DEEP LEARNING FOR PLANT DISEASE CLASSIFICATION

An Undergraduate Research Scholars Thesis

by

SRISHTI KUMAR

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Xiaoning Qian

May 2020

Major: Computer Engineering

TABLE OF CONTENTS

Page

ABSTRACT....1

DEDICATION....2

ACKNOWLEDGMENTS....3

NOMENCLATURE....4

CHAPTER

I. INTRODUCTION....5

II. METHODS....7

Dataset....7
Model Architectures....7

III. RESULTS....12

InceptionV3....12
AlexNet....13
GoogLeNet....13
ResNet50....14

IV. VISUALIZATION....16

Feature Visualization....16
Grad-CAM....17
Attention Modules....19

V. APPLICATIONS....22

Mobile Application....22
Graphical User Interface (GUI)....23

VI. CONCLUSION....25

REFERENCES....26

APPENDIX....28

ABSTRACT

Deep Learning for Plant Disease Classification

Srishti Kumar
Department of Electrical Engineering
Texas A&M University

Research Advisor: Dr. Xiaoning Qian
Department of Electrical Engineering
Texas A&M University

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 thesis 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. Last but not least, 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.

DEDICATION

I dedicate this paper to my family, for their encouragement, patience and love.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Qian, and my TA, Xueting Liu, for their continuous guidance and support throughout the course of this research. They were always willing to point me in the right direction, whether it was providing resources, advice on research presentations, or analyzing results.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience. I also want to extend my gratitude to my senior design professor Dr. Nowka, for his continuous support. Finally, thanks to my mom, dad, and brother, for their encouragement, patience and love.

NOMENCLATURE

CNN	Convolutional Neural Network
GUI	Graphical User Interface

CHAPTER I

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 [1].

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," thus raising the concerns of interpretability of the learned model [2]. Observing and analyzing intermediate outputs during the classification will produce more insight into the particular features learned by the CNN. Careful analysis of these observations will then help optimize the model appropriately to improve its accuracy in plant disease identification and diagnosis.

First, four CNN models will be presented, along with a brief discussion of each model's structure. Next, the results of corresponding CNN models, trained on a plant disease image

benchmark focusing on plant leaves – PlantVillage [8] – will be presented and analyzed. The focus will then shift to several visualization techniques, their implementation, and interpretation of the results. Lastly, several user applications will be discussed.

CHAPTER II

METHODS

Dataset

The dataset used for the project is the PlantVillage dataset, a public GitHub consisting of 54,306 images [8]. Images are of size 224 x 224, and of 3 types – color, grayscale, and segmented. They are divided into 38 categories (26 diseases, 14 total crop species). **Figure 1** displays images of all 38 plant species.

Figure 1. PlantVillage Dataset [1]

Model Architectures

Four CNN models were trained and tested 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.

InceptionV3

The InceptionV3 model consists of 25 million parameters. Model construction and training was done using Keras (a Python library) with the Tensorflow backend [5]. 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 128 [2]. **Figure 2** illustrates the model structure in more detail.

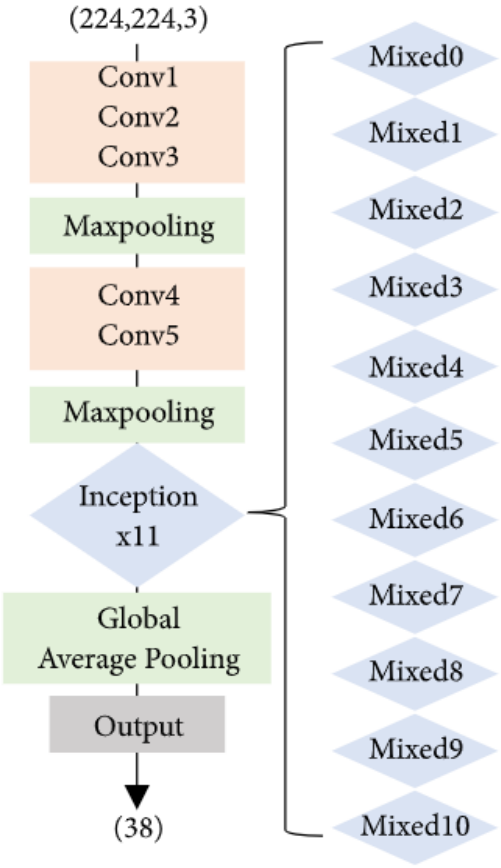

Figure 2. Inception V3 Model Architecture [2]

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.

