, M. Goyal, D. Gupta, A. Khanna, A. E. Hassanien, and H. M. Pandey, "An optimized dense convolutional neural network model for disease recognition and classification in corn leaf," *Computers and Electronics in Agriculture*, vol. 175, p. 105456, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169920302180

[21] S. Zhang, S. Zhang, C. Zhang, X. Wang, and Y. Shi, "Cucumber leaf disease identification with global pooling dilated convolutional neural network," *Computers and Electronics in Agriculture*, vol. 162, pp. 422–430, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169918317976

[22] B. Liu, C. Tan, S. Li, J. He, and H. Wang, "A Data Augmentation Method Based on Generative Adversarial Networks for Grape Leaf Disease Identification," *IEEE Access*, vol. 8, pp. 102 188–102 198, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9104723/

[23] X. Zhang, Y. Qiao, F. Meng, C. Fan, and M. Zhang, "Identification of Maize Leaf Diseases Using Improved Deep Convolutional Neural

Networks," *IEEE Access*, vol. 6, pp. 30 370–30 377, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8374024

[24] U. P. Singh, S. S. Chouhan, S. Jain, and S. Jain, "Multilayer Convolution Neural Network for the Classification of Mango Leaves Infected by Anthracnose Disease," *IEEE Access*, vol. 7, pp. 43 721–43 729, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8675730

[25] Y. Lu, S. Yi, N. Zeng, Y. Liu, and Y. Zhang, "Identification of rice diseases using deep convolutional neural networks," *Neurocomputing*, vol. 267, pp. 378–384, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217311384

[26] U. Mokhtar, N. El-Bendary, A. E. Hassenian, E. Emary, M. A. Mahmoud, H. A. Hefny, and M. F. Tolba, "SVM-Based Detection of Tomato Leaves Diseases," in *Intelligent Systems'2014 - Proceedings of the 7th IEEE International Conference Intelligent Systems IS'2014, September 24-26, 2014, Warsaw, Poland, Volume 2: Tools, Architectures, Systems, Applications*, ser. Advances in Intelligent Systems and Computing, D. P. Filev, J. Jablkowski, J. Kacprzyk, M. Krawczak, I. Popchev, L. Rutkowski, V. S. Sgurev, E. Sotirova, P. Szynkarczyk, and S. Zadrozny, Eds. Springer, Cham, 2015, vol. 323, pp. 641–652. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-11310-4_55

[27] U. Mokhtar, M. A. Ali, A. E. Hassanien, and H. Hefny, "Identifying Two of Tomatoes Leaf Viruses Using Support Vector Machine," in *Information Systems Design and Intelligent Applications*, ser. Advances in Intelligent Systems and Computing, J. Mandal, S. Satapathy, S. Kumar, P. Sarkar, and A. Mukhopadhyay, Eds. Springer, New Delhi, 2015, vol. 339, pp. 771–782. [Online]. Available: https://link.springer.com/chapter/10.1007/978-81-322-2250-7_77

[28] U. Mokhtar, M. A. Ali, A. E. Hassenian, and H. Hefny, "Tomato leaves diseases detection approach based on Support Vector Machines," in *2015 11th International Computer Engineering Conference (ICENCO)*. IEEE, 2015, pp. 246–250. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7416356

[29] A. E. Hassanien, T. Gaber, U. Mokhtar, and H. Hefny, "An improved moth flame optimization algorithm based on rough sets for tomato diseases detection," *Computers and Electronics in Agriculture*, vol. 136, pp. 86–96,

2017. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0168169916308225

[30] H. Sabrol and K. Satish, "Tomato plant disease classification in digital images using classification tree," in *2016 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2016, pp. 1242–1246. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7754351

[31] L. C. Ngugi, M. Abelwahab, and M. Abo-Zahhad, "Tomato leaf segmentation algorithms for mobile phone applications using deep learning," *Computers and Electronics in Agriculture*, vol. 178, p. 105788, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169920306529

[32] J. Liu and X. Wang, "Tomato Diseases and Pests Detection Based on Improved Yolo V3 Convolutional Neural Network," *Frontiers in Plant Science*, vol. 11, p. 898, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fpls.2020.00898

