Deep Learning for Plant Disease Classification

Srishti Kumar

CONCEPT OF OPERATIONS

Revision – 0.2

April 26, 2020

CONCEPT OF OPERATIONS

FOR

Deep Learning for Plant Disease Classification

TEAM <50>

APPROVED BY:

_____

Srishti Kumar Date

_____

John Lusher II, P.E. Date

_____

Abishalini Sivaraman Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	9/16/19	Srishti Kumar		First Submission
0.1	12/1/19	Srishti Kumar		End of Semester Submission
0.2	4/26/20	Srishti Kumar		End of Semester Submission

CONOPS Table of Contents

List of Figures

List of Tables

1. Executive Summary

Deep learning using convolutional neural networks (CNNs) has become increasingly successful in plant disease diagnosis, and consequently offers new promise in the ongoing battle against crop disease. While many studies continue to demonstrate its continued success in plant disease diagnosis, the complexity of the CNN architecture remains largely unexplored. The aim of this project is to shed more light onto the architectures of the CNNs for plant disease classification specifically, ensuring its reliability and authenticity by human intervention. These results can then be utilized to further improve the prediction accuracy of deep models by optimizing the underlying architecture. Four models are trained and tested on the PlantVillage dataset: InceptionV3, AlexNet, GoogLeNet, and ResNet50. To further improve the CNN model, attention modules are applied to the baseline CNN model. Additionally, several visualization techniques (feature visualization) are explored. These visualizations will help to uncover the CNN "black box," and shed light on features learned by the CNNs. Lastly, two user interfaces (mobile application and interactive graphical user interface) designed to facilitate system usability are discussed.

2. Introduction

One of the most significant threats to food security is plant disease. Historically, plant disease has taken a large negative toll on both crop yield and quality. Identification and diagnosis of these diseases has always been a significant problem. Past efforts involved chemical analysis (for pathogen detection), optical technologies (plant monitoring), and human visual inspection. However, these methods are often time-consuming and expensive. Deep learning, with advancing high-throughput image acquisition capabilities, offers a more accurate and practical solution for plant disease diagnosis. First, it only needs images of plants (leaves, fruits, etc.), thereby bypassing the need for laboratory analysis of plant samples. Next, it offers more versatility, as it can deal with various lighting conditions, resolution, and object size. Lastly, it offers a more cost-efficient method as images can be taken with even a cell phone camera, thus eliminating the need for expensive equipment or technology.

2.1. Background

The plant disease classification system proposed in this document is meant to provide a more viable method for image classification. This system will use a CNN for classification, which will be trained on thousands of plant images. This system aims to quickly and accurately classify a plant image with low overhead.

2.2. Overview

Among the many models used in deep learning, CNNs are used extensively in computer vision and image analysis, and have been used for plant image analysis. However, the complexity of the model has caused it to be perceived as a "black box[1]," thus raising the concerns of interpretability of the learned model. By observing and analyzing intermediate outputs during the classification, I will be able to gain more insight into the particular features learned by the CNN. Careful analysis of these observations will allow me to then tweak the model appropriately to improve its accuracy in plant disease identification and diagnosis. Those observations will then be used to extend the model to classify a new plant disease.

2.3. Referenced Documents and Standards

Document Title	Revision/Release Date	Publisher
Deep Learning with Python	2018	Manning Publications Co.
Keras Documentation	2.3.0	Keras Team
Python Library Reference	3.6	Python Software Foundation
Caffe Documentation	1.0	UC Berkeley
How Convolutional Neural Networks Diagnose Plant Disease	March 26, 2019	Plant Phenomics
Using Deep Learning for Image-Based Plant Disease Detection	September 6, 2016	Frontiers

Table 1: Reference Documents and Standards

3. Operating Concept

3.1. Scope

The plant classification system proposed in this document is designed for field scientists looking for a practical method to detect diseased plants. With this in mind, a CNN-based classification system should increase productivity by eliminating the labor/resource/monetary costs incurred by current methods (chemical/visual inspection) disease detection systems.

3.2. Operational Description and Constraints

The plant image will first be converted into a tensor (an array containing the pixel values). From there, it is then fed into the CNN, which classifies the image (species and healthy/diseased).

3.3. System Description

The system is broken up into 2 main subsystems: Image Processing and Mobile Application.

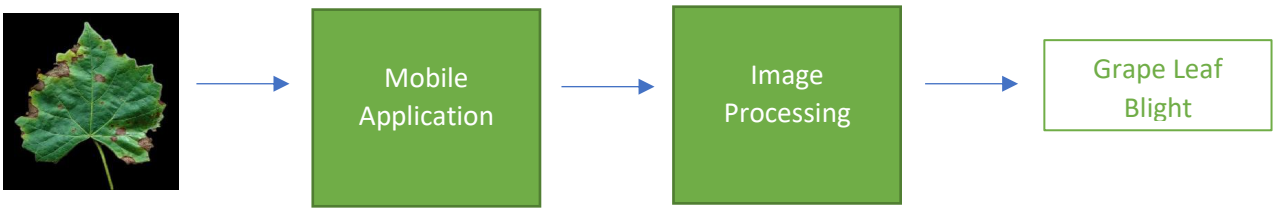


Figure 1: System Functional Block Diagram

Image Processing: The input to this subsystem is an image of a plant leaf. The system will then use a CNN to classify the plant image as healthy/diseased. The dataset used to train the model is the PlantVillage dataset, a public dataset containing over 54,000 plant images, divided into 38 plant species. All implementation and testing will be done on the Tamu HPRC clusters. Training of the model will be done on 2 different backends: Keras (a Python library using a Tensorflow backend) and Caffe (a deep learning framework developed at UC Berkeley.)

Mobile Application: This system will provide a user interface to the Image Processing subsystem in the form of an Android application. The CNN model developed in the Image Processing subsystem will be deployed to the mobile application, and stored as a protobuf file, which contains the model definition and weights. The user will take a picture of a (healthy/diseased) leaf using the mobile camera. This leaf will then be input to the CNN model subsystem, which will classify it as the appropriate species. This subsystem will give the project a real-world application as scientists working in the field will be able to easily classify plants.

3.4. Modes of Operations

With the current requirements of the project, there will be only 1 mode of operation where the users can have their plant image classified. While multiple CNN models will be trained, tested, and optimized during the model development, only 1 will be used by the system. If any additional

requirements related to modes of operation arise throughout the project development, this section shall be updated.

3.5. Users

The target users for this system are scientists, researchers, and small farmers. The user interface will be designed in such a way that no prior experience/understanding of image processing is required; the user simply has to upload a plant image to the application. At minimum, the user should have in his/her possession a mobile phone and a basic understanding of how to take/upload images from the mobile camera.

3.6. Support

A small manual detailing how to use the mobile application will be included in the code repository. However, the user interface will be designed to be simple enough that the use of a manual should not be required. While the image processing subsystem will be abstracted from the user, the CNN model will be heavily commented/documented, and will be accompanied by a README file.

4. Scenario(s)

4.1. Single Plant Leaf Classification

In this scenario, a single plant image is uploaded to the system. This plant image is classified as its appropriate species by the CNN, and the result is output to the user. The flow of data can be seen in more detail in Figure 3.

5. Analysis

5.1. Summary of Proposed Improvements

The proposed system offers many improvements over the status quo.

- Firstly, the only input required is an image of a plant leaf. This is a large improvement over the status quo as it eliminates the need for laborious chemical analysis/expensive laboratory equipment.

- Secondly, the model is not limited to a particular range of plant species. Provided sufficient data, it can be trained to classify and diagnose new types of plant disease.

- Lastly, the model is a more practical approach to plant disease classification. The model can be deployed to a mobile phone, allowing users to classify an image almost instantly. This provides a great benefit to field scientists, as the only requirement needed is a pocket-sized mobile phone.

5.2. Disadvantages and Limitations

The proposed system has several disadvantages.

- Firstly, the model has not been tested on non-leaf diseases. Disease is not strictly limited to a plant leaf, and may occur on the underside of a leaf/plant stem. However, the model has only been trained to detect diseases occurring on the surface of a leaf.

- Secondly, the model can only detect the presence of lesions/discoloration on the plant leaf. Disease rampant within the plant cell, but not observable on the surface of a leaf, is undetectable by the model.

- Thirdly, the model is only able to classify isolated leaf images. The PlantVillage dataset used to train the model contains ideal plant images – images against a bare background. However, in the real world plant leaves are unlikely to be photographed in isolation, and will likely have other plant leaves/stems in the background. Filtering these background elements will require more training.

- Lastly, the model only works for a given number of diseases (currently 38). This limits its usefulness, as it is only applicable for a certain range of plant species.

While the above disadvantages may appear to be serious limitations of the model, increasing the dataset diversity to include stems/non-ideal images/more plant species will likely solve the majority of these issues. However, data collection is a challenging task by itself, and is not the focus of this research, although different datasets will be experimented on.

5.3. Alternatives

There are a variety of alternate CNN backends that are usable, such as CNTK and Theano. However, the Keras-Tensorflow platform was chosen for its simple and well-documented API, in addition to its

popularity in practice. The same reasoning applies to the choice of Android Studio for the mobile application.

6. Ethical Considerations

There are several ethical dilemmas with regards to this project, but the primary one is that of misclassification.

Neural network models are a powerful tool for multiple types of problems, such as classification and regression problems. However, serious issues can result from subtle biases within the network. In the past, models such as the Google captioning system have infamously misclassified humans as animals. While the repercussions would not be so severe in the domain of plant disease diagnosis, they would still be problematic. Consider the scenario where the model consistently misclassifies a potato crop diseased with late blight as healthy. This could have serious repercussions for a farmer who may be led to believe that his crop is healthy (when actually diseased) and not take preventive action to eliminate the disease. From a data scientist perspective, the problem of misclassification is very challenging to debug, as the reasoning behind the choices made by the neural network is often ambiguous and complicated.

A more practical solution to the above problem is to increase the diversity of the dataset. Diversifying data in terms of leaf quality, background noise, and image quality helps to reduce biases in the model – which would otherwise highly influence the classification result. While multiple efforts were made to collect additional data during the spring semester, I was unable to collect my own datasets. Instead, I used the next best option available – data augmentation. This technique seeks to diversify the existing dataset by rotating, shifting, and flipping images.

While data augmentation is one possible solution, another is for users to be aware of the scope and limitations of the model, and let that influence their decisions accordingly. Due to the controlled leaf quality of the PlantVillage dataset, the model is likely to yield lower accuracy on images taken straight from the field, and will perform better on individual images. Moreover, it is to be noted that the model developed is not a replacement for existing tools and technologies (such as laboratory testing), but rather a *supplement.* While the model provides a simple disease diagnosis interface, samples should still undergo more thorough testing in traditional laboratory settings for full accuracy.

Deep Learning for Plant Disease Classification

Srishti Kumar

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 0.1

December 4, 2019

FUNCTIONAL SYSTEM REQUIREMENTS

FOR

Deep Learning for Plant Disease Classification

PREPARED BY:

_____

Srishti Kumar Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	9/30/19	Srishti Kumar		Initial Submission
0.1	12/2/19	Srishti Kumar		End of Semester Submission

FSR Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Purpose and Scope

This document gives an overview of the plant classification system that is being developed, in addition to defining the requirements of the system. The system is divided into 2 subsystems: Image Processing subsystem, and the Mobile Application subsystem. This system breakdown is depicted in Table 2.

This specification defines the technical requirements for the development items and support subsystems delivered to the client for the project. The below Table shows a representative integration of the project in the proposed CONOPS. The verification requirements for the project are contained in a separate Verification and Validation Plan.

The following definitions differentiate between requirements and other statements.

Shall:	This is the only verb used for the binding requirements.
Should/May:	These verbs are used for stating non-mandatory goals.
Will:	This verb is used for stating facts or declaration of purpose.

