IMAGE-BASED CROP DISEASE AND SEVERITY CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK

MAHDER DEMEKE DESALEGN

A Thesis Submitted To

The Department of Computer Science and Engineering

School Of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September, 2022

Adama, Ethiopia

IMAGE-BASED CROP DISEASE AND SEVERITY CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK

MAHDER DEMEKE DESALEGN

ADVISOR PROF. YUN KOO CHUNG

A Thesis Submitted to

The Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September, 2022

Adama, Ethiopia

Approval Sheet

I hereby certify that the recommendation and suggestion given by the proposal review Committee are appropriately incorporated into the final thesis proposal entitled "**Image-based Crop Disease and Severity Classification Using Convolutional Neural Network**" by Mahder Demeke.

Prf. Yun Koo Chung _____ _____

Major Advisor/Supervisor Signature Date

We, the undersigned, members of the Board of Examiners of the thesis by Mahder Demeke have read and evaluated the thesis entitled "**Image-based Crop Disease and Severity Classification Using Convolutional Neural Network**" and examined the candidate during the open defense. This is, therefore, to certify that the thesis is accepted for partial fulfilment of the requirement of the degree of Master of Science in Computer Science and Engineering

_____ _____ _____

Chairperson Signature Date

_____ _____ _____

Internal Examiner Signature Date

_____ _____ _____

External Examiner Signature Date

Finally, approval and acceptance of the thesis are contingent upon the submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____ _____ _____

Department Head Signature Date

School Dean	Signature	Date
Office of Postgraduate Studies,	Dean Signature	Date

Approval of Board of Reviewers

We, the undersigned, members of the Board of Reviewers of the final open defense by Mahder Demeke have read and evaluated the thesis entitled "**Image-based Crop Disease and Severity Classification Using Convolutional Neural Network**" and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirement of the Degree of Master of Science in Computer Science and Engineering

Name	**Signature**	**Date**
Mahder Demeke	_____	_____
Name of Student	Signature	Date
Prof. YunKoo Chung	_____	_____
Advisor	Signature	Date
_____	_____	_____
External Examiner	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Head	Signature	Date

Declaration

I hereby declare that this Master Thesis entitled "**Image-based Crop Disease and Severity Classification Using Convolutional Neural Network**" is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation

Mahder Demeke	_____	_____
Name of the student	Signature	Date

Recommendation

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "**Image-based Crop Disease and Severity Classification Using Convolutional Neural Network**" prepared under my guidance by **Mahder Demeke** submitted in partial fulfillment of the requirements for the degree of Masters of Science in **Computer Science and Engineering.**

Therefore, I recommend the submission of a revised version of the thesis to the department following the applicable procedures.

Prof. YunKoo Chung	_____	_____
Major Advisor	Signature	Date
_____	_____	_____
Co-advisor	Signature	Date