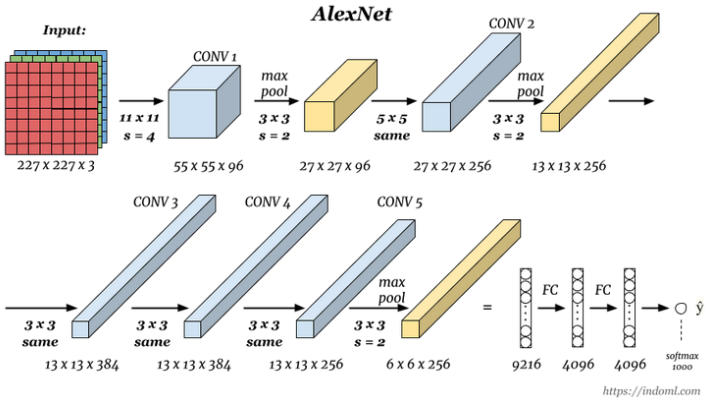

Figure 3. AlexNet Model Architecture [9]

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.

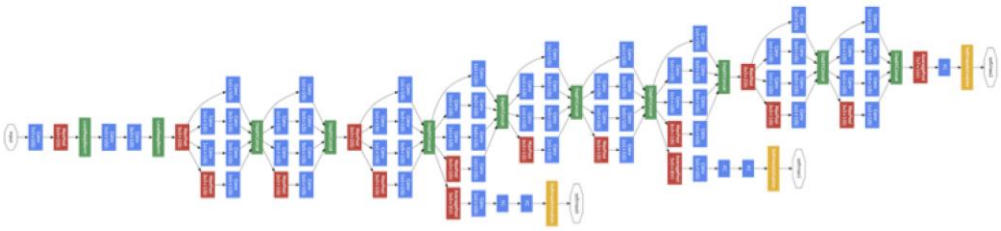

Figure 4. GoogLeNet Model Architecture [9]

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.

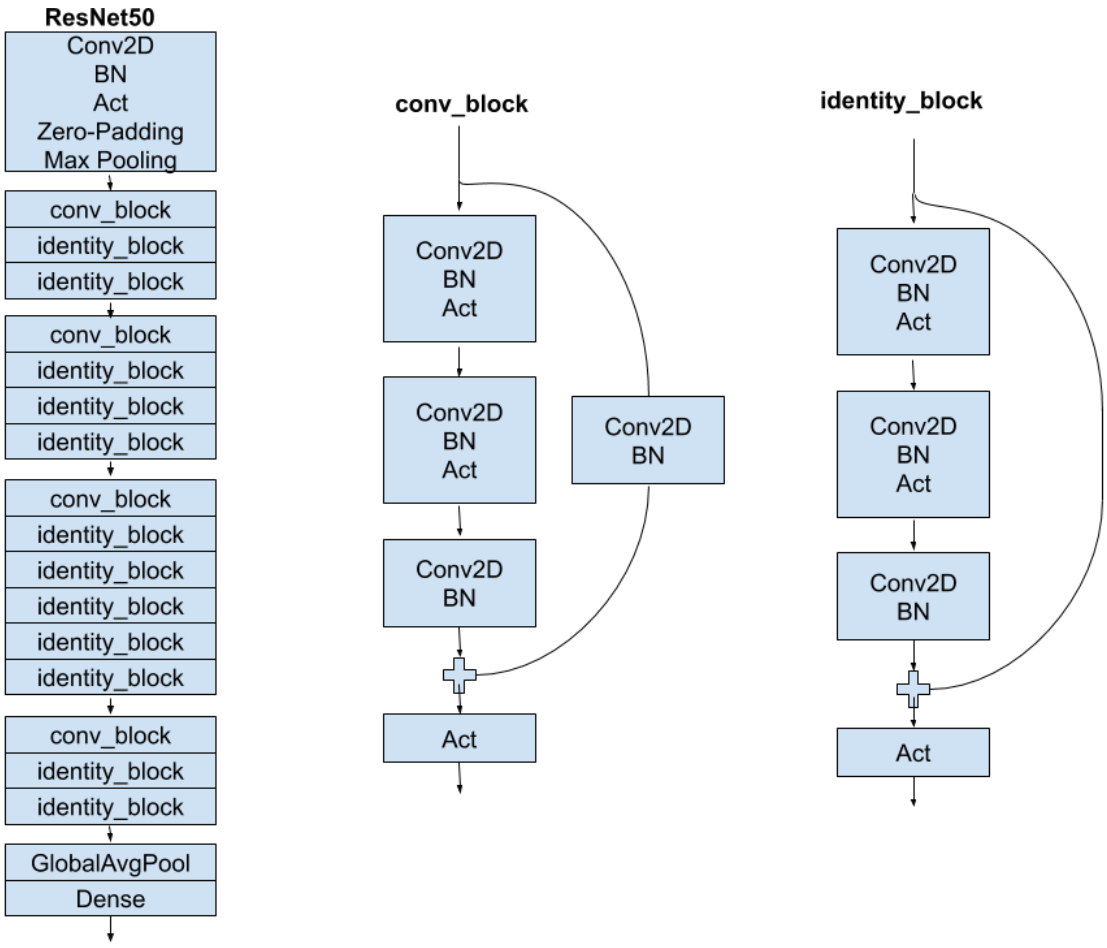

Figure 5. ResNet50 Model Architecture

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.