1.2. Responsibility and Change Authority

Srishti Kumar, the sole team member, holds the responsibility of ensuring that all the project requirements and subsystems are met. Any changes to performance requirements must be approved by the team leader.

Subsystem	Responsibility
Image Processing	Srishti Kumar
Mobile Application	Srishti Kumar

Table 1: Assigned Responsibilities

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Title	Revision/Release Date	Publisher
Deep Learning with Python	2018	Manning Publications Co.
Keras Documentation	2.3.0	Keras Team
Python Library Reference	3.6	Python Software Foundation
Caffe Documentation	1.0	UC Berkeley
How Convolutional Neural Networks Diagnose Plant Disease	March 26, 2019	Plant Phenomics
Residual Attention Network for Image Classification	April 2017	arXiv
Using Deep Learning for Image-Based Plant Disease Detection	September 6, 2016	Frontiers
Understanding and Coding A Resnet In Keras	January 4, 2019	Medium
Grad-CAM- Gradient-weighted Class Activation Mapping	March 29, 2019	Medium
Feature Visualization	August 28, 2019	Distill
TensorFlow Lite: ML for Mobile and Edge Devices	2020	Tensorflow
Student Notes: Convolutional Neural Networks (CNN) Introduction	2020	Belajar Pembelajaran Mesin Indonesia

Table 2: Applicable Documents

2.2. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

This section defines the minimum requirements that the development item(s) must meet. The requirements and constraints that apply to performance, design, interoperability, reliability, etc., of the system, are covered.

3.1. System Definition

The system is divided into 2 subsystems: Image Processing, and Mobile Phone Deployment. The Image Processing subsystem is responsible for processing the images. This is broken down into many sub-tasks. CNNs of different architectures (InceptionV3, AlexNet, GoogLeNet) are trained on input data and analyzed for accuracy. Training mechanisms include training from scratch, and transfer learning. Mobile Phone Deployment refers to porting the CNN onto a mobile phone application. This allows for increased usability, as a scientist/biologist wishing to classify a specimen can conveniently do so via a mobile application.

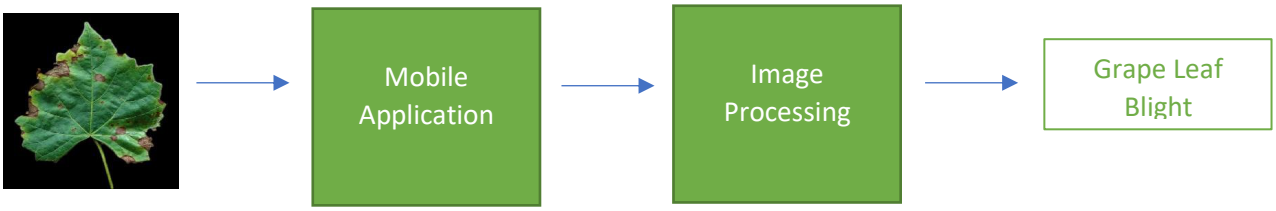


Figure 1: Plant Classification Functional Block Diagram

The above block diagram shows the general flow of information. Consider a user (scientist/farmer) who wish to analyze the state of their crops as healthy or diseased. The user inputs the plant image to a mobile application. The application then processes this image via a CNN model. After the CNN has finished processing the image, it outputs a list of possible classifications (plant species, healthy/diseased), along with a list of probabilities for those classifications.

CNN Architectures: 3 models will be analyzed: Alexnet, GoogLeNet, and InceptionV3. The following sections explain each architecture in great detail.

Convolutional Layer: A convolutional layer extracts features from the image by performing convolutions on the input image by a filter. The convolution layers learn complex features iteratively. The initial layers detect general features, such as edges. Successive layers combine these edges to detect shapes, and then image-specific objects such as a leaf.

Maxpooling Layer: A pooling layer downsamples the image generated from a convolution operation to reduce the number of parameters learned. In a maxpooling operation, the input image is probed by a max filter which retains only the max pixel value in every stride, thereby reducing the number of parameters. The max pixel value is normally characteristic of the most distinct feature (for example, an edge). A CNN architecture normally consists of alternating convolutional and maxpooling layers. This enables the model to extract the most elemental features, and use those as a basis for classification.

Classifier: The classifier is a project-specific layer that is added on to the end.

Fully Connected Layer: This layer is normally the last layer in a CNN network. It learns non-linear feature combinations by connecting every neuron from the previous layer to the next layer.

Dropout: This layer aims to fight overfitting (over-specialization of the model) by dropping a specified fraction of weights in a layer. As a result, the model is not over-specialized on the training data.

Flatten: This layer flattens the input (normally a matrix) into a 1D vector output. This vector is normally then fed into the softmax layer.

Softmax: This layer normalizes the outputs as a probability distribution between 0 and 1. In other words, the outputs sum to 1. For the purposes of this project, there are 38 outputs (corresponding to the 38 different classes in the PlantVillage dataset). For instance, if the 1^{st} probability in the output vector is 0.1, and the 17^{th} probability has the highest probability of 0.87, we can conclude that the input image belongs to Class#17 with probability 87%.

3.2. Characteristics

3.2.1. Input Data

Input Data: The CNN model(s) will be trained on the PlantVillage dataset, a public GitHub that consists of 54,306 images. These images include both healthy and diseased leaf images (26 diseases, 12 crop species) for a total of 38 labels. These images are further split into 3 categories: color, grayscale, and segmented (the leaf is segmented from its background and placed on a black background.) The image on the right displays all 38 plant images in the PlantVillage dataset. The labels are included at the bottom for clarification.

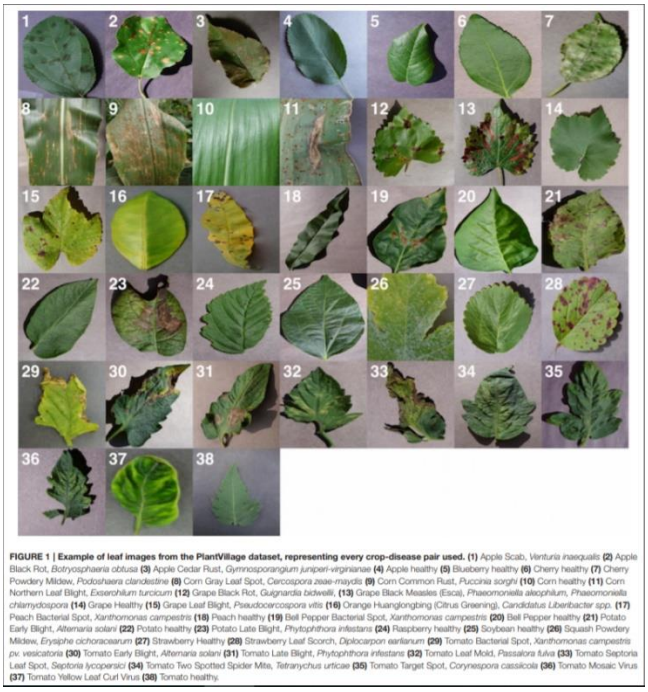


FIGURE 1 | Example of leaf images from the PlantVillage dataset, representing every crop-disease pair used. (1) Apple Scab, *Venturia inaequalis* **(2)** Apple Black Rot, *Botryosphaeria obtusa* **(3)** Apple Cedar Rust, *Gymnosporangium juniperi-virginianae* **(4)** Apple healthy **(5)** Blueberry healthy **(6)** Cherry healthy **(7)** Cherry Powdery Mildew, *Podosharea clandestine* **(8)** Corn Gray Leaf Spot, *Cercospora zeae-maydis* **(9)** Corn Common Rust, *Puccinia sorghi* **(10)** Corn healthy **(11)** Corn Northern Leaf Blight, *Exserohilum turcicum* **(12)** Grape Black Rot, *Guignardia bidwellii*, **(13)** Grape Black Measles (Esca), *Phaeomoniella aleophilum*, *Phaeomoniella chlamydospora* **(14)** Grape Healthy **(15)** Grape Leaf Blight, *Pseudocercospora vitis* **(16)** Orange Huanglongbing (Citrus Greening), *Candidatus Liberibacter spp.* **(17)** Peach Bacterial Spot, *Xanthomonas campestris* **(18)** Peach healthy **(19)** Bell Pepper Bacterial Spot, *Xanthomonas campestris* **(20)** Bell Pepper healthy **(21)** Potato Early Blight, *Alternaria solani* **(22)** Potato healthy **(23)** Potato Late Blight, *Phytophthora infestans* **(24)** Raspberry healthy **(25)** Soybean healthy **(26)** Squash Powdery Mildew, *Erysiphe cichoracearum* **(27)** Strawberry Healthy **(28)** Strawberry Leaf Scorch, *Diplocarpon earlianum* **(29)** Tomato Bacterial Spot, *Xanthomonas campestris pv. vesicatoria* **(30)** Tomato Early Blight, *Alternaria solani* **(31)** Tomato Late Blight, *Phytophthora infestans* **(32)** Tomato Leaf Mold, *Passalora fulva* **(33)** Tomato Septoria Leaf Spot, *Septoria lycopersici* **(34)** Tomato Two Spotted Spider Mite, *Tetranychus urticae* **(35)** Tomato Target Spot, *Corynespora cassiicola* **(36)** Tomato Mosaic Virus **(37)** Tomato Yellow Leaf Curl Virus **(38)** Tomato healthy.

Figure 2: PlantVillage Dataset

3.2.2. CNN Architectures

A variety of CNN architectures will be implemented, trained, and analyzed for accuracy. In particular, I will be analyzing the InceptionV3, AlexNet, and GoogLeNet architectures. The following sections explains each architecture in detail.

3.2.2.1. InceptionV3

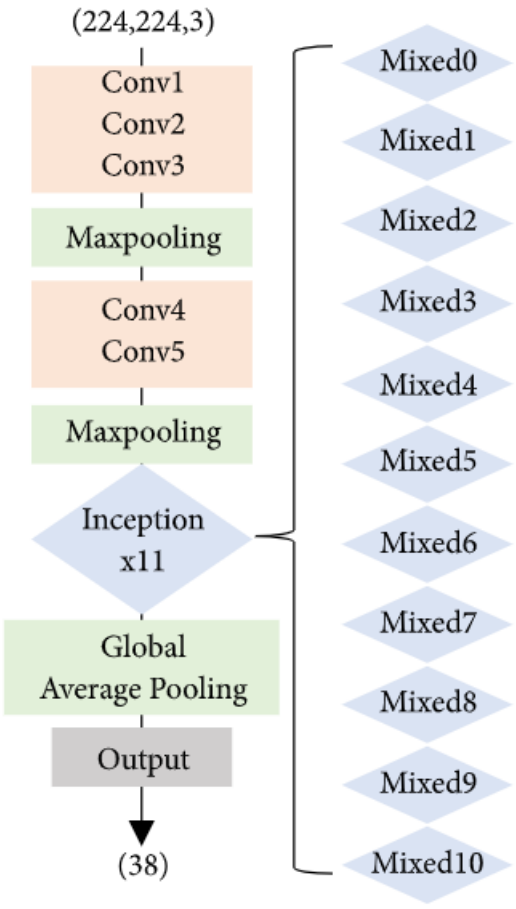


Figure 3: InceptionV3

Background: InceptionV3 is a widely-used image-recognition model. For the purposes of this project, the model was initialized with random weights, and trained from scratch. The block diagram of the InceptionV3 model is depicted to the left.

Input: image with size (224, 224, 3). The image width and height are 224 pixels, with 3 color channels to represent RGB images.

Output: A vector of size 38 containing the probability distribution of each class.