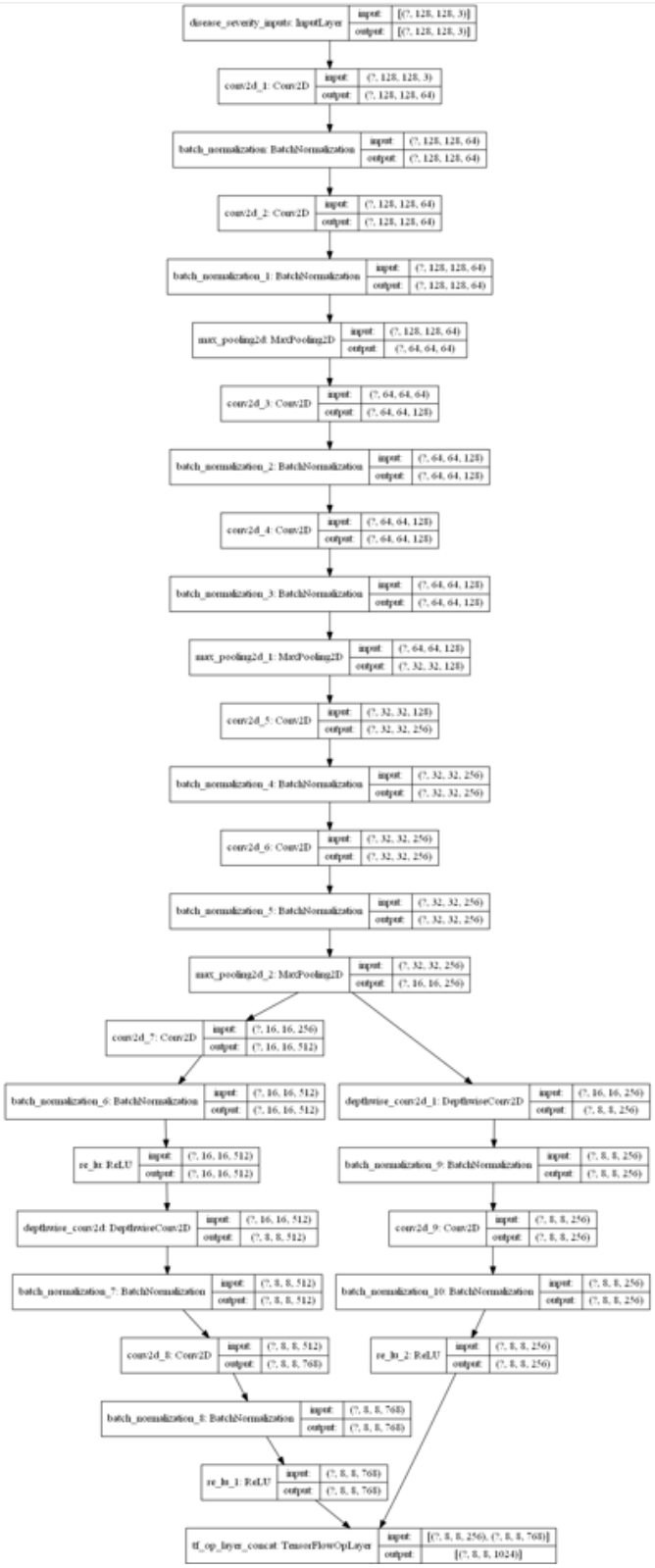
ACKNOWLEDGEMENT

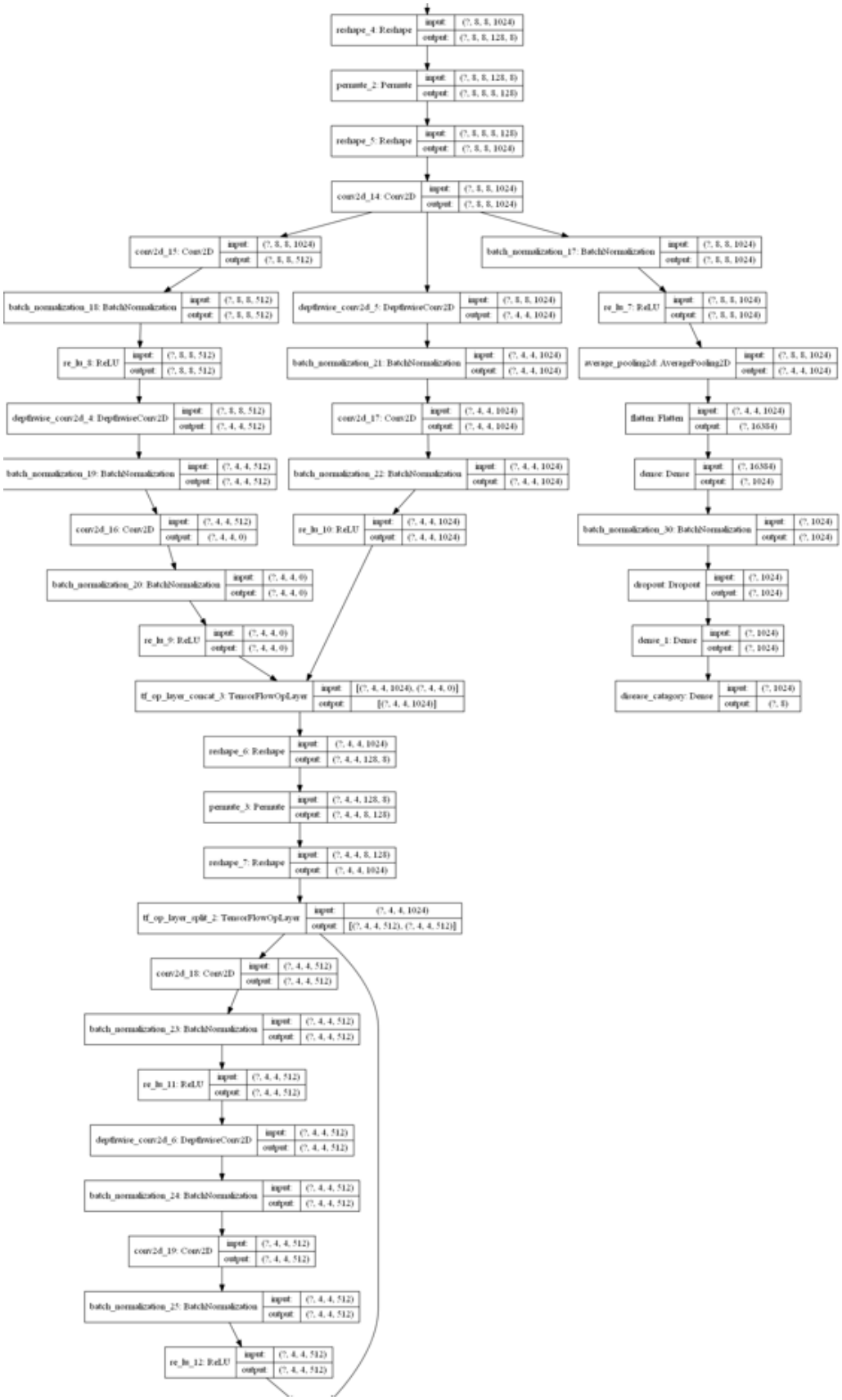
First and foremost, I would like to thank God for all his blessings throughout my life. I would like to express my deepest gratitude to my advisor, Prof. YunKoo Chung, for his invaluable patience and feedback. I am also extremely grateful to our coordinator, Dr. Mesifn Abebe, for his kindly provided guidance and supportive feedback. Furthermore, this research would be difficult without the invaluable support from Tagel Aboneh and Mosisa Desalegn by providing their dataset for free. Moreover, I am extremely grateful to EIAR and its branch employees. I especially appreciate the pathogen and breeding department members. I could not have undertaken this journey without them.

Thanks should also go to my research assistant Ashenafi Gemechu who helped me to develop the research view. And a sincere thanks to my classmates for their moral support. Lastly, I need to mention my family's invaluable appreciation and moral support, especially my dad for his informative experience on the agriculture department concept.