[33] Y. Zhang, C. Song, and D. Zhang, "Deep Learning-Based Object Detection Improvement for Tomato Disease," *IEEE Access*, vol. 8, pp. 56 607–56 614, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9044330

[34] A. Fuentes, S. Yoon, S. C. Kim, and D. S. Park, "A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition," *Sensors*, vol. 17, no. 9, p. 2022, 2017. [Online]. Available: https://www.mdpi.com/1424-8220/17/9/2022

[35] J. Liu and X. Wang, "Early recognition of tomato gray leaf spot disease based on MobileNetv2-YOLOv3 model," *Plant Methods*, vol. 16, pp. 1–16, 2020. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/32523613

[36] M. Brahimi, K. Boukhalfa, and A. Moussaoui, "Deep Learning for Tomato Diseases: Classification and Symptoms Visualization," *Applied Artificial Intelligence*, vol. 31, no. 4, pp. 299–315, 2017. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/08839514.2017.1315516

[37] A. Fuentes, D. H. Im, S. Yoon, and D. S. Park, "Spectral Analysis of CNN for Tomato Disease Identification," in *Artificial Intelligence and Soft Computing*, ser. Lecture Notes in Computer Science, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds., vol. 10245. Springer International Publishing, 2017, pp. 40–51. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-59063-9_4

[38] G. Yang, G. Chen, Y. He, Z. Yan, Y. Guo, and J. Ding, "Self-Supervised Collaborative Multi-Network for Fine-Grained Visual Categorization of Tomato Diseases," *IEEE Access*, vol. 8, pp. 211 912–211 923, 2020.

[39] V. Maeda-Gutierrez, C. E. Galvan-Tejada, L. A. Zanella-Calzada, J. M. Celaya-Padilla, J. I. Galván-Tejada, H. Gamboa-Rosales, H. Luna-Garcia, R. Magallanes-Quintanar, C. A. Guerrero Mendez, and C. A. Olvera-Olvera, "Comparison of Convolutional Neural Network Architectures for Classification of Tomato Plant Diseases," *Applied Sciences*, vol. 10, no. 4, p. 1245, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/4/1245

[40] Q. Wu, Y. Chen, and J. Meng, "DCGAN-Based Data Augmentation for Tomato Leaf Disease Identification," *IEEE Access*, vol. 8, pp. 98 716–98 728, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9099295

[41] A. K. Rangarajan, R. Purushothaman, and A. Ramesh, "Tomato crop disease classification using pre-trained deep learning algorithm," *Procedia Computer Science*, vol. 133, pp. 1040–1047, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918310159

[42] K. Zhang, Q. Wu, A. Liu, and X. Meng, "Can Deep Learning Identify Tomato Leaf Disease?" *Advances in Multimedia*, vol. 2018, 2018. [Online]. Available: https://www.hindawi.com/journals/am/2018/6710865/

[43] A. Abbas, S. Jain, M. Gour, and S. Vankudothu, "Tomato plant disease detection using transfer learning with C-GAN synthetic images," *Computers and Electronics in Agriculture*, vol. 187, p. 106279, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0168169921002969

[44] H. Durmuş, E. O. Güneş, and M. Kırcı, "Disease detection on the leaves of the tomato plants by using deep learning," in *2017 6th International Conference on Agro-Geoinformatics*. IEEE, 2017, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8047016

[45] P. Tm, A. Pranathi, K. SaiAshritha, N. B. Chittaragi, and S. G. Koolagudi, "Tomato Leaf Disease Detection Using Convolutional Neural Networks," in *2018 Eleventh International Conference on Contemporary Computing (IC3)*. IEEE, 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8530532/

[46] P. Bir, R. Kumar, and G. Singh, "Transfer Learning based Tomato Leaf Disease Detection for mobile applications," in *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*. IEEE, 2020,

pp. 34–39. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9231174

[47] J. Yang, X. Guo, Y. Li, F. Marinello, S. Ercisli, and Z. Zhang, "A survey of few-shot learning in smart agriculture: developments, applications, and challenges," *Plant Methods*, vol. 18, no. 1, p. 28, Mar 2022. [Online]. Available: https://doi.org/10.1186/s13007-022-00866-2