Description: The model alternates between Conv and Maxpooling layers. Every successive layer reduces the size of the input image, thus reducing the image to its most elemental image pattern. The model picks up on these patterns, and uses them to classify the image. The Mixed layers are indicative of deeper convolutional layers, where both the complexity of shape and color diversity is expected to increase.

The 2nd-to-last layer performs an Average Pooling instead of a Maxpooling. For feature extraction, maxpooling is valued over average pooling since only the most important features are desired to be kept. In contrast, average pooling computes the average of all pixels instead of the max, resulting in all values incorporated into the output vector. Consequently, average pooling is a more generalized computation. This is preferred for the output vector since classification, not feature extraction is the focus, and the pooling does not significantly alter the classification results.

The resultant vector is then fed into a softmax layer, which normalizes the vector into a probability distribution of values from 0 to 1.

Rationale: This is the layout of the model as specified in the Toda-Okura paper, and must be reproduced before further optimization.

3.2.2.2. AlexNet

The second CNN uses AlexNet as its underlying architecture, and the PlantVillage dataset as its input. AlexNet is comprised of 5 convolution layers, followed by 3 fully-connected layers, and finally a softmax layer. The convolution layers are followed by a normalization and pooling layer, similar to InceptionV3. As in InceptionV3, the output is a vector of size 38 containing the probability distribution.

Rationale: This is the layout of the model as specified in the Frontiers paper, and must be reproduced before further optimization.

3.2.2.3. GoogLeNet

The third CNN uses GoogLeNet as its underlying architecture, and the PlantVillage dataset as its input. In contrast to AlexNet, GoogLeNet is a much deeper architecture, and is comprised of 22 layers. It uses inception modules, which contain parallel 1x1, 3x3, 5x5 convolution layers along with a Maxpooling layer in parallel. Because the model is a parallel arrangement instead of a sequential one, it can simultaneously capture more features.

Rationale: This is the layout of the model as specified in the Frontiers paper, and must be reproduced before further optimization.

3.2.3. Training Mechanisms

To further assess their performance, the AlexNet models and GoogleNet models will be evaluated via 2 different training mechanisms:

1. Training from Scratch
2. Transfer Learning

Each of these training mechanisms is described in more detail below. After implementing both methods, the results will be compared and analyzed. In most cases, transfer learning proves to be more accurate as the neural network already contains weights corresponding to the correct classification, and does not have to learn the weights from only a single dataset.

3.2.3.1. Training from Scratch

In order to the train the AlexNet and GoogLeNet models from scratch, the models (not pretrained on any prior datasets) will be run on the PlantVillage datasets. Their resulting performance will then be recorded and analyzed.

Rationale: This is the 1st training mechanism performed as described in the Frontiers paper, and must be reproduced before further optimization.

3.2.3.1 Transfer Learning

Transfer Learning involves using pretrained models, and then adapting those models onto a new dataset. In this case, the AlexNet and GoogleNet models will be trained on the ImageNet dataset, and then evaluated on the PlantVillage dataset. The results of the Training from Scratch and Transfer Learning methods will then be compared and analyzed.

Rationale: This is the 2nd training mechanism performed as described in the Frontiers paper, and must be reproduced before further optimization.

7.3.2.3 Optimization

After the various CNNs have been evaluated and analyzed, they will be optimized, possibly even for a new dataset.

7.3.2.4 Mobile Phone Deployment

Once optimized, the CNN will be deployed onto an Android mobile application for usability purposes. The CNN will be deployed via TFlite, which compresses the TensorFlow model into a lighter, deployable version.

Appendix A: Acronyms and Abbreviations

API	Application Programming Interface
Caffe	An open-source, deep learning framework
CNN	Convolutional Neural Network
FSR	Functional System Requirement
ICD	Interface Control Document
GPU	Graphical Processing Unit

Table 3: Acronyms and Abbreviations

Appendix B: Definition of Terms

- Alexnet: A CNN architecture that contains 5 convolutional layers, followed by 3 fully-connected layers, and ending with a softmax layer. It has on the order of 60 million parameters.
- GoogLeNet: A type of CNN that contains 22 layers, and uses parallel convolutional layers. It has on the order of 5 million parameters.
- GitHub: A website that provides hosting for software development version control using Git.
- ImageNet: A large image database containing over 14 million images
- InceptionV3: A type of CNN that is 48 layers deep.
- Keras: A high-level neural networks API, written in Python and using Tensorflow/Theano for its backend.
- Python: Python is an interpreted, high-level programming language for general-purpose programming.

Deep Learning for Plant Disease Classification

Srishti Kumar

INTERFACE CONTROL DOCUMENT

REVISION – 0.1

December 4, 2019

INTERFACE CONTROL DOCUMENT

FOR

Deep Learning for Plant Disease Classification

PREPARED BY:

_____

Srishti Kumar Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher II, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	9/30/19	Srishti Kumar		Initial Submission
0.1	12/2/19	Srishti Kumar		End of Semester Submission

ICD Table of Contents

List of Figures

List of Tables

1. Overview

This document describes the interfaces between the different subsystems of the Image Classification system.

2. References and Definitions

2.1. References

Document Number	Revision/Release Date	Document Title
Python Library Reference	3.6	Python Software Foundation
Keras Library Reference	2.3.0	Keras Documentation
TensorFlow Documentation	1.10	Google Brain Team

Table 5: Reference Documents

2.2. Definitions

- Android Studio: The official integrated development environment for Google's Android operating system.
- Keras: A high-level neural networks API, written in Python and using Tensorflow/Theano for its backend.

Python: Python is an interpreted, high-level programming language for general-purpose programming.

3. Physical Interface

The project will be deployed to a mobile application. The CNN model will first be frozen (essentially unalterable), and converted to a TFLITE file. This contains the model definition and weights, and will then be deployed to the app. The app will be created via Android Studio, and usable by any Android device.

3.1. Frontend

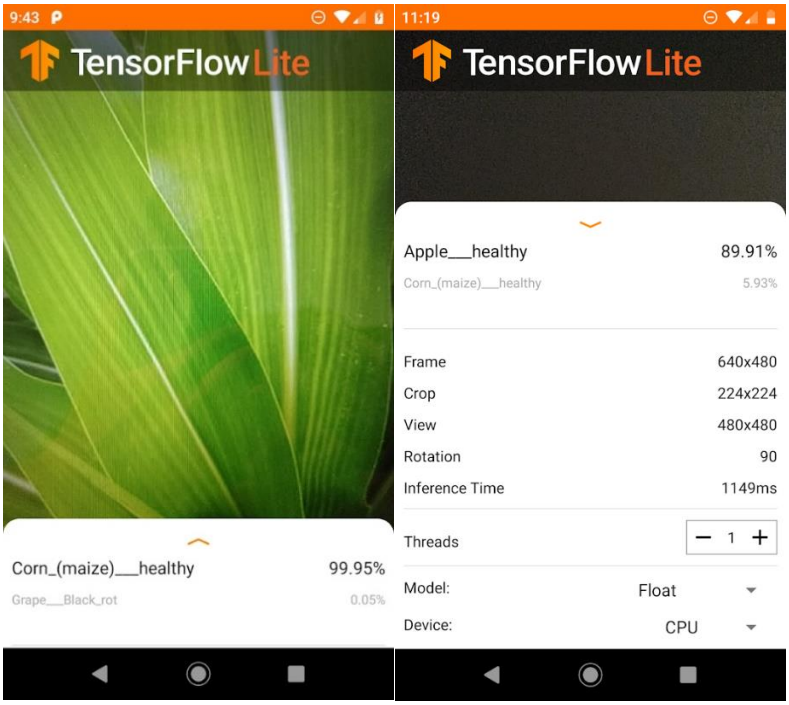


Figure 1: Mobile Application Interface

Figure#1: This interface displays a life camera preview, allowing classification to take place instantaneously. The classification results are displayed at the bottom. As seen in the image above, the plant leaf is classified as a Healthy Corn Leaf with 99.95% accuracy.

Interface#2: This interface displays the information about additional camera parameters. Details such as frame/crop size, thread number, and model specifications. While the number of threads can be altered by pressing the +/- buttons, changing the other parameters involves editing the codebase.

3.2. Backend

The user will not interface with the backend, which contains the CNN-based image processing. In effect, the image processing unit will appear as a black box to the user. This is done intentionally because:

1) The user is assumed to have no background in image processing
2) The CNN model should not be modified

As a result, the backend implementation is abstracted from the user, and the only interface needed is the frontend application.

4. Communications / Device Interface Protocols

Communication details are listed below.

4.1. Wireless Communications (WiFi)

No WiFi/internet connection is needed for communication. The CNN model is stored on the mobile application. The only required feature is the camera, which can be operated without a WiFi connection.

4.2. Host Device

The mobile application will be built with Android Studio. Hence, all Android-compatible devices are usable.

4.3. Camera Interface

The existing mobile camera will be used to take the pictures. No exterior camera will be used.

Deep Learning for Plant Disease Classification

Srishti Kumar

IMAGE PROCESSING SUBSYSTEM REPORT

REVISION – 404 Final

April 26, 2020

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	12/4/19	Srishti Kumar		End of Semester Submission
0.1	4/26/20	Srishti Kumar		End of Semester Submission

Image Processing Subsystem Table of Contents

List of Figures

1. Introduction

The purpose of this document will be to discuss the Image Processing subsystem of the overall system. This subsystem is the primary subsystem in this project. This report will cover essential features such as Dataset, Design, Results, and Validation.

2. Theory

The Image Processing subsystem is the primary subsytem in this project. This subsystem takes in as input an image of a plant leaf, and outputs a classification corresponding to one of the 38 species in the PlantVillage dataset. This high-level design is shown in Figure X. The model used to perform the classification is a convolutional neural network model (CNN).

Figure 1: Image Processing Subsystem Overview

3. Design

3.1 Dataset

The CNN model(s) were trained on the PlantVillage dataset, a public GitHub dataset that consists of 54,306 images. The images are divided into 38 categories: 14 crop species, and 26 diseases. Thus, each model is trained on both healthy and diseased leaf images. Figure X displays all 38 plant images in the PlantVillage dataset. The labels are included at the bottom for clarification. Each plant species is further split into 3 image categories: color, grayscale, and segmented. This image distribution can be seen in Figure X.

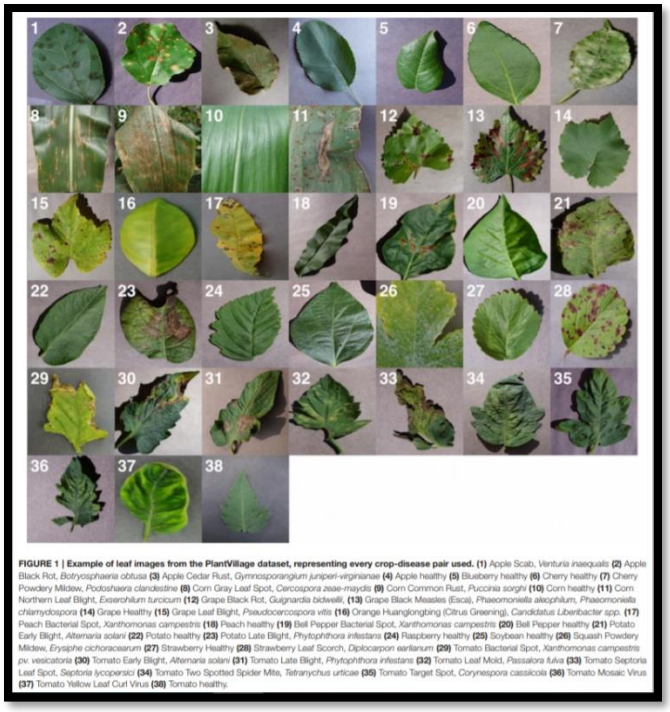


Figure 2: PlantVillage Dataset

Figure 3: Leaf Color Categories

3.2 Model Architecture

During the design portion of this subsystem, four state-of-the art CNNs were constructed and evaluated on the PlantVillage dataset: InceptionV3, AlexNet, GoogLeNet, and ResNet50. Images of size 224 x 224 were fed to each model, and classified into one of the 38 PlantVillage categories. The paper references are included at the end of this subsystem report.