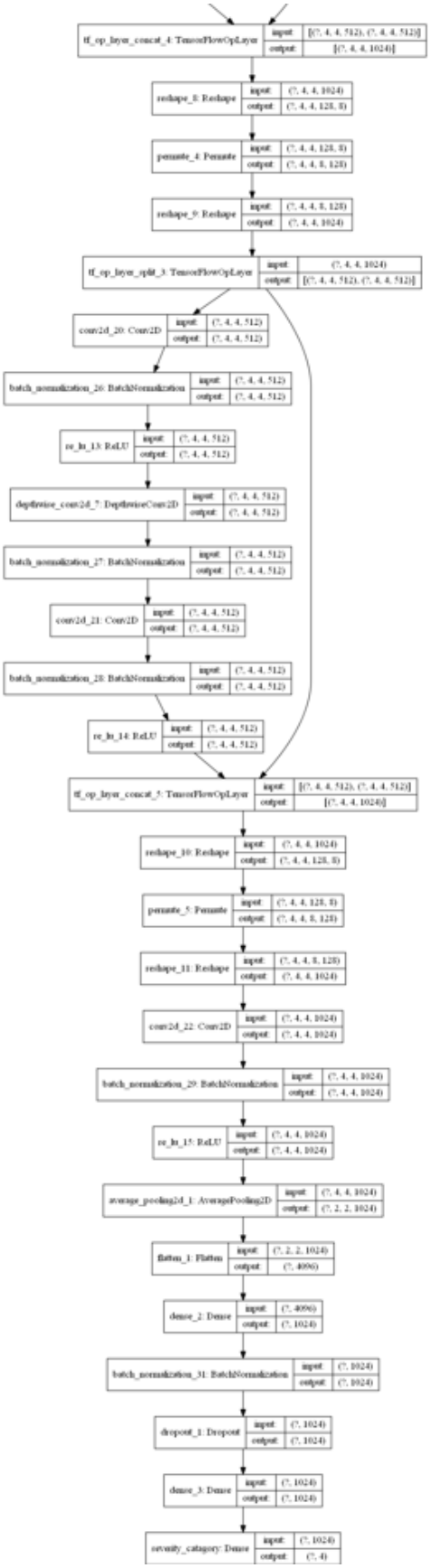
Table of Contents

severity	disease	file	Prediction result
0	0	/home/cv-sig/Research Data/h/0_0_level0healthy...	disease pred: [0] severity pred [0]
0	4	/home/cv-sig/Research Data/h/0_4_level0health...	disease pred: [4] severity pred [0]
1	1	/home/cv-sig/Research Data/h/1_1_level1commo	disease pred: [1] severity pred [1]
1	2	/home/cv-sig/Research Data/h/1_2_level1blightM...	disease pred: [2] severity pred [1]
3	5	/home/cv-sig/Research Data/h/3_5_level3leafRus...	disease pred: [5] severity pred [3]
3	3	/home/cv-sig/Research Data/h/3_3_level3GLSMaiz...	disease pred: [3] severity pred [3]
2	7	/home/cv-sig/Research Data/h/2_7_level2yellowR...	disease pred: [7] severity pred [2]
2	7	/home/cv-sig/Research Data/h/2_7_level2yellowR...	disease pred: [7] severity pred [3]
3	6	/home/cv-sig/Research Data/h/3_6_level3steamRu.	disease pred: [6] severity pred [3]
2	6	/home/cv-sig/Research Data/h/2_6_level2GLSMaiz	disease pred: [6] severity pred [2]
2	1	/home/cv-sig/Research Data/h/2_1_level2commonl	disease pred: [3] severity pred [2]


disease_severity_inputs: InputLayer
input: [(?, 128, 128, 3)]
output: [(?, 128, 128, 3)]
conv2d_1: Conv2D
input: (?, 128, 128, 3)
output: (?, 128, 128, 64)
batch_normalization: BatchNormalization
input: (?, 128, 128, 64)
output: (?, 128, 128, 64)
conv2d_2: Conv2D
input: (?, 128, 128, 64)
output: (?, 128, 128, 64)
batch_normalization_1: BatchNormalization
input: (?, 128, 128, 64)
output: (?, 128, 128, 64)
max_pooling2d: MaxPooling2D
input: (?, 128, 128, 64)
output: (?, 64, 64, 64)
conv2d_3: Conv2D
input: (?, 64, 64, 64)
output: (?, 64, 64, 128)
batch_normalization_2: BatchNormalization
input: (?, 64, 64, 128)
output: (?, 64, 64, 128)
conv2d_4: Conv2D
input: (?, 64, 64, 128)
output: (?, 64, 64, 128)
batch_normalization_3: BatchNormalization
input: (?, 64, 64, 128)
output: (?, 64, 64, 128)
max_pooling2d_1: MaxPooling2D
input: (?, 64, 64, 128)
output: (?, 32, 32, 128)
conv2d_5: Conv2D
input: (?, 32, 32, 128)
output: (?, 32, 32, 256)
batch_normalization_4: BatchNormalization
input: (?, 32, 32, 256)
output: (?, 32, 32, 256)
conv2d_6: Conv2D
input: (?, 32, 32, 256)
output: (?, 32, 32, 256)
batch_normalization_5: BatchNormalization
input: (?, 32, 32, 256)
output: (?, 32, 32, 256)
max_pooling2d_2: MaxPooling2D
input: (?, 32, 32, 256)
output: (?, 16, 16, 256)
conv2d_7: Conv2D
input: (?, 16, 16, 256)
output: (?, 16, 16, 512)
batch_normalization_6: BatchNormalization
input: (?, 16, 16, 512)
output: (?, 16, 16, 512)
depthwise_conv2d_1: DepthwiseConv2D
input: (?, 16, 16, 256)
output: (?, 8, 8, 256)
re_lu: ReLU
input: (?, 16, 16, 512)
output: (?, 16, 16, 512)
batch_normalization_9: BatchNormalization
input: (?, 8, 8, 256)
output: (?, 8, 8, 256)
depthwise_conv2d: DepthwiseConv2D
input: (?, 16, 16, 512)
output: (?, 8, 8, 512)
conv2d_9: Conv2D
input: (?, 8, 8, 256)
output: (?, 8, 8, 256)
batch_normalization_7: BatchNormalization
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
batch_normalization_10: BatchNormalization
input: (?, 8, 8, 256)
output: (?, 8, 8, 256)
conv2d_8: Conv2D
input: (?, 8, 8, 512)
output: (?, 8, 8, 768)
re_lu_2: ReLU
input: (?, 8, 8, 256)
output: (?, 8, 8, 256)
batch_normalization_8: BatchNormalization
input: (?, 8, 8, 768)
output: (?, 8, 8, 768)
re_lu_1: ReLU
input: (?, 8, 8, 768)
output: (?, 8, 8, 768)
tf_op_layer_concat: TensorFlowOpLayer
input: [(?, 8, 8, 256), (?, 8, 8, 768)]
output: [(?, 8, 8, 1024)]


reshape_4: Reshape
permute_2: Permute
reshape_5: Reshape
conv2d_14: Conv2D
conv2d_15: Conv2D
batch_normalization_17: BatchNormalization
batch_normalization_18: BatchNormalization
depthwise_conv2d_5: DepthwiseConv2D
re_lu_7: ReLU
re_lu_8: ReLU
batch_normalization_21: BatchNormalization
average_pooling2d: AveragePooling2D
depthwise_conv2d_4: DepthwiseConv2D
conv2d_17: Conv2D
flatten: Flatten
batch_normalization_19: BatchNormalization
batch_normalization_22: BatchNormalization
dense: Dense
conv2d_16: Conv2D
re_lu_10: ReLU
batch_normalization_30: BatchNormalization
batch_normalization_20: BatchNormalization
dropout: Dropout
re_lu_9: ReLU
dense_1: Dense
tf_op_layer_concat_3: TensorFlowOpLayer
divorce_category: Dense
reshape_6: Reshape
permute_3: Permute
reshape_7: Reshape
tf_op_layer_split_2: TensorFlowOpLayer
conv2d_18: Conv2D
batch_normalization_23: BatchNormalization
re_lu_11: ReLU
depthwise_conv2d_6: DepthwiseConv2D
batch_normalization_24: BatchNormalization
conv2d_19: Conv2D
batch_normalization_25: BatchNormalization
re_lu_12: ReLU