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.

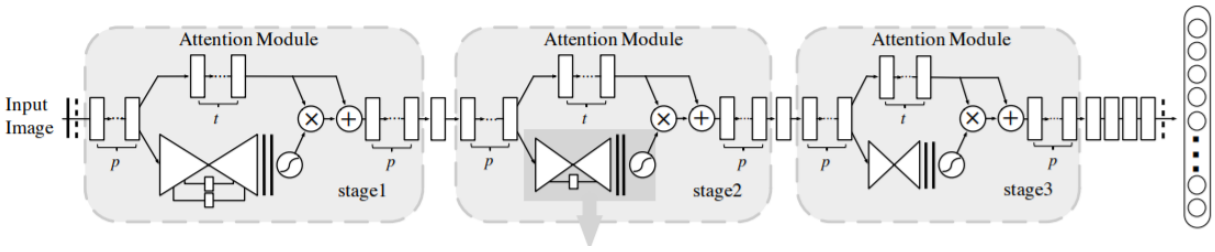

Figure 6. AttentionResnet56 Model Architecture [3]

CHAPTER III

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.

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.

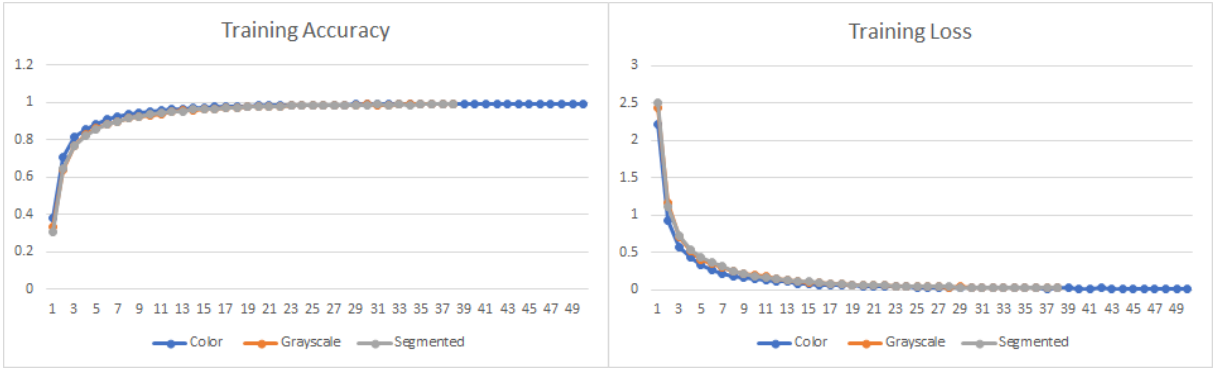

Figure 7. InceptionV3 training results. (Left) Training accuracy, (Right) Training loss

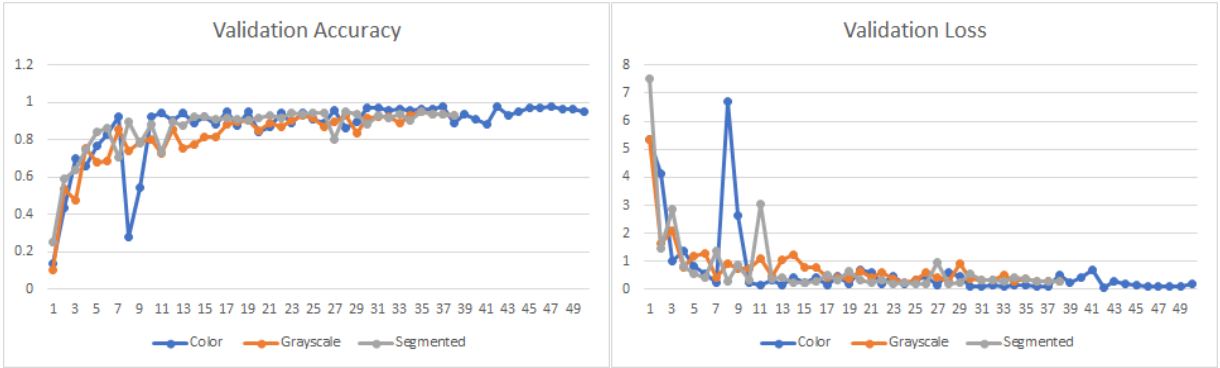

Figure 8. InceptionV3 validation results. (Left) Validation accuracy, (Right) Validation loss

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.