3.2.1 InceptionV3

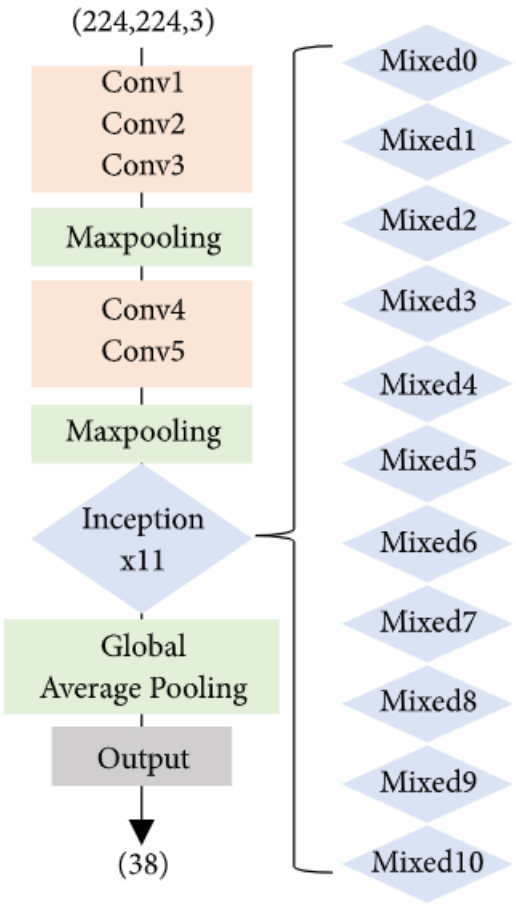


Figure 4: InceptionV3 Block Diagram

Background: InceptionV3 is a widely-used model for image recognition. For the purposes of this project, the model was initialized with random weights, and trained from scratch. The block diagram of the InceptionV3 model is depicted to the left.

Input: Image with size (224, 224, 3).

(The image width and height are 224 pixels, with 3 color channels to represent RGB images.)

Output: A vector of size 38 containing the probability distribution of each class.

Mixed layers are indicative of deeper convolutional layers, where both the complexity of shape and color diversity is expected to increase.

Global Average Pooling: the 2nd-to-last layer performs an Average Pooling instead of a Maxpooling. For feature extraction, Maxpooling is valued over Average Pooling since only the most important features are desired to be kept. In contrast, average pooling computes the average of all pixels instead of the max, resulting in all values incorporated into the output vector. Consequently, average pooling is a more generalized computation. This is preferred for the output vector since classification, not feature extraction, is the focus, and the pooling does not significantly alter the classification results obtained during training.

Softmax: The resultant vector is then fed into a softmax layer, which normalizes the vector into a probability distribution of values from 0 to 1.

3.2.2 AlexNet

The AlexNet and GoogLeNet models were built using Caffe, a deep learning framework, and were based on the implementation in [1]. While the original paper uses five different train-test splits (20:80, 40:60, 50:50, 60:40, 80:20), this implementation only used the 80:20 split. To reduce training time, the models used were pretrained on ImageNet. Both AlexNet and GoogLeNet were trained for 30 epochs, with a learning rate of 0.005. Weights were optimized using stochastic gradient descent. Both implementations were done on a single GPU.
AlexNet is comprised of 5 convolution layers, followed by 3 fully connected layers and finally a softmax layer. In contrast to InceptionV3, the model is much larger (60 million weights vs 25 million weights). The model was trained with a batch size of 24.

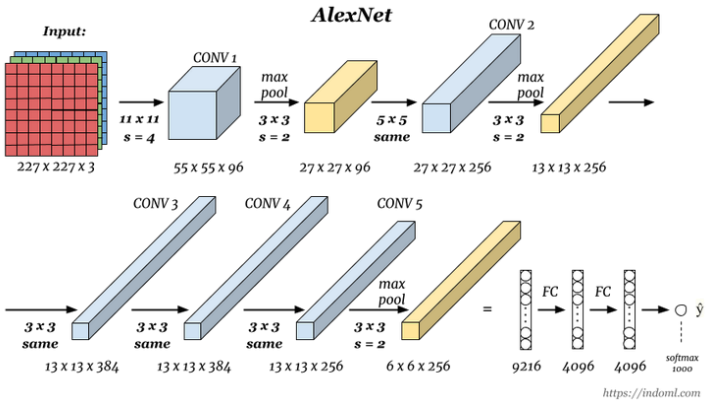


Figure 5. AlexNet Model Architecture

3.3.3 GoogLeNet

GoogLeNet is comprised of 22 layers. It is also known as InceptionV1, as it was the first model to that used inception modules. Out of the 4 models analyzed, GoogLeNet has the smallest number of parameters (5 million weights); however, the complexity of the model (22 layers) made training from scratch undesirable. The model was trained with a batch size of 100.

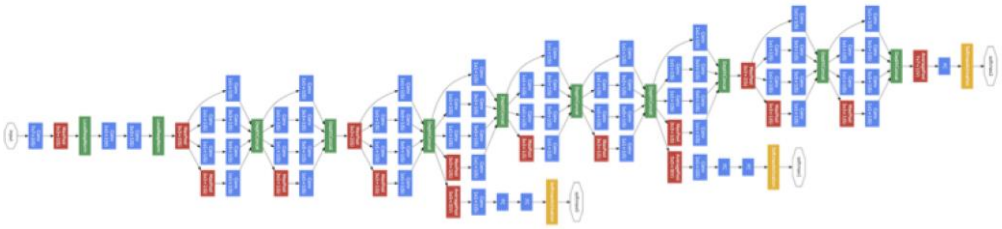


Figure 6. GoogLeNet Model Architecture

3.3.4 ResNet50

The ResNet50 implementation was taken from the standard Keras implementation [5], and consists of over 23 million parameters [4]. The model was trained from scratch using Keras. The model consists of repeating convolution and identity layers, as detailed in **Figure 5**. The advantage of the ResNet50 model is its ability to train very deep networks. Training of such networks had previously been impractical, due to problems such as vanishing and exploding gradients. ResNet50 cleverly avoids this problem through its use of skip connections, which connect a layer's output to the output of the layer before it, thus creating an identity function. This prevents gradient degradation during backpropagation to ensure that deeper layers will perform at least as well as shallow layers, and not worse.

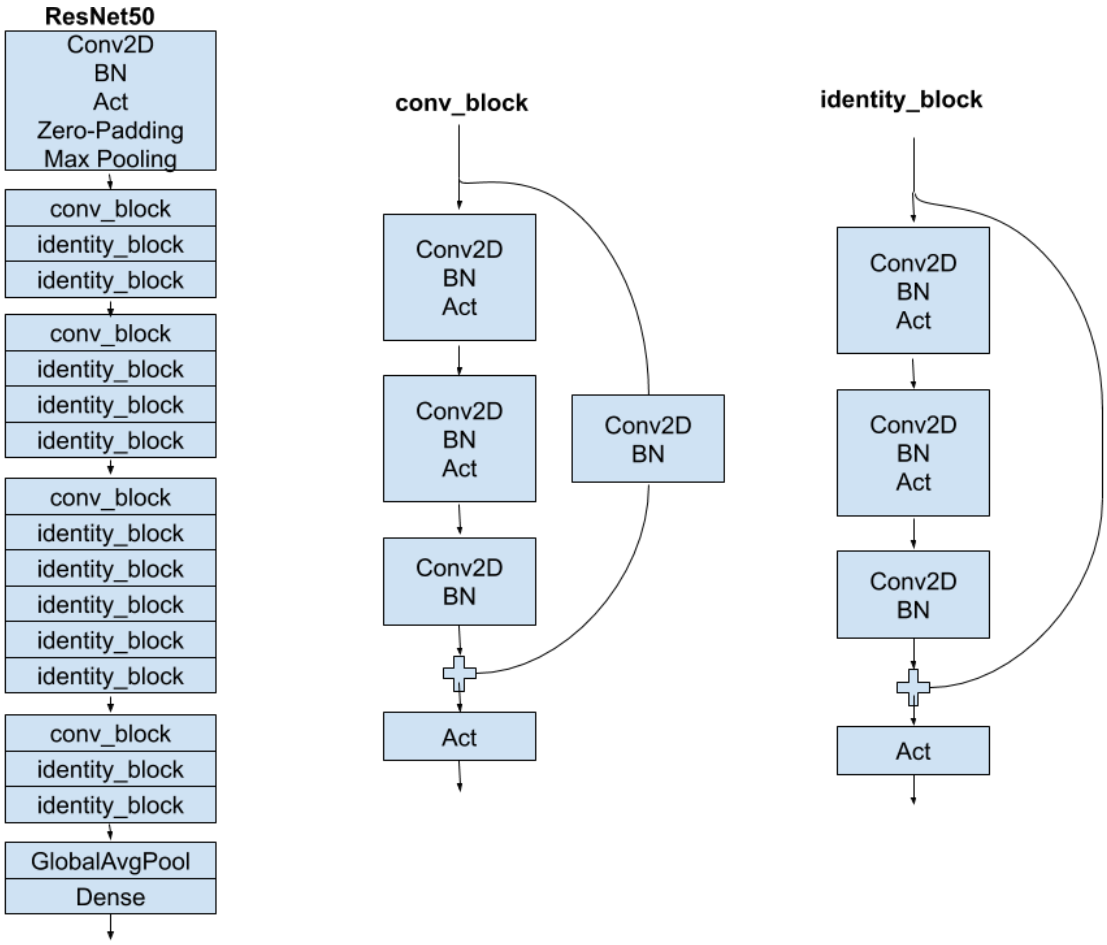


Figure 7. ResNet50 Model Architecture

The network was initialized with a random set of weights, and optimized using the Adam optimizer with a learning rate of 0.005. Images were split into training, validation, and testing data in a 60:20:20 ratio, and were fed into the network in batches of 32.

3.3.5 Attention Modules

Lastly, the structure of attention modules was also explored [3]. An attention module uses a bottom-up, top-down feedforward structure to generate attention-aware features. This feedforward structure was used to combine the feedforward and feedback process into a single process. **Figure 6** illustrates the structure of AttentionResnet56, a model constructed in [3] using a stacked attention module structure.

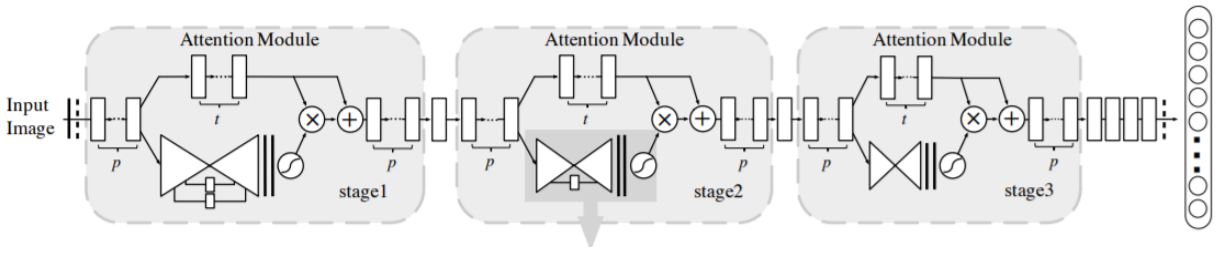


Figure 8. AttentionResnet56 Model Architecture

4. Results

The following section analyzes the training results for the models previously described. All implementations and experiments were performed on Texas A&M's HPRC Terra cluster.