.....109


tf_op_layer_concat_4: TensorFlowOpLayer
reshape_8: Reshape
permute_4: Permute
reshape_9: Reshape
tf_op_layer_split_3: TensorFlowOpLayer
conv2d_20: Conv2D
batch_normalization_26: BatchNormalization
re_lu_13: ReLU
depthwise_conv2d_7: DepthwiseConv2D
batch_normalization_27: BatchNormalization
conv2d_21: Conv2D
batch_normalization_28: BatchNormalization
re_lu_14: ReLU
tf_op_layer_concat_5: TensorFlowOpLayer
reshape_10: Reshape
permute_5: Permute
reshape_11: Reshape
conv2d_22: Conv2D
batch_normalization_29: BatchNormalization
re_lu_15: ReLU
average_pooling2d_1: AveragePooling2D
flatten_1: Flatten
dense_2: Dense
batch_normalization_31: BatchNormalization
dropout_1: Dropout
dense_3: Dense
severity_category: Dense

LIST OF TABLES

LIST OF FIGURES

LIST OF SAMPLE CODE

LIST OF ABBREVIATIONS

AdaGrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
BNMRC	Bako National Research Center
BPNN	Back-Propagation Neural Network
CL	Computational Learning
CNN	Convolutional Neural Network
DCNN	Deep Convolutional Neural Network
DL	Deep Learning
EIAR	Ethiopian Institute Of Agricultural Research
FCL	Fully Connected Layer
F-CNN	Fully- Convolutional Neural Network
FFNN	Feed Forward Neural Network
GDP	Gross Domestic Product
GLS	Gray Leaf Spot
GPU	Graphics Processing Unit
IoT	Internet Of Things
ML	Machine Learning
NLP	Natural Language Processing
ReLU	Rectified Linear Unit
RF	Random Forest
RGB	Red Green Blue
RMSprop	Root Mean Square Prop
RP	Recognizing Pattern
S-CNN	Segment-Convolutional Neural Network
SNNPR	South Nation Nationality People Representative
SVM	Support Vector Machine
VGG	Visual Geometry Group

ABSTRACT

Crop protection-related problems are the main challenges in the agriculture sector since they have a large impact on crop yield reduction. In particular, disease is one of the major protection problems that result in food insecurity and agricultural system unsustainability. To prevent crop disease from spreading at the earliest stage, modern technology is preferable to the traditional approach. In this study, a modified Convolutional Neural Network model is proposed to identify and classify disease types and severity levels on Wheat and Maize crops by extracting leaf features. Six types of disease classification and three levels of severity are taken into consideration in addition to healthy leaf for each crop. The proposed model contains a modified version of the VGG16 architecture and a pair of channel shuffle blocks as an auxiliary structure, which enables it to perform multi-labeled classification with a single model. It has been trained through backpropagation and optimized with the Adam optimizer using field crop RGB image training datasets. To provide a more practical approach, 65% of the data was gathered from research centers, and the remainder was gathered from private data sources in combination with greenhouse images with manual farm field capturing. Also, to expand the collected data set and the variation of data, different augmentation techniques are applied. A highly accurate result has been achieved after experimenting with different regularizers and different hyper parameters. The proposed model is composed of the combination of the VGG structure and ShuffleNet structure and it achieves 0.98 accuracy for both disease and severity classification. The model shows its robustness by showing a performance of high accuracy with low computational cost compared to VGG model and better practicability since it considers real field data. The multi-labeling classification ability makes the model multi-tasking, with reduced computational time and data simultaneously.

Keywords: *Deep Learning, Convolutional Neural Networks, Disease classification, Detection, Severity, Disease identification, Severity features, Wheat Rust, Maize disease, Leaf disease, Crop disease classification*

CHAPTER ONE

1. INTRODUCTION

1.1.Background

By combining engineering, artificial intelligence, and a wealth of data, technology is becoming the primary method for resolving the various issues that people encounter in their daily lives. This technological development opens up a vast arena for research in numerous industries, including farm automation. Agriculture is the science of raising animals and cultivating land to create food. Additionally, it is a significant economic activity that offers employment opportunities, food, and raw materials like fuel, fiber, and medication to a significant portion of the world's population. In order to fulfill the rising demand for food and agricultural raw materials, which is expected to reach 9 billion people by 2050, agricultural production would need to expand by 25%.Automation is a big solution for farmers' challenges such as crop diseases, weed management, pesticide control, and drainage facilities (Miceli, P. A., Blair, W. D., & Brown, M. M, 2018). Numerous researchers continue to use a variety of strategies to better this industry.

Plant disease is a dynamic process of biotic and abiotic entities that interface with the normal function of a plant over a while. The cause of plant disease due to infections or biotic are living parasitic organisms, hosts, and pathogens like fungi, viruses, and nematodes. It is also caused by environmental conditions such as nutrition, moisture, toxic chemicals, and meteorological conditions. Disease management in agriculture has different approaches usually spraying anti-pathogen chemicals based on disease severity is one of them. Since Maize and Wheat are the most important highly cultivated cereal crop that is not able to achieve the expected yield production. Due to disease feeding the population by the current yield is challenging therefore, applying research on these crops is important.

Wheat is the second most important crop cultivated almost in all regions but yields less due to rusts such as a leaf, yellow, and stem caused by fungi (Olana, M. D., Rajesh Sharma, R., Sungheetha, A., & Chung, Y. K. , 2019). Also, Maize is an introduced strategic crop in

Ethiopia for food insecurity that mainly grows in SNNPR and Oromia regions in about a 1.77million hectares (EUBFE, 2015) some of the major diseases occurring in Maize crops in different production zone are shown below in

Table 1 disease affection in a different Maize production zone *(Twumasi-Afriyie, S., Zelleke, H., Yihun, K., Asefa, B., & Tariku, S., 2002)*

Production zones	**Elevation (m)**	**Rainfall (mm)**	**Disease**
Mid-altitude sub-humid	1000 – 1800	1000-1250	blight, rust, GLS, ear rot,
Moisture stress	500-1800	< 800	rust, blight
High altitude sub-humid	1800-2400	1200-2000	leaf blight, rust, grey leaf spot, ear rot
Low altitude sub-humid	< 1000	1200-1500	maize streak virus, grey leaf spot, rust

In particular, for vast agriculture fields, it is too expensive, time-consuming, and even crop-damaging to identify the disease and anticipate its severity level of it by expert observations. The alternative option for disease control is to spray fungicide or bactericide by anticipating the typical disease, but excessive fungicide and herbicide use results in crops that are spoilt, damaged, and unhealthy, and it may also pollute the environment. Researchers were prompted to provide a solution by utilizing a novel technology due to the limitations of the standard detection of plant disease.

To create disease detection models, ML methods like Bayes classifier, Random Forests, Artificial Neural Network, Support Vector Machine, and K-Nearest Neighbor have been used (Vijaypal Singh Dhaka, Sangeeta Vaibhav Meena, Geeta Rani, Deepak Sinwar, Kavita, 2021).A branch of machine learning called Deep learning becomes the modern era tool in the 21st century which extends Machine Learning (ML) concept by studying statistical models called deep neural networks. It can learn complex and hierarchical representations from raw data by itself also it is a powerful tool in image recognition.