[48] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proceedings of the 32nd Internation Conference on Machine Learning*. JMLR, 2015. [Online]. Available: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

[49] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf

[50] A. Graves, *Long Short-Term Memory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 37–45. [Online]. Available: https://doi.org/10.1007/978-3-642-24797-2_4

[51] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf

[52] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[53] X. Zhang, F. Sung, Y. Qiang, Y. Yang, and T. M. Hospedales, "Deep comparison: Relation columns for few-shot learning," *CoRR*, vol. abs/1811.07100, 2018. [Online]. Available: http://arxiv.org/abs/1811.07100

[54] D. Argüeso, A. Picon, U. Irusta, A. Medela, M. G. San-Emeterio, A. Bereciartua, and A. Alvarez-Gila, "Few-shot learning approach for plant disease classification using images taken in the field," *Computers and Electronics in Agriculture*, vol. 175, p. 105542, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169920302544

[55] B. Wang and D. Wang, "Plant leaves classification: A few-shot learning method based on siamese network," *IEEE Access*, vol. 7, pp. 151 754–151 763, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8869770/

[56] S. G. Wu, F. S. Bao, E. Y. Xu, Y.-X. Wang, Y.-F. Chang, and Q.-L. Xiang, "A leaf recognition algorithm for plant classification using probabilistic neural network," in *2007 IEEE international symposium on signal processing and information technology*. IEEE, 2007, pp. 11–16. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4458016/

[57] O. Söderkvist, "Computer vision classification of leaves from swedish trees," 2001.

[58] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. B. Soares, "Leafsnap: A computer vision system for automatic plant species identification," in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 502–516.

[59] I. Egusquiza, A. Picon, U. Irusta, A. Bereciartua-Perez, T. Eggers, C. Klukas, E. Aramendi, and R. Navarra-Mestre, "Analysis of few-shot techniques for fungal plant disease classification and evaluation of clustering capabilities over real datasets," *Frontiers in Plant Science*, vol. 13, 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fpls.2022.813237

[60] S. Jadon, "Ssm-net for plants disease identification in low data regime," in *2020 IEEE / ITU International Conference on Artificial Intelligence for Good (AI4G)*, 2020, pp. 158–163. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9311073

[61] S. Mohanty, "Minileaves challenge," https://www.aicrowd.com/challenges/aicrowd-blitz-may-2020/problems/minileaves#links, 2020, accessed: 2022-07-02.

[62] Y. Li and J. Yang, "Meta-learning baselines and database for few-shot classification in agriculture," *Computers and Electronics in Agriculture*, vol. 182, p. 106055, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169921000739

[63] L. M. Tassis and R. A. Krohling, "Few-shot learning for biotic stress classification of coffee leaves," *Artificial Intelligence in Agriculture*, vol. 6, pp. 55–67, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2589721722000046

[64] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Schroff_FaceNet_A_Unified_2015_CVPR_paper.html

[65] Y. Li and J. Yang, "Few-shot cotton pest recognition and terminal realization," *Computers and Electronics in Agriculture*, vol. 169, p. 105240, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919323038

[66] G. Hu, H. Wu, Y. Zhang, and M. Wan, "A low shot learning method for tea leaf's disease identification," *Computers and Electronics in Agriculture*, vol. 163, p. 104852, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919300407

[67] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[68] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[69] S. Nesteruk, D. Shadrin, and M. Pukalchik, "Image augmentation for multitask few-shot learning: Agricultural domain use-case," *CoRR*, vol. abs/2102.12295, 2021. [Online]. Available: https://arxiv.org/abs/2102.12295

[70] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1126–1135. [Online]. Available: https://proceedings.mlr.press/v70/finn17a.html

[71] M. A. Jamal and G.-J. Qi, "Task agnostic meta-learning for few-shot learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[72] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *CoRR*, vol. abs/1803.02999, 2018. [Online]. Available: http://arxiv.org/abs/1803.02999

[73] Y. Wang and S. Wang, "Imal: An improved meta-learning approach for few-shot classification of plant diseases," in *2021 IEEE 21st International Conference on*

Bioinformatics and Bioengineering (BIBE), 2021, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9635575

[74] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 499–515.