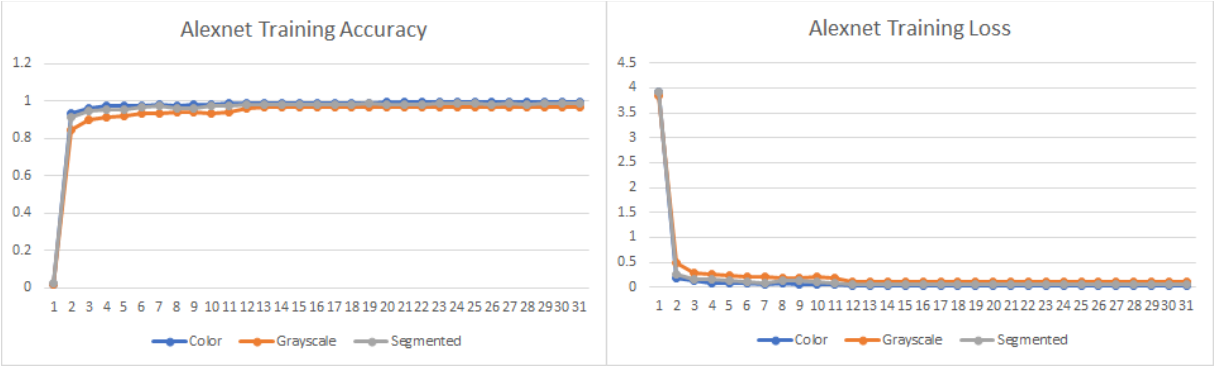

Figure 9. AlexNet training results. (Left) Training accuracy, (Right) Training loss

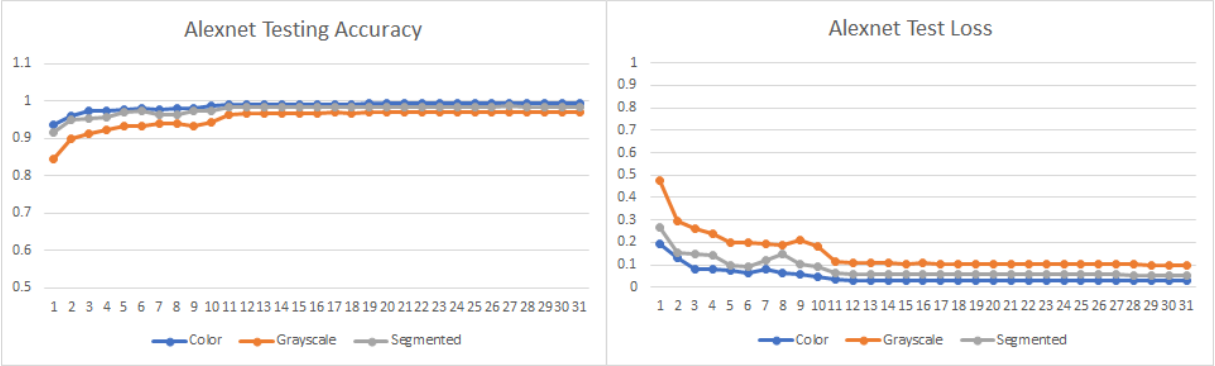

Figure 10. AlexNet testing results. (Left) Testing accuracy, (Right) Testing loss

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.

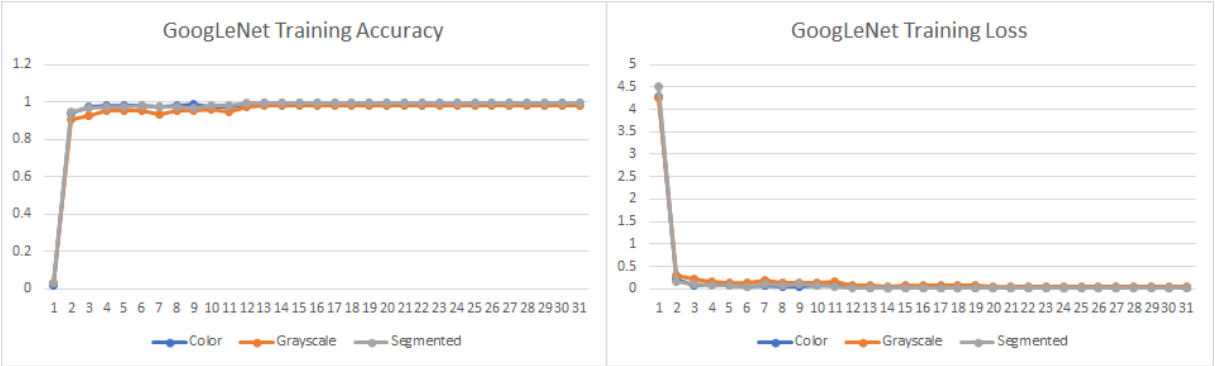

Figure 11. GoogLeNet training results. (Left) Training accuracy, (Right) Training loss