Therefore research on deep learning technology to determine whether the crop is affected by a disease and even classify the type of disease and severity is required to handle this problem more effectively. Therefore, I suggest a CNN model that takes into account all leaf learnable features and employs it to apply it to the detection and identification of disease on a sample of two crops dataset. The suggested model can be used to categorize diseases and rate their severity. As a result, a modified VGG16 model is suggested in this study to carry out two tasks within a single model and boost recognition accuracy with lower computational cost. The majority of the training data are collected from a field data source, and some are samples from the greenhouse to represent the variation of data from different sources. The goal of this study was to improve the standard identification models which have been trained with laboratory images and made for a single classification task.

Deep learning is a subfield of machine learning that study deep neural network which is a statistical model (Miceli, P. A., Blair, W. D., & Brown, M. M, 2018). Before reaching at CNN, research on building a feasible neural network had a significant history. For image detection and recognition, CNN is a viable neural network that can extract features and translation invariance. The extractor module has processing layers that may be taught to extract particular information from the image. The categorization module is then sent the retrieved characteristics. Due to its invariability to translation and fewer trainable parameters, it became state of the art for image detection .Therefore Convolutional Neural Networks are one of the primary study fields now being used to enhance the efficiency of agricultural automation, including disease identification and severity prediction. (Vijaypal Singh Dhaka et al., 2021).

1.2.Motivation of the Study

The fact that Ethiopia is a country with undeniable natural resources but that farmers still struggle with food insecurity at the present level of crop production because of various difficulties, specifically drought, pests, and crop diseases (Mohamed, A. A. , 2017) , served as the inspiration for this research topic. Due to this, the administration is battling food insecurity in the nation and is looking for aid from industrialized nations, which has an effect on economic and diplomatic relations. Due to a lack of technologically assisted agricultural production to increase yield production, it was unable to produce the desired results. In order to get the desired

results, feed the growing population, and boost exports, deep learning is one of major approach to be implemented. CNN will specifically be used to spot disease early on and prevent it from spreading to the entire area.

The inability of farmers to identify the specific type and level of crop disease, as well as the ongoing need for professional scouts and pathologists in each farm area, has a negative impact on addressing crop disease early, before severe crop damage occurs. These factors encouraged me to conduct research in this field and contribute to my position by developing automatic disease classification and severity identification model, which are the primary difficulties in plant protection for the nation's major crops. So that it can be useful in order to facilitate expert jobs and statistical data analysis in agricultural research institutions, as well as to benefit small and large farm field owners.

1.3. Statement of the Problem

Identification of diseases and classifying their severity are very essential in agricultural production. It is one of the most crucial preventive strategies for lowering crop yield losses caused by severe illness. Different crop leaf diseases result in different symptoms. Because of this, it presents challenges to inexperienced farmers as well as professional scouts, necessitating repeated chemical treatment. Therefore, a disease classification system is needed to automatically classify the type and severity of the disease at an early stage, reducing the need for skilled pathologists in every farm region, which is expensive.

Even though several researchers are employing various strategies to address this issue by incorporating image analysis, deep learning, and machine learning, the computing cost of their algorithms and their evaluation results are inefficient due to the high number of parameters and the difficulty of being implemented in low-end devices. During tests using actual field data, performance deteriorates because the majority of related research on disease categorization and severity prediction is solely focused on a less varied input. They were unable to cope with varying illumination conditions, leaf occlusions, species diversity, and various growth stages. Diverse real-world data should be taken into account when training the model. However, due to the usage of a laboratory image dataset, the training data that was being used by multiple researchers had a simple or plain background. However, it might not be as straightforward as a

training image dataset in the actual world. Consequently, the necessity for a reliable categorization model is still being investigated.

Additionally, the majority of the research in this field concentrates only on one of the tasks of disease classification or severity classification. The conventional method of implementation takes multiple works to perform the two classifications, which results in high training computational and time costs. A multi-labeling approach is necessary to prevent this issue since it allows for the reduction of the training time and computational resource needs for the two tasks separately. Therefore, a modified model that can execute multi-labeling with a lower computational cost and robustness with the variation in real-world data is required, as opposed to keeping with the prior models. A modified methodology for image-based disease and severity categorization using CNN was put out to address these issues.

1.4. Research Questions

The following three research questions will be addressed by this study.

RQ1: What would be the effect of the variation of real field data on classification model performance?

RQ2: How to reduce the computational expense of the classification model while improving disease and severity prediction?

RQ3: What would be the performance of the model on classifying disease and severity occurrence simultaneously?

1.5. Objectives of the Study

1.5.1. General objective

The general objective of this study is to build an efficient multi-labeling disease and severity classification model for Maize and Wheat crops using a Convolutional Neural Network.

1.5.1.1. Specific Objectives

The following are the study's specific goals that must be attained in order to fulfill its overall purpose.

- To study relevant field concepts and scientific studies on disease and severity classification for wheat and maize crop
- To prepare and organize a local dataset using various data sources for three different types of diseases affecting wheat and maize separately, each with three levels of severity
- To analyze the organized input image dataset with different preprocessing and augmentation techniques for enhanced image output.
- To design and optimize a multi-labeling CNN-based model with an emphasis on higher performance and less computational expense.
- To assess the performance of the proposed model and the baseline work by converting VGG 16 to a multi-label classification model

1.5.2. Significance of the Study

This study will offer a variety of advantages to various stakeholders as well as the research community. It can also be used for long-term study and investment in the many agricultural areas. It can also be applied in the different agricultural sectors for investment and even for extended research. A few of the applications are;

- It can be used by wheat and maize mandated agricultural institutions to implement agricultural automation and statistical disease analysis in various farm fields.
- This study could advance long-term Agro technology investigations into disease and severity classification.
- Future applications of the idea could include significant farm field investments and even private farm field owners.

1.6.Scope of the Study

The objective of this research is to develop and apply a model that can identify disease, characterize the type of disease that is present, and assess the severity of the condition. To improve the outcome on some issues that result from decreased variability, the CNN model's architecture will be improved, as well as preprocessing methods. Additionally, with many tasks and minimal computing expense, it can resolve the narrowed disease categorization on a single crop to multiple crops.

1.7.Limitation of the Study

Due to the tight timeline required to conduct the research within the growing season for the crops, the majority of the leaf images used in this study were not directly taken. Instead, it was gathered from a secondary data source, which could affect how well the image dataset performs. A moderate outlier in the model's performance is also revealed by the fact that the data set, which was gathered from several research centers, has varying image quality. As a result, it will have a moderate negative impact on the performance of the model.