4.1 InceptionV3

Figures 7 and **8** illustrate the performance of InceptionV3. The model was trained on each of the color, grayscale, and segmented categories. Validation accuracies were 94.9%, 93.2%, and 92.8%, respectively. Both the grayscale and segmented categories converged before 50 epochs based on the trend of improvement in validation loss.

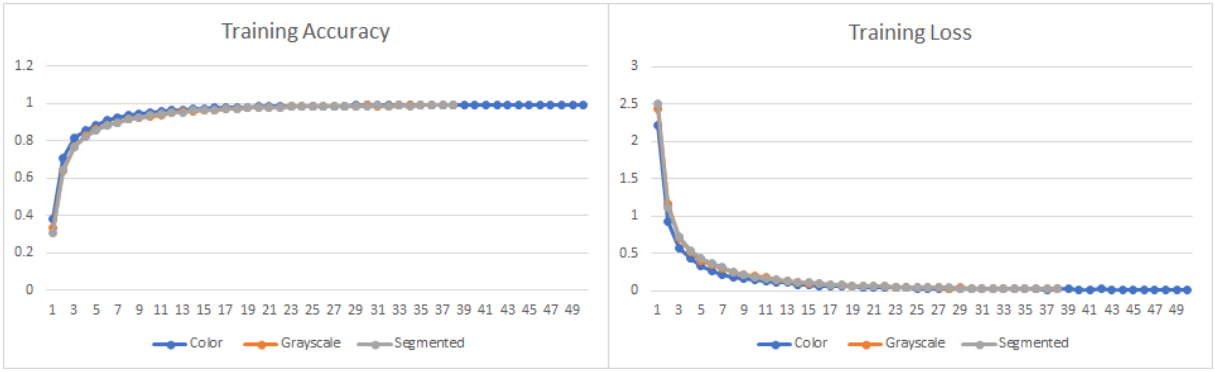


Figure 9. InceptionV3 training results. (Left) Training accuracy, (Right) Training loss

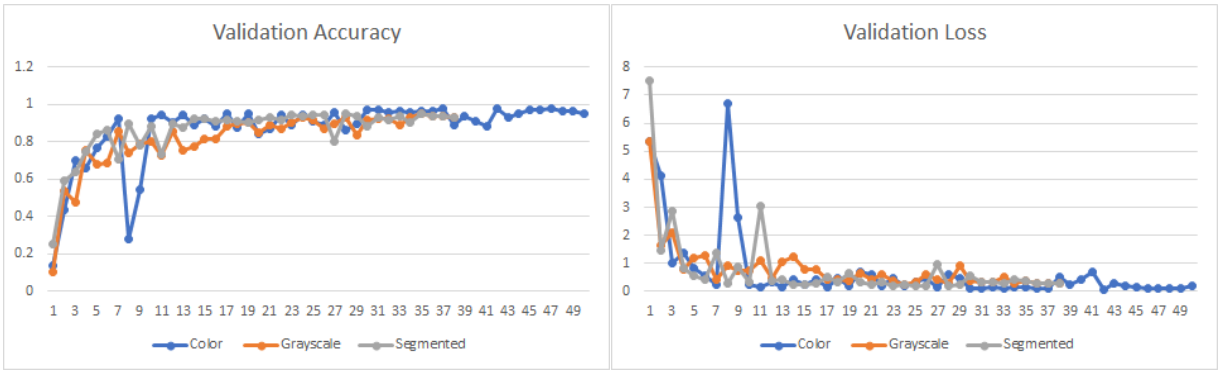


Figure 8. InceptionV3 validation results. (Left) Validation accuracy, (Right) Validation loss

4.2 AlexNet

Figures 9 and **10** illustrate the performance of AlexNet. The model was trained on each of the color, grayscale, and segmented categories. Test accuracies were 99.2%, 97.0%, and 98.4%, respectively. These are higher than the accuracies obtained for InceptionV3, most likely because a pretrained model is used.

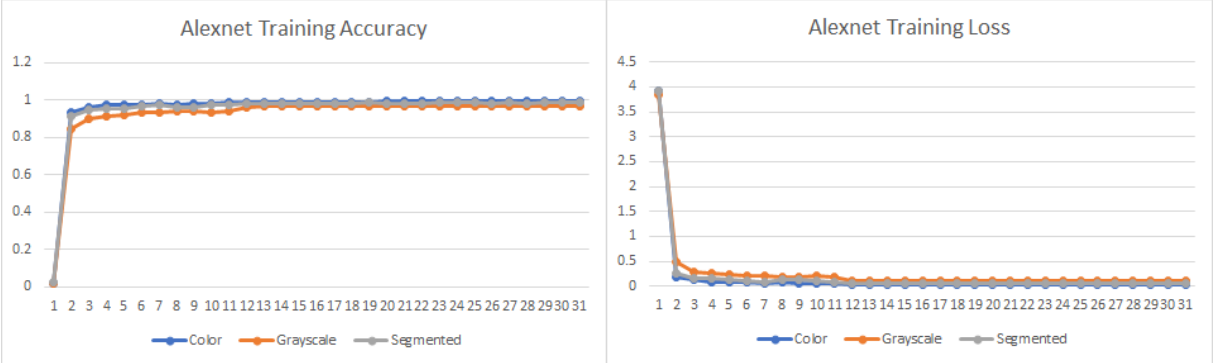


Figure 11. AlexNet training results. (Left) Training accuracy, (Right) Training loss

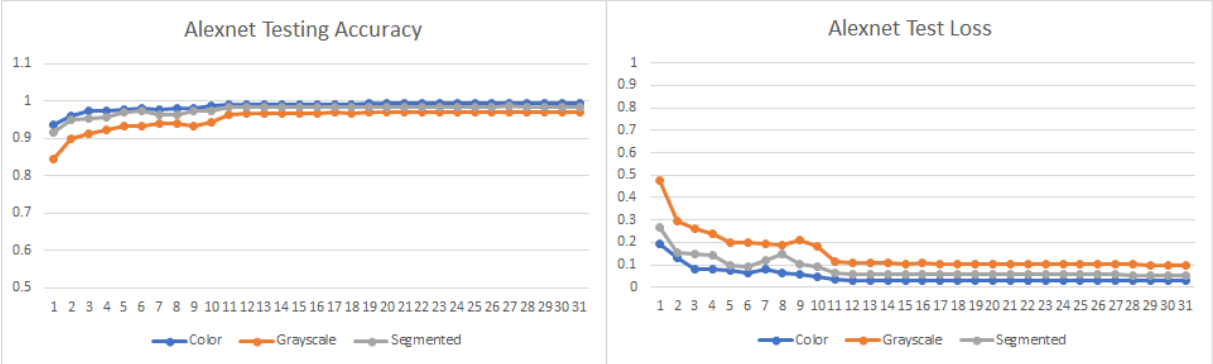


Figure 12. AlexNet testing results. (Left) Testing accuracy, (Right) Testing loss

4.3 GoogLeNet

Figures 11 and **12** illustrate the performance results of GoogLeNet. Again, the model was trained on each of the color, grayscale, and segmented categories, and achieved test accuracies of 99.5%, 98.1%, and 99.2%, respectively. As in the case of AlexNet, the test accuracy is higher than the accuracies obtained for InceptionV3, most likely because a pretrained model is used.

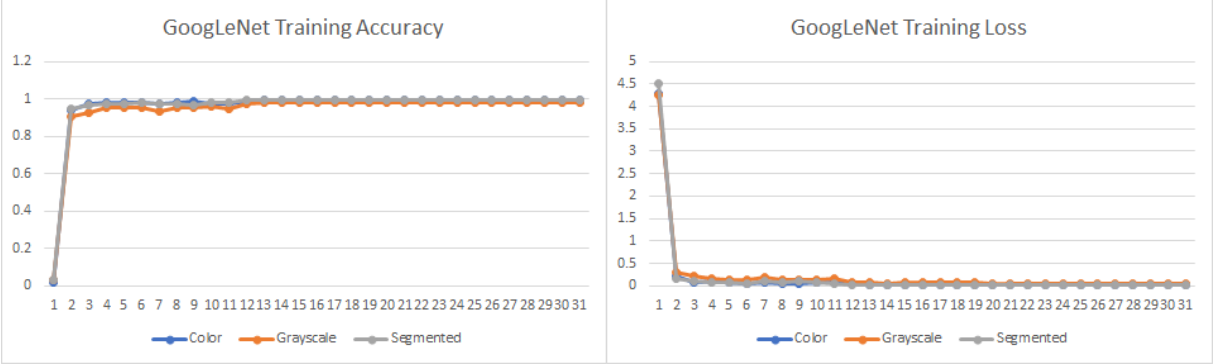


Figure 13. GoogLeNet training results. (Left) Training accuracy, (Right) Training loss

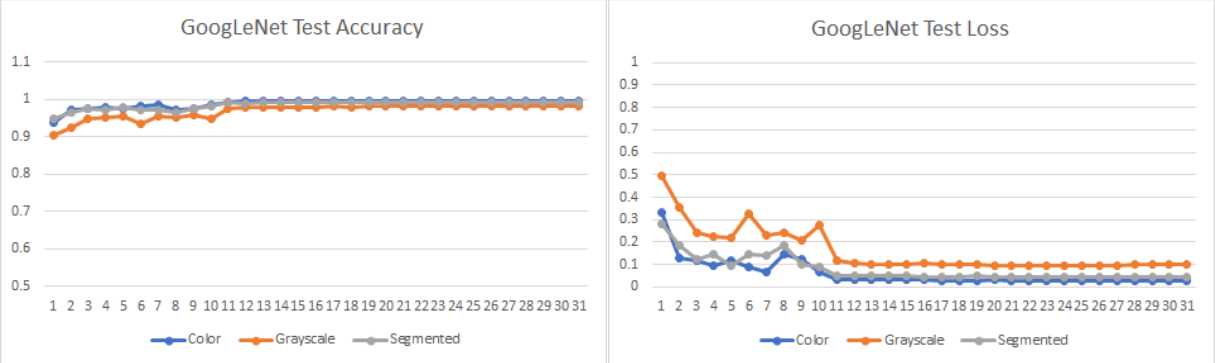


Figure 14. GoogLeNet testing results. (Left) Testing accuracy, (Right) Testing loss

Further analysis of the above three models shows that while the grayscale category tended to reach a lower accuracy than the color/segmented categories, every model reached a consistent >95% test accuracy. The lower grayscale accuracy can be attributed to the fact that lesions often have a different color than the leaf itself, making them less likely to be captured by a grayscale image.

4.4 ResNet50

Figure 13 illustrates the performance results for ResNet50. In contrast to the other models, it was only trained on the color category. The model reached a validation accuracy of 86%, which is the lowest of all models trained. Due to the limited computational resources, the reported accuracy may not reach the local optimum as there are oscillations of training curves. With more training epochs, accuracy may be further improved.