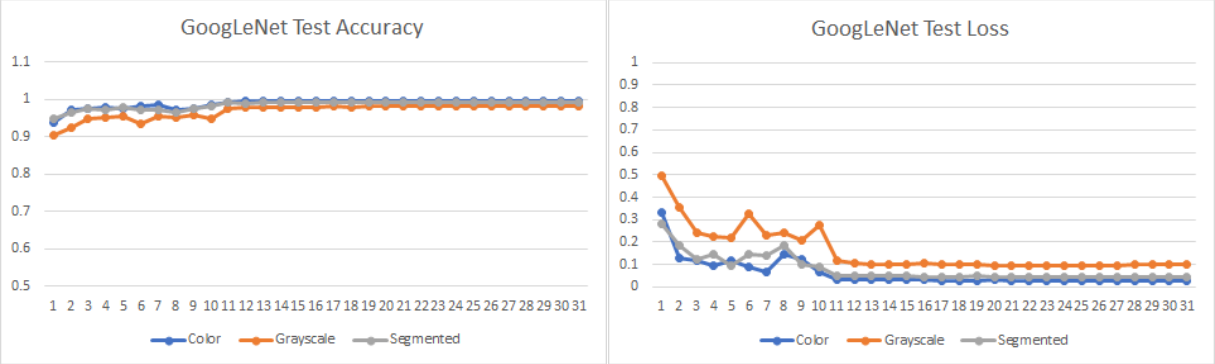

Figure 12. GoogLeNet testing results. (Left) Testing accuracy, (Right) Testing loss

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.

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.

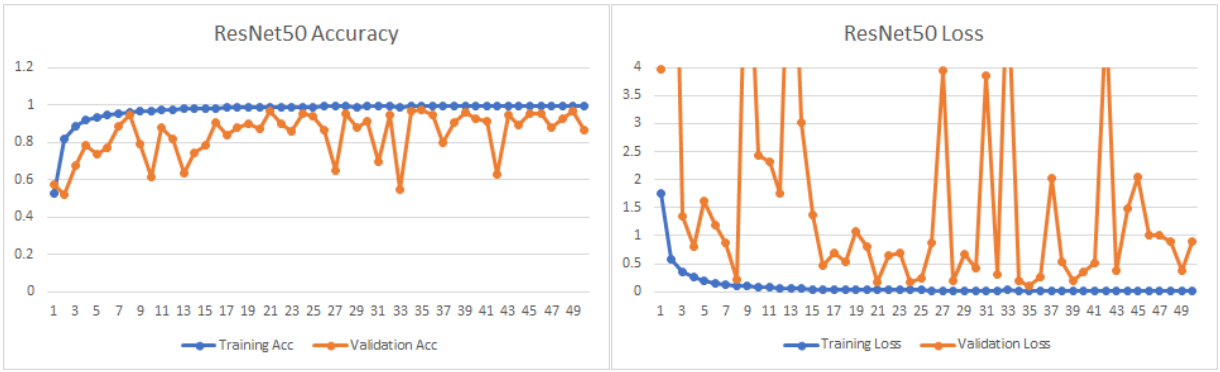

Figure 13. ResNet50 results. (Left) Training and Validation accuracy, (Right) Training and Validation loss

CHAPTER IV

VISUALIZATION

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 could 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].

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 14**. 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.

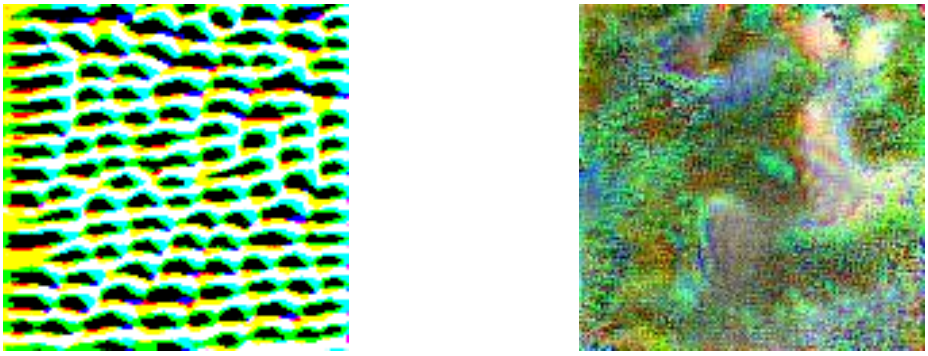

Figure 14. Feature visualization of neurons. (Left) Conv5 layer. (Right) Mixed10 layer.