1.8.Paper Organization

The rest of this paper is organized as follows. CHAPTER TWO review the relevant literature on important concepts and elements that were used in the thesis and include related research done by various researchers. Then, CHAPTER THREE presents the methodology and materials used to accomplish the research task. CHAPTER FOUR discusses the proposed model while CHAPTER FIVE illustrates the implementation of the proposed model with some sample codes. Results found from the experiment and discussion of the result including research questions with their respective answers are provided in detail in CHAPTER SIX. Finally, the study's conclusion and future work are presented.

CHAPTER TWO

2. LITERATURE REVIEW AND RELATED WORK

2.1. Introduction

This chapter delves into the literature on the problem domain and the research that has been done in this area. The review includes a detailed description of the crops and disease analysis, as well as a discussion of the sector's challenges. Researchers' previous approaches to problem solving, such as ML, Deep learning, and CNN, are also covered.

2.2. Overview of Problem Domain

Crops are plants that are grown and harvested for food, such as grains or cereals, vegetables, and fruits. They can be classified in various ways (Crop - definition of crop by The Free Dictionary, 2022) according to sowing season, origin, and utility. They are divided into six categories based on their utility: food, feed, fiber, oil seed, ornamental, and industrial crops. (Types of Crops (6 Major Types) | Informed Farmers, 2022). Cereal crops, which are classified as food crops, are the focus of this study. Cereals, such as wheat, maize, barley, millet, and rice, are the primary source of carbohydrates consumed by the majority of the world's population (Macauley, H., & Ramadjita, T., 2015). They are the main stable foods for the majority of the population, and increased production is still required to meet the highest demand. Maize and wheat are the primary crops for storing food resources to meet the demands of the human diet by delivering essential nutrients and energy (Galanakis, C. M. (Ed.), 2018).

2.2.1. Wheat

Wheat is the most important food crop, primarily used to make bread flour. Wheat straw can also be used to make animal feed and packaging materials such as paper. It is also used in a variety of products, including baked goods, alcoholic beverages, and alcohol. Ethiopia has an average yield per hectare of 11.83. (CSA., 1990-2001.). Despite the fact that wheat is the most important crop in the world due to rising population, changing food preferences, and strong urbanization, there are a variety of traits influencing wheat yield production to achieve the

expected yield. One of the most serious threats is pests and disease. Wheat is susceptible to a variety of diseases and pests, the most common of which are caused by fungus, bacteria, viruses, and insects.

2.2.2. Maize

Maize is the other important crop which is originated in Central America (Dowswell, C. R., Paliwal, R. L., & Cantrell, R. P., 2019), was introduced to Ethiopia in the 1600s to 1700s (Haffangel, 1961). It is a widely-grown staple food crop that has the highest average yield per hectare which is 17.95 and is third in area coverage (CSA., 1990-2001.). It grows in a variety of agro-ecological zones and farming systems from high to low land, and from high to low rainfall (Mulatu, K., Bogale, G., Tolesa, B., Worku, M., Desalegne, Y., & Afeta, A. , 1993). Maize is a high-priority crop for food insecurity because of its wide adaptation, total production, and productivity (Worku, M., Tuna, H., Nigussie, M., Deressa, A., Tanner, D., & Twumasi-Afriyie, S., 2002) and provides numerous benefits, including human nutrients, feed components for livestock, and raw materials for many industrial products (Tichkule, S, G. , & Dhanashri, 2015). Because of its high nutritive value and ability to share, maize is susceptible to a variety of plant diseases. While this cereal disease is affected by rainfall, temperature, and soil type. Maize leaf disease has a significant impact on yield production, both in terms of quality and quantity. (Abate, T., Shiferaw, B., Menkir, A., Wegary, D., Kebede, Y., Tesfaye, K., ... & Keno, T., 2015)

2.3.Challenges of Cereal Production

Due to challenges in the production of those cereals attaining the required output is still far from reality. Some of the challenges are climate change, land degradation, persistent biotic and abiotic stress and less financial resource for research are the main one. High incidence of disease, insect pests, and weeds are the continuous problems raised by biotic and abiotic stress (Macauley, H., & Ramadjita, T., 2015).Pathogens such as fungi, bacteria, viruses, and nematodes are primarily responsible for cereal crop disease and pests. Managing this plant pathogen is difficult because it necessitates specialized equipment, training, or experience, and accurate diagnosis is difficult in some cases. They are difficult to identify due to their small size

and the presence of multiple types (Koike, S. T., Gaskell, M., Fouche, C., Smith, R., & Mitchell, J. , 2000). The following subtitles discuss some of the diseases that affect maize and wheat.

2.3.1. Maize diseases

i. **Grey Leaf Spots (GLS):** GLS, also known as *Cercospora* leaf spot is caused by *C. zeae-maydis,* a major maize disease. It grows in humid subtropical and temperate climates. It is one of the most serious threats to maize production, resulting in yield losses of up to 29.1%.For GLS, no major resistance genes are known. Lesions that begin as small, regular, elongated brown-gray necrotic spots growing parallel to the veins are a symptom of this disease. (Jeffers, 2004).

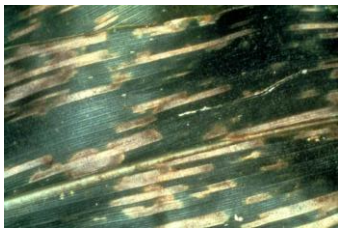


Figure 1 GLS symptom *(Jeffers, 2004)*

ii. **Turcicum Leaf Blight (TLB):** TLB is caused by *Exserohilum Turcicum* (Pass.) and affects maize from seedling to maturity. (Keno, T., Azmach, G., Gissa, D. W., Regasa, M. W., Tadesse, B., Wolde, L., ... & Mahabaleswara, S., 2018). Its incidence varies from 95 to 100% and can result in yield losses of up to 70% (Worku, M., Twumasi Afriyie, S., Wolde, L., Tadesse, B., Demisie, G., Bogale, G., ... & Prasanna, B. M., 2012). These diseases are most common in Ethiopia's warm and humid climate. Early symptoms include slightly oval, water-soaked, small spots on the leaves that develop into elongated, spindle-shaped necrotic lesions. It develops symptoms as the plant grows and leads to complete foliage burning (Jeffers, 2004).