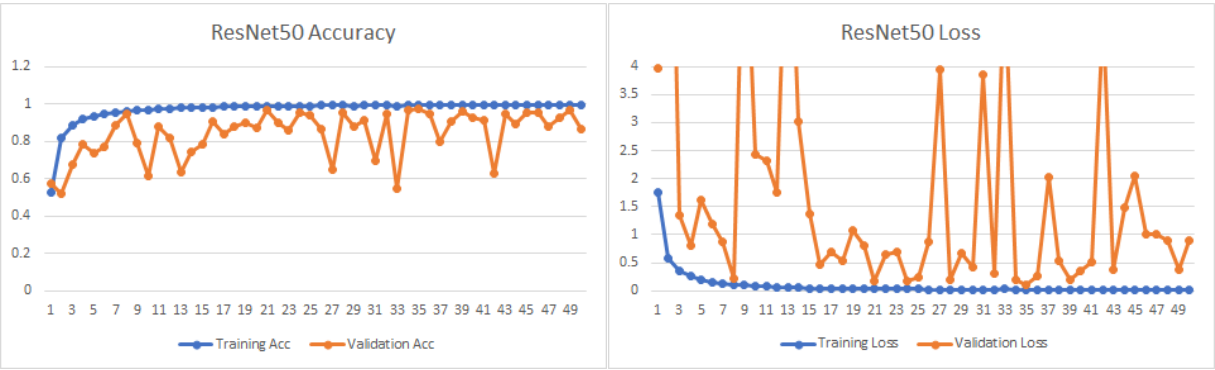


Figure 15. ResNet50 results. (Left) Training and Validation accuracy, (Right) Training and Validation loss

4.5 Attention Modules

While feature visualization and attention maps provide useful insights into features learned, they are primarily for human interpretation, and do not influence the actual training process. In contrast, attention modules generate attention-aware features [3], and adding them to the ResNet50 model could lead to improved performance.

Addition of the attention modules was done after careful study of the ResNet50 and AttentionResNet56 structures. Both models consist of a repeating structural pattern – convolution and identity blocks in ResNet50, compared to residual and attention blocks in AttentionResNet56. Upon further inspection, it was noted that the *conv_block* in ResNet50 and *residual_block* in AttentionResNet56 are similar, as both consisted of 3 convolutional layers and a skip connection. Thus, the identity blocks in ResNet50 were replaced with attention modules in multiple stages, as detailed in **Figure 17**. The resultant models were then trained, but only on the color image category of the PlantVillage dataset.

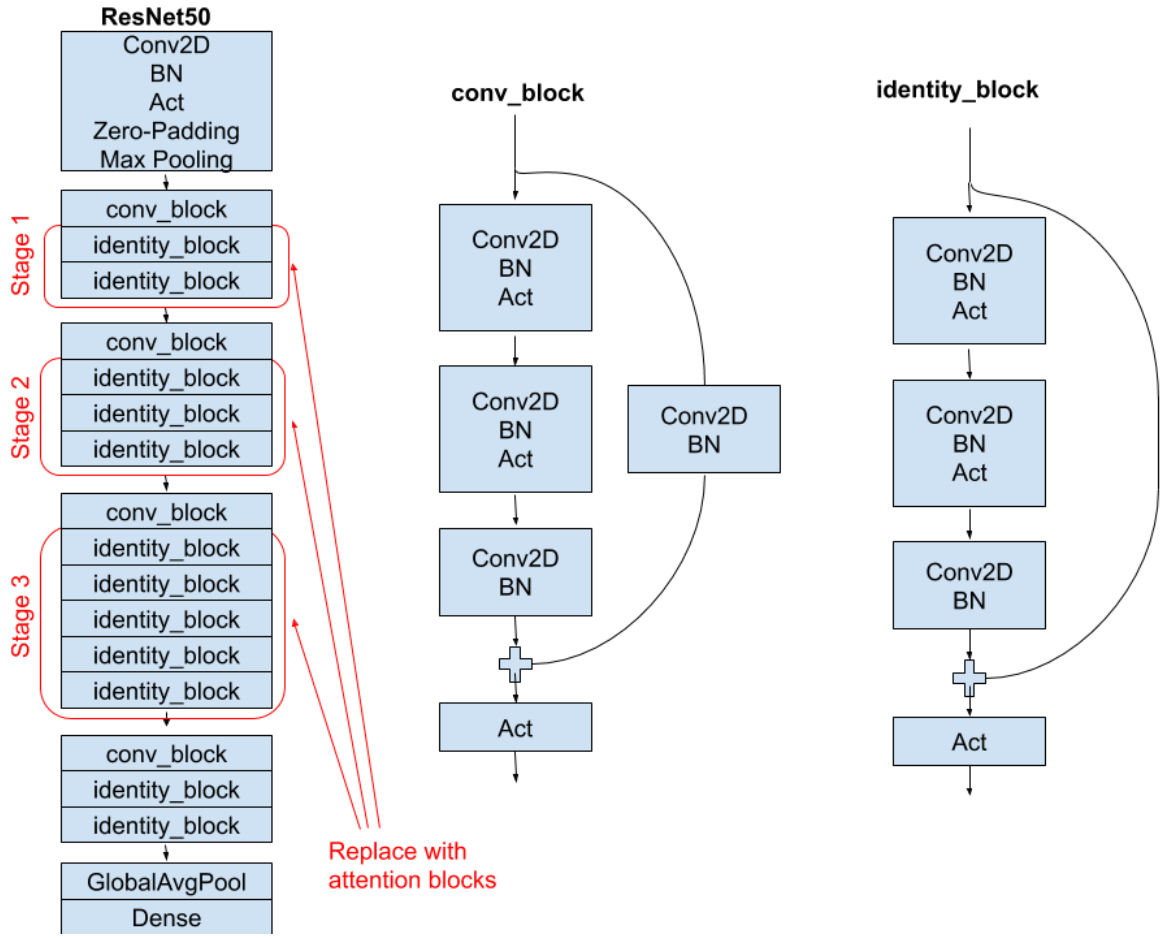


Figure 16. Proposed changes to ResNet50

Addition of the first and second attention blocks produced validation accuracies of 86% and 90%, respectively. However, addition of the third attention block led to a visible decrease in performance, and the model achieved 77% accuracy. Both the training and validation plots are characterized by many inconsistencies and sudden jumps in accuracy, as depicted in **Figure 18**.

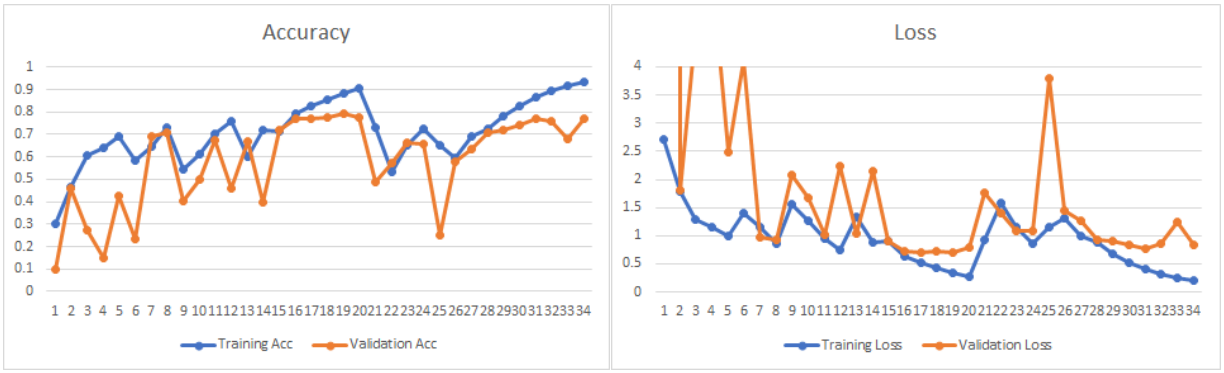


Figure 17. ResNet50 model (3 attention blocks) results. (Left) Training and Validation accuracy, (Right) Training and Validation loss

These results demonstrated that addition of multiple attention blocks resulted in an overly complex model for the PlantVillage dataset. It was then speculated that decreasing the model complexity could potentially increase accuracy. To test this hypothesis, Stage 3 of ResNet50 was removed altogether. **Figure 19** displays the model results. The model achieved a validation accuracy of 95%, much higher than the 86% observed earlier. These results illustrate that increased model complexity can adversely affect performance. By the same reasoning, elimination of layers could lead to performance improvements.

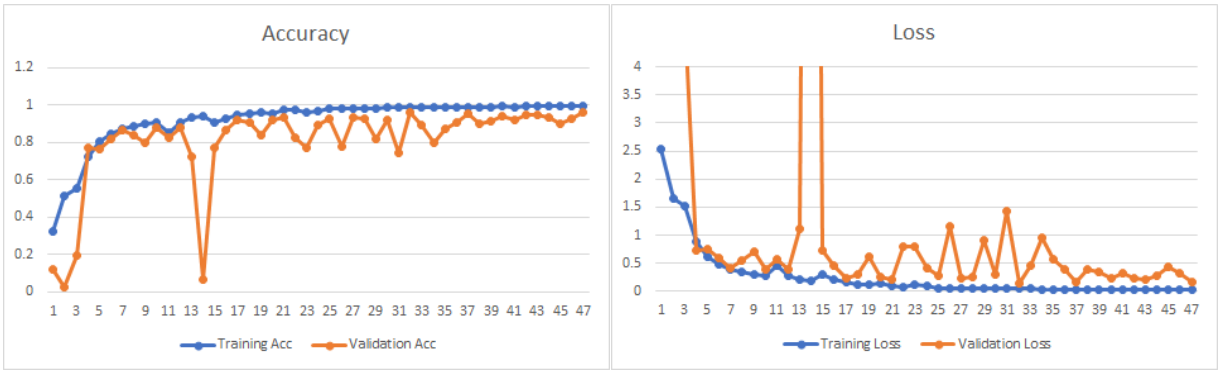


Figure 18. ResNet50 model (3 attention blocks, 3rd stage removed) results. (Left) Training and Validation accuracy, (Right) Training and Validation loss

To this effect, the ResNet18 model was trained on the PlantVillage dataset, and subsequently analyzed. The model achieved 97% accuracy, as shown in Figure X. The ResNet18 models appears to converge much faster than ResNet50, and achieves a higher accuracy. It is concluded that elimination of convolutional layers from the model does not have a significant effect on accuracy, and that the model can be simplified for the controlled dataset quality.

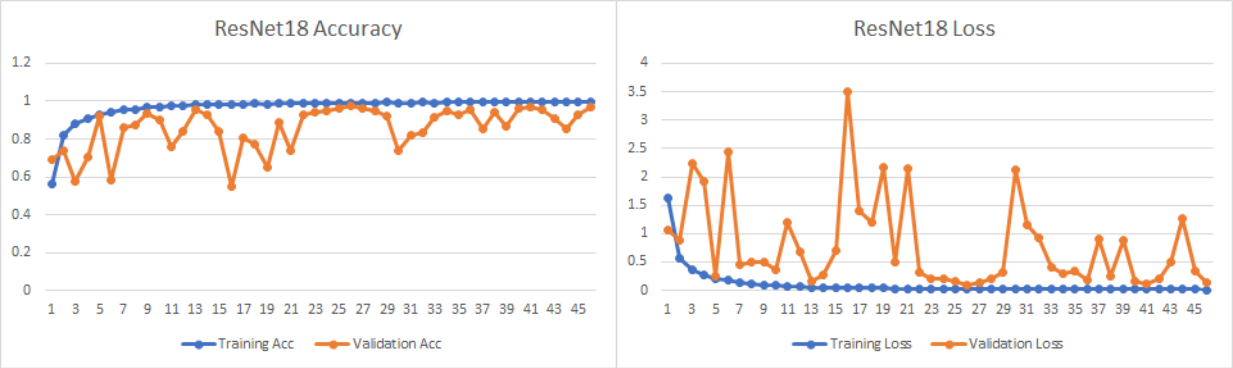


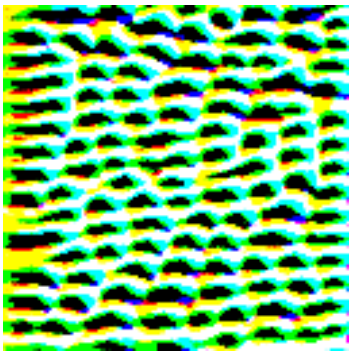
Figure 19. ResNet18 model results. (Left) Training and Validation accuracy, (Right) Training and Validation loss

5. Validation

The InceptionV3, AlexNet, and GoogLeNet models achieved very high accuracies for the PlantVillage dataset. It was unclear, however, how the models made their predictions – for instance, which features the model responded positively/negatively to, and how those choices affected the final classification results. Probing the network to gain more insight into these learned features will further validate the results obtained, and potentially lead to further model fine-tuning. In this section, the InceptionV3 model structure is analyzed through various visualization methods. The implementation for these methods is based on the work done in [2].