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 15** 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 15. Grad-CAM results for Grape Black Rot

Figure 16 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].

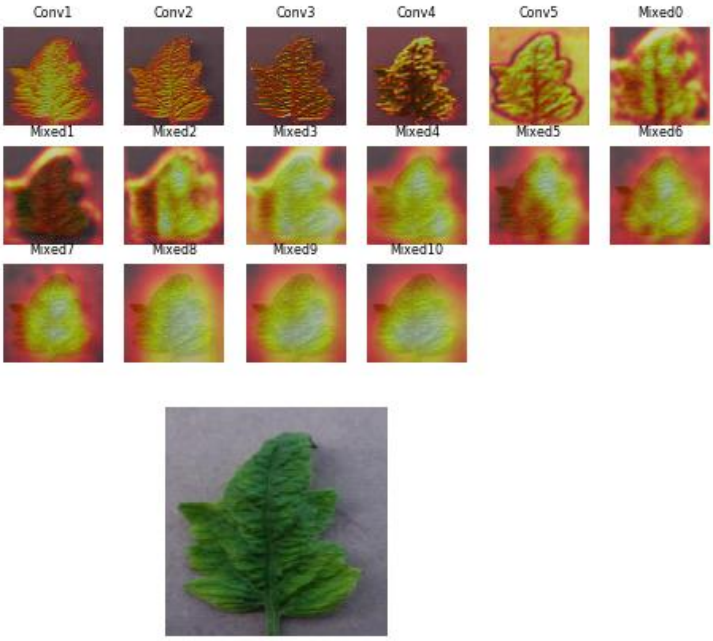

Figure 16. Grad-CAM results for Tomato Yellow Leaf Curl Virus

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.