[75] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[76] Y.-X. Wang, R. Girshick, M. Hebert, and B. Hariharan, "Low-shot learning from imaginary data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/html/Wang_Low-Shot_Learning_From_CVPR_2018_paper.html

[77] B. Oreshkin, P. Rodríguez López, and A. Lacoste, "Tadam: Task dependent adaptive metric for improved few-shot learning," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/66808e327dc79d135ba18e051673d906-Paper.pdf

[78] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, "Meta-learning with latent embedding optimization," *CoRR*, vol. abs/1807.05960, 2018. [Online]. Available: http://arxiv.org/abs/1807.05960

[79] W. Chen, Y. Liu, Z. Kira, Y. F. Wang, and J. Huang, "A closer look at few-shot classification," *CoRR*, vol. abs/1904.04232, 2019. [Online]. Available: http://arxiv.org/abs/1904.04232

[80] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 491–507.

[81] Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola, "Rethinking few-shot image classification: A good embedding is all you need?" in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 266–282.

[82] N. Dvornik, C. Schmid, and J. Mairal, "Selecting relevant features from a multi-domain representation for few-shot classification," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 769–786.

[83] ——, "Diversity with cooperation: Ensemble methods for few-shot classification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[84] S. V. Nuthalapati and A. Tunga, "Multi-domain few-shot learning and dataset for agricultural applications," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, vol. abs/2109.09952, October 2021, pp. 1399–1408. [Online]. Available: https://arxiv.org/abs/2109.09952

[85] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. [Online]. Available: https://ieeexplore.ieee.org/document/7780459

[86] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[87] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha, "Few-shot learning via embedding adaptation with set-to-set functions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[88] P. Galeano, E. Joseph, and R. E. Lillo, "The mahalanobis distance for functional data with applications to classification," *Technometrics*, vol. 57, no. 2, pp. 281–291, 2015. [Online]. Available: https://doi.org/10.1080/00401706.2014.902774

[89] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10302

[90] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya, "A survey on addressing high-class imbalance in big data," *Journal of Big Data*, vol. 5, no. 1, pp. 1–30, 2018. [Online]. Available: https://link.springer.com/article/10.1186/s40537-018-0151-6

[91] Y. Li, H. Lu, J. Li, X. Li, Y. Li, and S. Serikawa, "Underwater image de-scattering and classification by deep neural network," *Computers and Electronics in Agriculture*, vol. 54, pp. 68–77, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0045790616302075

[92] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer Vision, Graphics, and Image Processing*, vol. 39, no. 3, pp. 355–368, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0734189X8780186X

[93] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in Data Mining: Formulation, Detection, and Avoidance," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 4, pp. 1–21, 2012.

[94] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *32nd International Conference on Machine Learning, ICML*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[95] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html

[96] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2670313

[97] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts Ave: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[98] S. Khirirat, H. R. Feyzmahdavian, and M. Johansson, "Mini-batch gradient descent: Faster convergence under data sparsity," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 2880–2887. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8264077

[99] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[100] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int. J. of Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015. [Online]. Available: https://link.springer.com/article/10.1007/s11263-015-0816-y

[101] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 618–626. [Online]. Available: https://ieeexplore.ieee.org/document/8237336

[102] A. Chowdhury, M. Jiang, S. Chaudhuri, and C. Jermaine, "Few-shot image classification: Just use a library of pre-trained feature extractors and a simple classifier," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 9445–9454. [Online]. Available: https://openaccess.thecvf.com/content/ICCV2021/html/Chowdhury_Few-Shot_Image_Classification_Just_Use_a_Library_of_Pre-Trained_Feature_ICCV_2021_paper.html

[103] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 5353–5360. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7299173

[104] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/279181

[105] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[106] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[107] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

List of Publications

Journal

1. **S. Ahmed**, M. B. Hasan, T. Ahmed, R. K. Sony and M. H. Kabir, "Less is More: Lighter and Faster Deep Neural Architecture for Tomato Leaf Disease Classification". in *IEEE Access*, vol. 10, pp. 68868 - 68884, 2022. doi: 10.1109/ACCESS.2022.3187203