Figure 2 Turcicum leaf blight symptom

iii. **Common Leaf Rust (CLR):** CLR is caused by *Puccinia Sorghi Schwein*, the other major maize disease in Ethiopia that can cause serious production problems across the major maize growing regions, resulting in yield losses ranging from 12 to 61% [18]. It has been observed in subtropical, temperate, and highland environments with high humidity. In our country, it is mostly found between 1,500 and 2,000 meters above sea level. In the early stages of infection, symptoms are dark brown or light orange colored postulates that turn black as the plant matures (Jeffers, 2004).

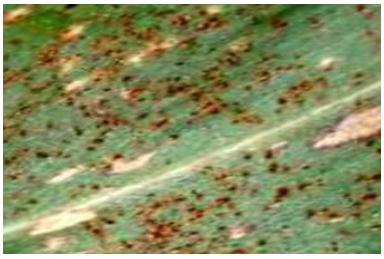


Figure 3 Maize Common Leaf Rust leaf

2.3.2. Wheat Disease

i. **Stem Rust:** *Puccinia Graminis f. sp. Tritici* causes stem rust (PGT). It is also known as black rust, and it occurs in all crop areas and primarily attacks the crop stem. Its yield loss varies according to location. The disease's symptoms include dark brown and reddish postulates on leaves and stems. Mature postulates may break the epidermis and cause the epidermis to turn white around the postulates. (Mehta, 2014).

Figure 4 Stem Rust symptom

ii. **Yellow Rust (Strip Rust):** Strip rust is another name for Yellow rust, the most destructive disease caused by the *Puccinia striiformis* fungus. Depending on the location and time, it may result in a yield loss of 5 to 50%. It can infect the entire plant except for the ground, but it usually affects the leaves and glumes. This rust causes *uredia* on leaves in a linear fusion and a yellow mass of color. When the plant matures, *telia* and *teliuto* spores are visible. (Mehta, 2014).

Figure 5 Wheat Yellow Rust Symptom

iii. **Leaf Rust:** Leaf rust, also known as brown rust, is the most common wheat disease in the world and is caused by the fungus *Puccinia triticina Ericks*. It can result in a 55% yield loss in susceptible wheat curvatures. This disease causes formation of *uredia* on both sides of the leaf and postulates that are orange in color and range from circular to slightly oblong. (Mehta, 2014).

Figure 6 Wheat leaf rust symptom

2.4.Severity Calculation in the Field

Severity and Incidence calculation: determined by observing the crop area showing disease symptom and the healthy portion of the crop. The percentage of incidence is calculated based on the affected section over the total area. Depending on the percentage of the affected section the crop disease incidence is assigned between an intervals of 0-3. (Blankson, D., Asare-Bediako, E., Frimpong, K. A., Ampofo, E., Taah, K. J., & Van der Puije, G. C., 2018)

$$\text{Incidence (\%)} = \frac{affected\ area}{total\ area} \times 100 \qquad \text{eq.(1)}$$

Table 2 Severity level assignment criteria's and description

No.	Incidence level	Description	Affection (%)	Severity
1	0 level/pure	No symptom	-	No infection
2	Level1	Some symptom or spots on young leaf	1-40	Mild infection
3	Level2	Moderate symptom coverage on young and old leafs	40-75	Moderate infection
4	Level3	Sever symptom and very severely affected and even dead leaf	75-100	Severe Infection

2.4.1. Disease Management

Numerous researchers are working in the area to find a solution to the common challenge of identifying diseases and their severity without the use of trained scouts, chemicals, or specialized tools, which could result in inconsistent results and inaccuracies (Tichkule, S, G. , & Dhanashri, 2015).Additionally, it has been attempted to prevent greater disease severity incidence (Alehegn, 2019). That was caused by an inefficient time needed to identify the disease by using an effective categorization system.

At the moment, one of the main study areas is the early detection and classification of agricultural diseases. It makes a significant contribution to monitoring vast field crops and early disease detection to minimize crop production loss. As a result, the agriculture industry places

great emphasis on it in both economic and technical aspects (Tichkule, S, G. , & Dhanashri, 2015) Agriculture productivity has improved as a result of technological innovation. Machine learning is one of the most promising technologies. Plant disease detection, agricultural product sorting and grading, biotechnology, medical diagnostics, industrial automation, yield prediction, and other applications are possible using machine learning. To detect disease categories and predict severity level, computer vision and machine learning are being integrated. As a result, several researchers are actively working on it (Alehegn, 2019). Due to the model's flexibility and less-than-satisfactory accuracy findings, it is still a ways off from being deployed. As a result, it is an open study issue with gaps that need to be found by subsequent scholars.

2.5.Machine Learning

Machine learning's capacity to learn from experience and make decisions based on example data has made it one of the preferred technologies for resolving human issues. Its range of practical applications now includes computational learning, natural language processing, and pattern recognition .This discipline enables a computer the ability to act as a human by learning from training data. Machine learning algorithms can be classified as:

- ***Supervised Learning:*** is an algorithm that takes a labeled example dataset as input and outputs information that deduces the input data's label. It is applicable to both regression and classification problems. The primary goal of this study is image classification, which is a supervised learning problem because labeled images are fed into the model (Burkov, 2019)to predict the category of both disease type and severity incidence. Deep neural networks and shallow neural networks are the two types of supervised learning.
- **Semi-Supervised Learning:** Semi-Supervised Learning is a mixture of supervised and unsupervised machine learning methods. Learning data is made up of a small percentage of labeled data and the remainder of unlabeled data (Burkov, 2019).
- **Unsupervised Learning:** is a powerful analysis tool that recognizes patterns and trends in unlabeled data through the use of an algorithm that learns from the data. Primarily useful for clustering datasets (Tadele, 2021).
- **Reinforcement Learning:** in this algorithm, the machine interacts with its surroundings and interprets their current state as a vector of features. It then continuously improves by using

trial and error with positive and negative feedback as a reward for changing the current state of the environment (Tadele, 2021).

2.5.1. ML In Image Classification

Classification is the process of labeling unlabeled examples in order to solve a problem. Image classification in machine learning can be accomplished by a classification algorithm taking labeled images as input and producing a model that can label unlabeled inputs (Burkov, 2019) where a label is a member of a finite set of categories that can be two (binary classification like healthy vs sick) or more than two (multi-classification). Image classification or recognition has long been a focus of Machine Learning research because it is the primary means by which computers interact with their surroundings (Tadele, 2021).