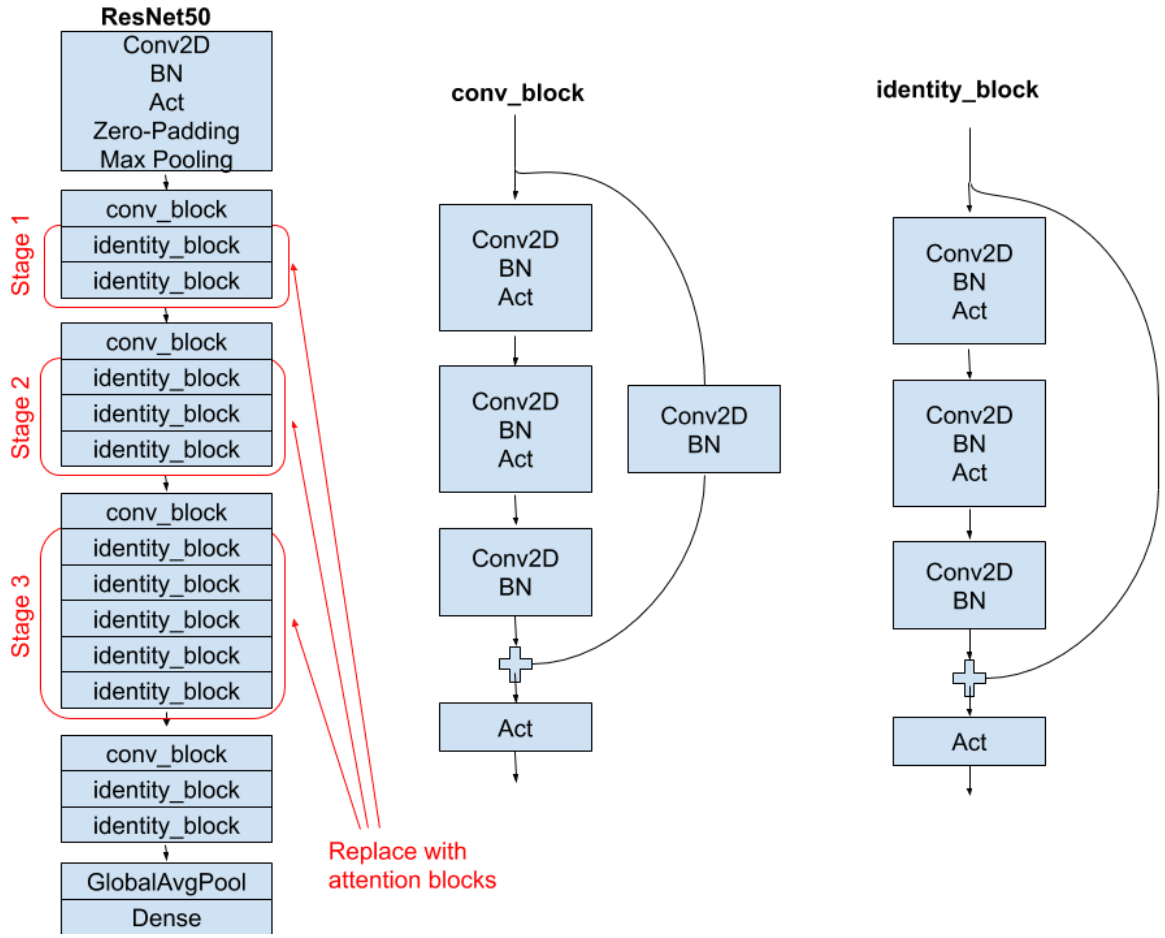

Figure 17. Proposed changes to ResNet50

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**.

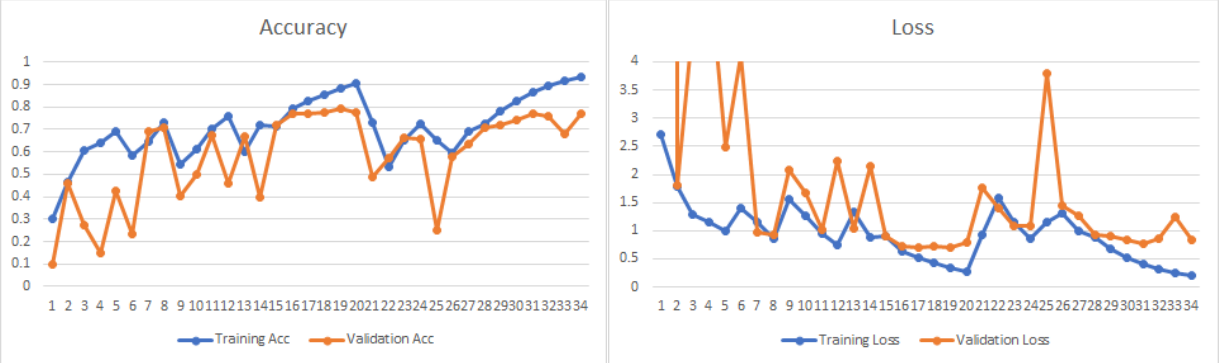

Figure 18. ResNet50 model (3 attention blocks) results. (Left) Training and Validation accuracy, (Right) Training and Validation loss

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.

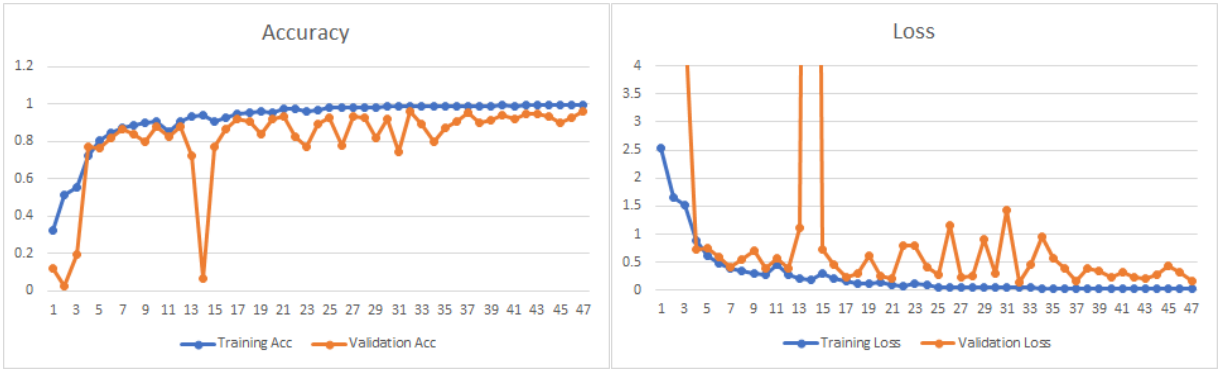

Figure 19. ResNet50 model (3 attention blocks, 3rd stage removed) results. (Left) Training and Validation accuracy, (Right) Training and Validation loss

CHAPTER V

APPLICATIONS

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. Two interfaces were developed for this purpose. The first is a mobile application and allows real-time classification of plant leaves. The second is a GUI and allows users to upload images of plant leaves for classification. Both interfaces function independently of each other.

Mobile Application

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.

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. Thus, to avoid any conversion errors, the model was retrained and directly saved as a

.tflite file. The model was trained on images from 2 of the 38 plant species: Corn (maize) healthy, and Grape Black Rot. As seen in **Figure 20**, the app successfully classifies this particular leaf as Corn (maize) healthy, with 99.95% accuracy.