5.1 Feature Visualization

To gain a better understanding of features learned by the CNN, random neurons from every layer were optimized for a given input image. Because shallow layers are primarily focused on edge/corner detection, it was expected that their constituent neurons would produce images containing simple patterns, while deeper neurons would be geared towards object-specific shapes. The resultant images produced by the optimized neurons are depicted in **Figure 20**. As expected, the neuron from the Conv5 layer appears to be activated by simple, repetitive patterns – possibly indicative of striations on plant leaves. On the other hand, the Mixed10 neuron appears to be learning more image-specific objects, such as color or texture, as evidenced by the more abstract image. While multiple convolutional and mixed layers were investigated using this method, the majority of the images generated were rather abstract and their interpretation prone to subjectivity. Thus, this method was not pursued further.



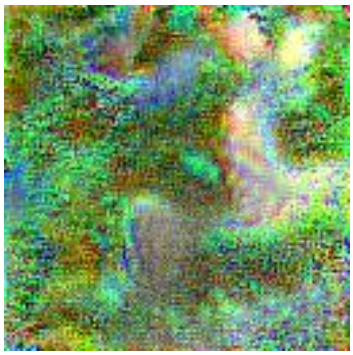


Figure 20. Feature visualization of neurons. (Left) Conv5 layer. (Right) Mixed10 layer.

5.2 Grad-CAM

The second approach tested was Grad-CAM, a type of attention map. Grad-CAM, or Gradient-weighted Class Activation Mapping, highlights the region of interest for a given image [6]. **Figure 21** shows the results of applying Grad-CAM to a leaf belonging to the Grape Black Rot category. From the figure, it is seen that the initial layers – Conv1 to Mixed0 – produce images with highlighted edges, indicating that the model believes these regions to be an important feature for classification. Inspection of the input image reveals the same regions of the leaf to be affected by black rot. These results affirm that the model is indeed capable of detecting lesions, and moreover, that they are a significant feature for classification. While the images corresponding to the deeper Mixed layers are nearly completely highlighted, this is most likely because the image size is continually decreased through the successive convolutional layers, resulting in a lower-resolution image.

Figure 21. Grad-CAM results for Grape Black Rot

Figure 22 shows the results of applying Grad-CAM to a leaf of type Tomato Yellow Leaf Curl Virus. Again, images produced from the shallower layers are of higher resolution than deeper ones. While the yellowed leaf edges are not distinctly marked, the model appears to be learning the texture of the tomato leaf. This is most apparent in the Conv2 and Conv3 layers, in which even the most minute texture details are clearly marked. In comparison to feature visualization, Grad-CAM provides more meaningful insights into features learned by the model. Nevertheless, the suitability of the model layer must be evaluated for the plant species beforehand [3].

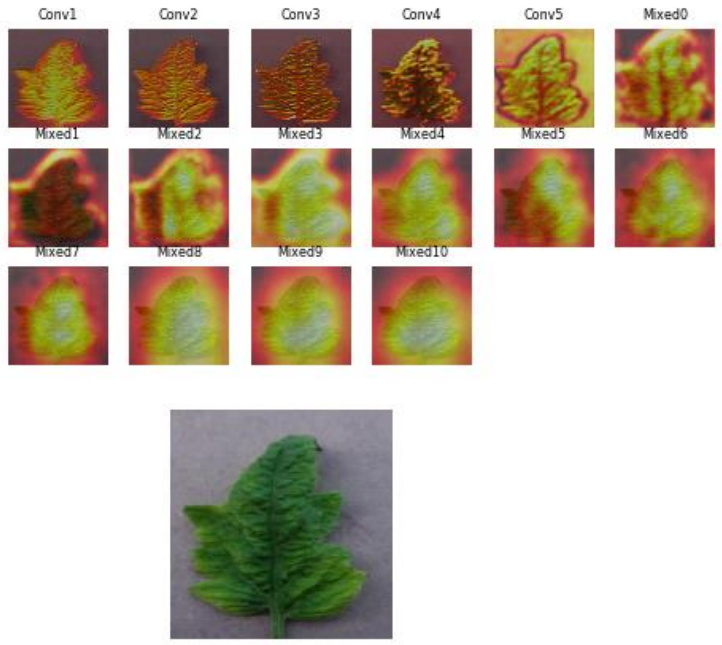


Figure 22. Grad-CAM results for Tomato Yellow Leaf Curl Virus

6. Conclusion

The Image Processing subsystem is an integral part of the overall plant disease classification system, as it provides the classifier functionality. The fact that the above models reached an accuracy approximately >90% demonstrates that a variety of architectures are capable of successful diagnosis of plant disease, although differences in model structure contributes to discrepancy in accuracy.

6.1 Learnings

Many skills were gained during the development of this subsystem. First and foremost, I gained valuable experience working with various neural network models, machine learning, and image processing concepts. While not all of these ideas were incorporated into the final design, the amount of background knowledge gained throughout the development of this subsystem was priceless, and made a significant difference in the final subsystem outcome. In addition to deep learning techniques, I also picked up a large amount of Python, as all code was written exclusively in Python. I polished and expanded my knowledge of command-line Linux, as I used the command-line interface exclusively for developing and training my models on HPRC. Lastly, many important research-oriented skills such as paper reading/analysis and communication, were also developed.

Deep Learning for Plant Disease Classification

Srishti Kumar

MOBILE APPLICATION SUBSYSTEM REPORT

REVISION – 404 Final

April 26, 2020

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	12/4/19	Srishti Kumar		End of Semester Submission
0.1	4/26/20	Srishti Kumar		End of Semester Submission

Mobile Application Subsystem Table of Contents

List of Figures

1. Introduction

The purpose of this subsystem report is to discuss the details of the Mobile Application subsystem. This subsystem aims to provide a user interface to the project through the use of a mobile app.

2. Theory

Deep learning has proved to be quite successful for plant disease diagnosis. While the models discussed thus far have achieved high accuracy, it is essential to provide a user interface to the system. The model can then be easily accessed by scientists, researcher, or farmer for their work. In today's technology-dominated society, nearly every individual either possesses or has access to a smartphone. Thus, deploying the model to a smartphone seemed the most ideal consumer application, especially considering its small size and ease of use. The app was created using the Tensorflow Lite Android library [7], and based on the library's Image Classification template. The app takes as input a live camera preview and outputs instantaneous classification results. The mobile application developed allows real-time classification of plant leaves.

3. Design

The app used the default design provided by the Tensorflow Lite Android library. The top and middle section display the live camera preview. The bottom section displays the top 2 classification results and their respective accuracies. This tab can be expanded to display additional information, such as the image size, number of threads, and model specifications.

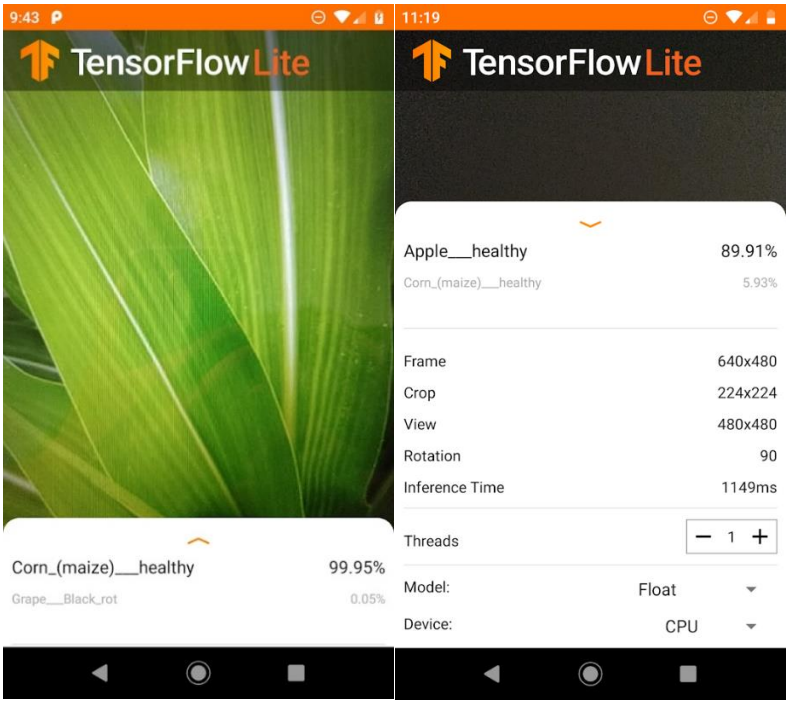


Figure 1: Mobile Application Interface

3.1 Software

The app used the default software provided by the Tensorflow Lite Android library. The codebase consists of multiple files that control the graphics layout, camera, and model deployment. While

these files were not modified, they were read carefully to gain a general understanding of their functionality. Extra print statements were also added as needed when debugging erroneous classification results. The .tflite file (containing the model) and labels file (containing the plant categories) were then edited to reflect the trained model .

3.2 Model Deployment

Model deployment consisted of several steps. Due to size and memory restrictions of smartphones, the existing model (.h5 file) needed to be converted into a more lightweight model (.tflite file). Many attempts were made to directly convert the model from .h5 to .tflite, in addition to using a .pb file as an intermediate. However, the results were either inconsistent or locked at a single classification, most likely due to unseen conversion errors related to the model weights. Several codebases, such as the Tensorflow Demo App, amongst others, were tested, yet resulted in the same errors. Thus, to avoid any conversion errors, the model was retrained and directly saved as a .tflite file.

Ultimately, the model was successfully trained on images from 5 of the 38 plant species: Grape Black Rot, Corn (maize) healthy, Apple (healthy), Tomato Yellow Leaf Curl Virus, and Squash Powdery Mildew. As seen in **Figure 20**, the app successfully classifies this particular leaf as Corn (maize) healthy, with 99.95% accuracy. Addition of more species (7 species, or 10) often resulted in the locked classification results. This is most likely attributed to errors in model deployment. Multiple models, such as NasNet, DenseNet, and ResNet50 were tried in fixing this issue. Although multiple smaller and larger models were tested, the issue persisted, and the app currently only classifies 5 out of 38 species using the InceptionV3 model.

Figure 2: Classification Results

4. Validation

The Mobile App subsystem was validated throughout multiple steps. First, since the model deployed had an accuracy of >95%. After deployment, multiple images were classified with >90% accuracy. Because the app uses a live camera preview, the classification results are slightly challenging to pinpoint at times, since they update every second. However, showing the model the same picture for several seconds would lead to the same classification result, thus affirming the original premise. In testing this model, random images from the internet were searched for, keeping in mind that controlled image quality would lead to a higher classification result since that is what the model was trained on. More details on this subsystem validation can be found in the system report that follows this one.

5. Conclusion

The Mobile Application subsystem is an integral part of the overall plant disease classification system, as it provides a user interface to the system. The model does have its shortcomings – only 5 out of the 38 species were included for in the final model – but it is important to note that in real-world situations, the number of species tested would be far less, as growers would already know the species of the crop they are growing.

5.1 Learnings

Many skills were gained during the development of this subsystem. The most important was gaining exposure to and experience working with mobile app deployment. As neural network models become increasingly prominent, knowing how to deploy a model to a mobile application will become a useful skill, as it makes the classification process very simple and convenient.