Figure 20: Mobile Application

Graphical User Interface (GUI)

To further facilitate user interaction with the model, a Graphical User Interface (GUI) was also developed. 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.

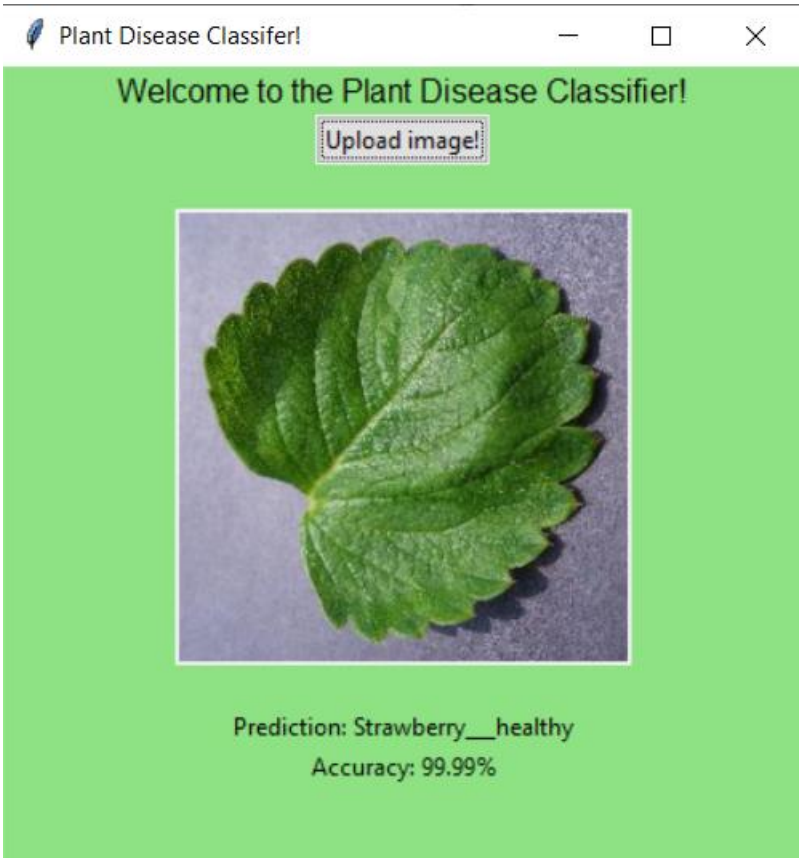

Figure 21: GUI Classification Example

CHAPTER V

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.

REFERENCES

[1] S. Mohanty, D. P., Marcel, and Hughes, “Using Deep Learning for Image-Based Plant Disease Detection,” *Frontiers*, 06-Sep-2016. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fpls.2016.01419/full. [Accessed: 05-Apr-2020].

[2] Y. Toda and F. Okura, “How Convolutional Neural Networks Diagnose Plant Disease,” *Plant Phenomics*, 26-Mar-2019. [Online]. Available: https://spj.sciencemag.org/plantphenomics/2019/9237136/. [Accessed: 05-Apr-2020].

[3] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, X. Tang, "Residual Attention Network for Image Classification," arXiv:1704.06904 [cs], April 2017.

[4] Medium. 2020. *Understanding And Coding A Resnet In Keras*. [Online] Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33 [Accessed 5 April 2020].

[5] F. Chollet et al., “Keras.” https://keras.io, 2015.

[6] M. Chetoui, “Grad-CAM- Gradient-weighted Class Activation Mapping,” *Medium*, 29-Mar-2019. [Online]. Available: https://medium.com/@mohamedchetoui/grad-cam-gradient-weighted-class-activation-mapping-ffd72742243a. [Accessed: 05-Apr-2020].

[7] “TensorFlow Lite: ML for Mobile and Edge Devices,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/lite. [Accessed: 05-Apr-2020].

[8] Mohanty, Sharada, PlantVillage-Dataset, 2016, Github Repository, https://github.com/salathegroup/plantvillage_deeplearning_paper_dataset

[9] Prijono, L., 2020. *Student Notes: Convolutional Neural Networks (CNN) Introduction*. [Online] Belajar Pembelajaran Mesin Indonesia. Available: <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/> [Accessed 5 April 2020].

[10] C. Olah, A. Mordvintsev, and L. Schubert, "Feature Visualization," *Distill*, 28-Aug-2019. [Online]. Available: https://distill.pub/2017/feature-visualization/#enemy-of-feature-vis. [Accessed: 05-Apr-2020].

APPENDIX

All codes, images, and model weights can be found on my GitHub repository:

https://github.tamu.edu/skumar55/Senior_Research_Project