5.2 Extensions

To further facilitate user interaction with the model, a Graphical User Interface (GUI) was also developed. Note that this interface is completely separate from the mobile application and was simply built as another user interface. The GUI was developed using the Python (Tkinter) library. To classify an image, the user simply selects the “Upload image” button to upload an image from their file system. The image is then processed by the GUI backend, and outputs a classification. The classification results then appear on the GUI, as shown in **Figure 21**. With the exception of the initial image, individual images are classified within seconds. While the GUI could be extended to provide support for batch classification, this interface is primarily focused on providing a simple high-level understanding of the overall classification process.

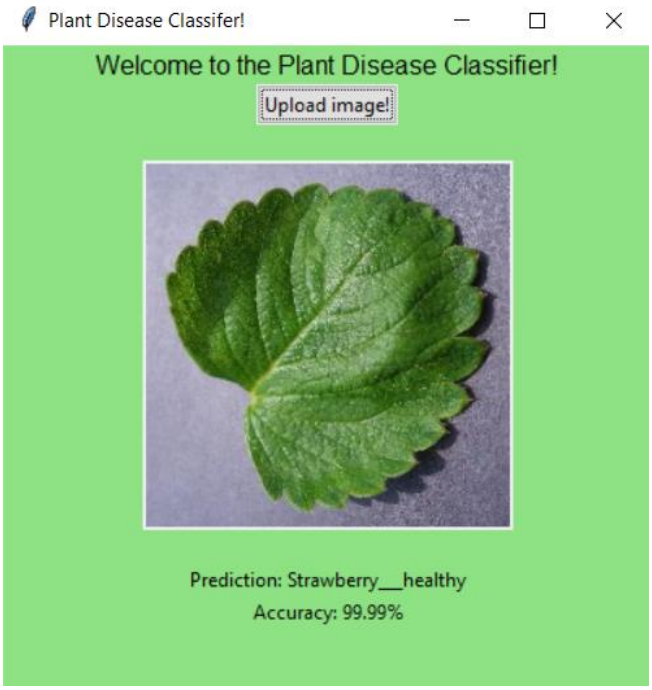


Figure 3: GUI Classification Example

Deep Learning for Plant Disease Classification

Srishti Kumar

SYSTEM REPORT

REVISION – 404 Final

April 26, 2020

Change Record

Rev.	Date	Originator	Approvals	Description
0.1	4/26/20	Srishti Kumar		End of Semester Submission

System Report Table of Contents

1. Overview

This project is an accurate and reliable design for plant classification. The overall system consists of two subsystems – Image Classification, and Mobile App development. Integration of all subsystems was performed to ensure a functioning system, and validation and data were necessary to ensure each subsystem met the expected standards and specifications. Upon culmination of these tasks, the classifier model was finalized and ready for use as a simple and efficient tool in the field.

2. Development Plan and Execution

As stated previously, two subsystems composed the final classifier. The Image Processing subsystem, being the core of the project, was the main focus for both the fall and spring semesters. Development of the Mobile Application was initiated in the fall semester and finalized in the spring semester. This report primarily focuses on the spring semester design process, as it is the most relevant.

3. Design

3.1 Dataset

The design aspect was simplified by the PlantVillage dataset. For the majority of deep learning models, data collection (and preparation) often constitutes a significant part of the project – almost as much as the actual model design and training itself – as the model is only as good as the data that it is trained on. The presence of the PlantVillage dataset significantly simplified this aspect as I was not required to collect my own data. Later on, when I tried to collect my own data in the spring semester, I ran into all sorts of challenges – unlabeled data, different species, lack of suitable images, etc – and concluded that the PlantVillage dataset, while controlled, was an extremely good start since it allowed me to focus exclusively on model development.

3.2 Model Development

With regards to model development, a variety of models were trained and tested on the PlantVillage dataset. These include InceptionV3, AlexNet, GoogLeNet, and ResNet50. A variety of different features were also explored, such as pretrained networks, residual connections, elimination of intermediate layers, etc. While many may wonder why it was necessary to train such a diverse array of models, model architecture and sizing is just as important as hyperparameter turning. While pretrained, heavier models such as ResNet50 may yield 90%+ accuracy on the PlantVillage dataset, for instance, if elimination of multiple layers/pretrained ImageNet weights leads to a slimmer model with the same accuracy, this would lead to a significant improvement in terms of training time and resources. Additionally, more lightweight models are also easier to deploy to a smartphone.

4. Execution

Both subsystems fit into each other very smoothly; as such, subsystem integration did not necessitate much change. The model developed in the Image Processing subsystem was deployed to the Mobile Application. This interface then used the model as a black box to perform the classification, and output the results to the user. The milestone plan is shown in the below figure.

Date	Objectives	Execution
9/29	Milestone	Read *Deep Learning with* Python (Keras) textbook. Experiment with simple CNNs on toy datasets.
10/6	Milestone	Create input dataset of 6:2:2 ratio Start work on reproducing InceptionV3 model
10/13	Milestone	Continue work on reproducing InceptionV3 model Experiment with various techniques to reduce overfitting
10/20	Validation1	Reproduce InceptionV3 model (train from scratch)
10/27	Validation2	Reproduce AlexNet model (transfer learning)
11/3	Validation3	Reproduce GoogLeNet model (transfer learning)
11/10	Milestone	Create a prediction script (Caffe) Comparison: Analyze all 3 models Demo
11/17	Milestone	Comparison: Analyze all 3 models
11/24	Milestone	Set up pipeline for Mobile App deployment
12/01	Validation4	Set up pipeline for Mobile App deployment
12/08	Validation5	Report
1/19	Milestone	Evaluate winter break progress
1/26	Validation	Read paper: How to incorporate attention results?
2/2	Validation	Incorporate attention results into CNN & evaluate results
2/9	Milestone	Incorporate attention results into CNN & evaluate results
2/16	Validation	Discuss new datasets and design new model(s)
2/23	Validation	Finish model design and begin training of new model(s)
3/1	Milestone	Finish model design and begin training of new model(s)
3/8	N/A	Spring Break – Debug Mobile Application
3/15	N/A	Spring Break (extended) - Debug Mobile Application
3/22	Validation	Debug AttentionResNet56: add attention modules to ResNet50
3/29	Milestone	Debug AttentionResNet56: add attention modules to ResNet50
4/5	Validation	Train ResNet18 and experiment with layer elimination
4/12	Validation	Train ResNet18 with attention
4/19	Milestone	Mobile App Polishing

Table 1: Execution and Validation Plan

5. Validation Plan

The validation plan of the fully integrated system is shown below. Because the Mobile Application subsystem includes the Image Processing subsystem, only the former was tested and validated (the latter was already tested and validated during its development). The app was shown 20 different images of each of the 5 species. The classification results (correct/incorrect) are depicted in the image below. As shown, the species Grape Black Rot, Corn (Healthy), and Apple (Healthy) achieved the highest accuracy (95%-100%). While accuracy is slightly lower for the Tomato Yellow Leaf Curl Virus and Squash Powdery Mildew species, they are still reasonably high, and a larger number of tests would have likely demonstrated a higher convergence. This test is relatively simple, and more generalizable results would necessitate a larger number of trials – perhaps in the thousands. While this was performed for the individual models, testing on such a scale for a mobile application would prove to be very challenging as it cannot be automated – the user must physically move their phone over a leaf (or an image of a leaf) – to obtain the classification result. However, these results do show promise for generalizability on larger datasets.

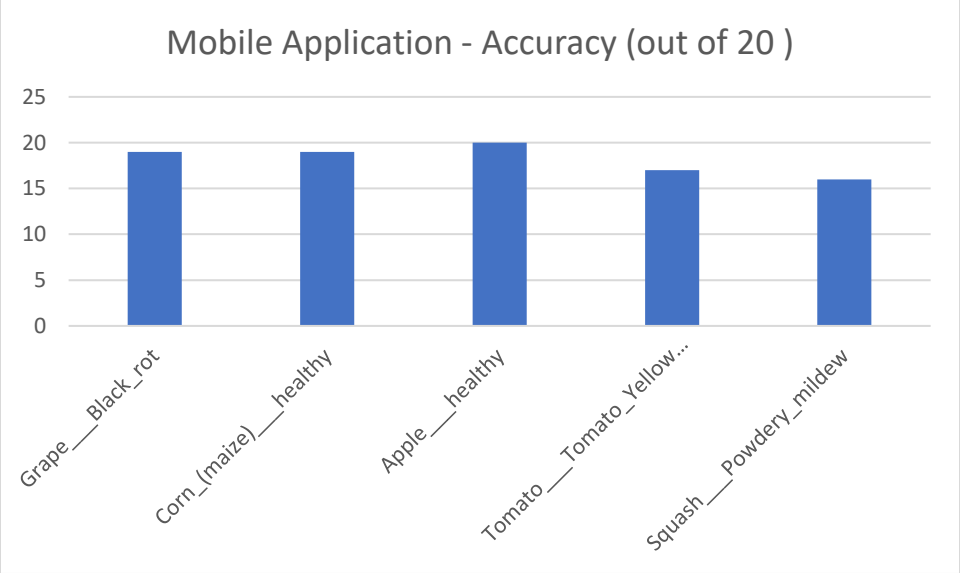


Figure 1: Mobile Application Validation

Collection of more data from the Internet could be a possible method to obtain additional data. However, it is worth noting that the research paper that compiled the PlantVillage dataset also compiled a smaller dataset of images (which, unfortunately, is not accessible) using Bing Image search tools to further test their model. Despite using sophisticated search filters, they were only able to acquire 2 small datasets of approximately 100 images, each image corresponding to one of the 38 categories. While such techniques could be a desirable next step for this project, the small dataset size, in addition to my lack of plant pathology expertise, would likely negatively influence the results.

6. Conclusion

Deep learning models hold great potential in plant disease diagnosis. In particular, CNN models have been shown to produce excellent results, due to their image-oriented model structure. Four different models were tested in this work, each possessing distinct characteristics such as inception modules,

residual learning, and pretrained weights. While each model achieved high accuracies, further analysis of the effect of these traits on the overall result could lead to development of more suitable models. Visualization methods such as feature visualization and attention maps shed more light on the features learned, by probing the network at various layers. These visualizations effectively opened the CNN "black box" for human inspection, and allowed validation of the aforementioned results. Moreover, it was also discovered that the removal of intermediate layers led to higher accuracy on the PlantVillage data, most likely due to the controlled leaf image quality. Additional efforts in model simplification could lead to further improvements in accuracy and loss, as well as reduction of training time. The classification success of the two user interfaces, developed to facilitate user interaction with the model, suggests their potential deployment in the field. While CNNs have shown great success in plant disease classification, crop disease diagnosis is but one of many fields to be yet transformed by deep learning.

6.1 Learnings

This Senior Design research project was an extremely valuable learning experience. While coordinating both the research and senior design aspects of the project was not always easy, invaluable lessons were learnt from both spheres. The research focus allowed me to experiment with a multiplicity of models, instead of solely focusing on the final product/external features. I initially came into this project with no prior knowledge of deep learning, and while the journey was initially rocky, the project gave me lasting real-world experience unobtainable from a classroom setting. Secondly, the senior design aspect gave structure to the project, as it demanded strict deadlines and planning. Development of external features such as the mobile application and GUI gave me basic UI experience, and added diversity to the final project. This project touched on a diverse array of skills and concepts. However, it is necessary to note that the final output of this project is not a product in itself – but rather a stepping stone to a larger discovery of deep learning used for plant disease diagnosis purposes. Using the pipelines developed in this research project, models for other datasets can easily be developed, trained, and tested. Moreover, these models can be fine-tuned for their respective datasets – farmers may only choose to classify 1 or 2 plant species, but for multiple diseases within those species. Techniques such as visualization can then be employed to ascertain that the model is learning the correct features. Ultimately, this project served as a capstone to my college experience, and the feedback and critique received throughout provided priceless insight on this project.