2.5.2. Machine Learning for Plant Disease Identification

Several researchers are currently using deep learning to detect and diagnose diseases. Plant disease identification based on categorization of leaf image has produced excellent results using several ML approaches. Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), Nave Bayes (NB), K-Nearest Neighbors (K-NN), Logistic Regression (LR), Artificial-Neural Network (ANN), and other traditional methods are used to address this issue (Ramesh, S., Hebbar, R., Niveditha, M., Pooja, R., Shashank, N., & Vinod, P. V., 2018), (Panigrahi, K. P., Das, H., Sahoo, A. K., & Moharana, S. C., 2020). The following is a discussion of the most influential and widely used machine learning algorithms in the classification of leaf diseases.

2.5.2.1. Random Forest

Random Forest is an ensemble learning supervised machine-learning algorithm that has several collections of decision trees and outputs the target class with the highest vote of each tree target output. It's used for classification, regression, and a variety of other tasks. The tree is built using both bagging and random feature selection. It generates an uncorrelated forest of trees, each with a more accurate prediction than the previous one. Random forests handle overfitting better than Decision trees and can handle both numerical and categorical data (Ramesh, S., Hebbar, R., Niveditha, M., Pooja, R., Shashank, N., & Vinod, P. V., 2018). Although it has been used to

classify leaf diseases by some researchers, competing machine and deep learning algorithms have outperformed it (Dong, M., Mu, S., Shi, A., Mu, W., & Sun, W., 2020).

2.5.2.2. Support Vector Machine

Support Vector Machine (SVM) is a binary classification-only machine-learning algorithm. To classify input in high-dimensional data, it constructs a set of hyperplanes. A hyperplane is a decision boundary that separates data points that lies between two classes. As a result, support vectors are closer to the hyperplane data points that determine position and orientation. SVM primarily deals with two-category data, but a new approach, one-versus-rest (one-vs-all) and one-vs-one, has recently addressed multi-class problems. SVM was used in leaf disease identification and outperformed Random Forest and logistic regression. (Das, D., Singh, M., Mohanty, S. S., & Chakravarty, S., 2020).

2.5.2.3. Decision Tree (DT)

DT is another supervised ML algorithm that is primarily used for classification but can also be used for regression problems. It solves problems by constructing a Decision tree model from a set of datasets that have been sorted by feature values. This model divides the dataset using a tree structure, in which each node represents a feature and each leaf node represents a class label. Despite the fact that it is prone to overfitting, its ability to visualize data for ease of understanding by humans makes it suitable for use in data mining for large data classification (Panchal, P., Raman, V. C., & Mantri, S, 2019) It has been used in the classification of wheat severity (Zhao, J., Xu, C., Xu, J., Huang, L., Zhang, D., & Liang, D., 2018).

2.5.2.4. KNN (K Nearest Neighbor)

Similarly, KNN is a supervised nonparametric machine learning technique that can be used for both regression and classification. This method is classified as lazy learning because it always uses all of the training data for classification and makes no assumptions or generalizations. It has high computational costs due to a lack of specialized training and summary. It is used in disease identification with excellent results in various research papers such as (Hossain, E., Hossain, M. F., & Rahaman, M. A., 2019), (Vaishnnave, M. P., Devi, K. S., Srinivasan, P., & Jothi, G. A. P., 2019).

2.5.3. Deep Learning

Deep learning is a subset of machine learning that involves training neural networks with a number of hidden or non-output layers. It is more powerful and adaptable than traditional machine learning and does not require human feature extraction. It mimics the structure of the human brain and excels at image classification tasks (Tadele, 2021). Deep learning has some advantages over regular ML, despite difficulties such as large data size, computational device, exploding gradient, and vanishing gradient. These issues are being addressed over time, with a greater accumulation of data currently being acquired from the worldwide internet, mobile devices, and individual researchers (Big Data). The computational complexity is being solved by improving the technology with a more capable GPU processor and online GPU platform. To overcome the problems of expanding and vanishing gradients, simple techniques such as gradient clipping, L1 or L2 regularization, and backpropagation are used (Burkov, 2019). Leaf disease and severity classification are two major research areas that benefit from it. The CNN algorithm is the most commonly used deep learning algorithm in this field.

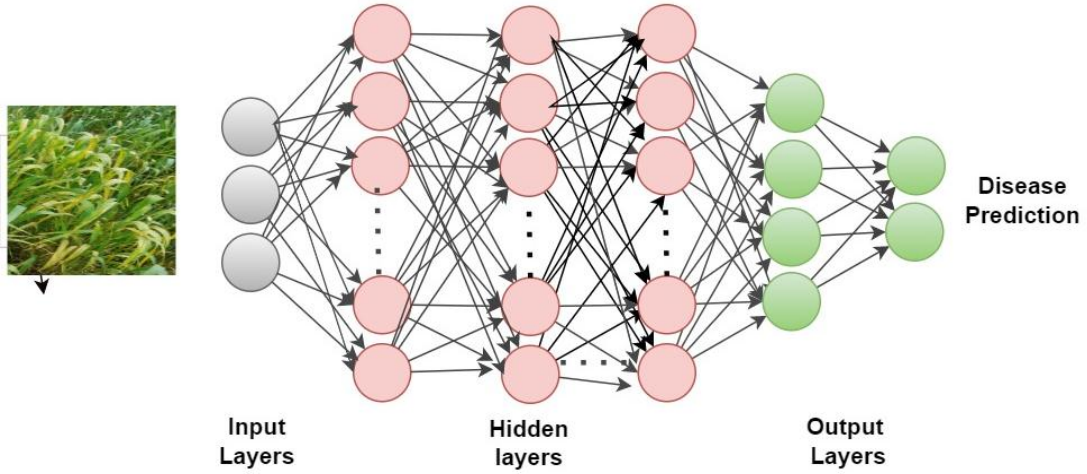


Figure 7 Description of Deep neural Network model

2.5.4. Convolutional Neural Networks (CNN)

CNN is a type of FFNN (Feed Forward Neural Network) that significantly reduces the number of parameters in the Deep Neural Network without sacrificing model quality. It has been used to process data with a grid pattern and has had great success in image processing. CNN is made

up of three kinds of layers: convolutional, pooling, and fully connected. To extract features, it takes the input image from the first input layer and passes it through intermediate layers (convolutional and polling layers). To extract features, convolutional layers generate defined filters with fixed sizes such as (3x 3, 5x5). The image and filter convolution operation generates an activation map, which is used as an input to the next layer. The pooling layer reduces the size of inputs in order to reduce model parameters while retaining distinguishing characteristics. Finally, CNN outputs fully connected layers that use extracted features for classification. The process of optimizing kernel parameters to reach the ground truth using backpropagation and gradient descent is known as model training. Convolutional neural networks, the most advanced technology in this field, have been widely used in the classification of plant diseases. They have solved the problems associated with traditional classification.

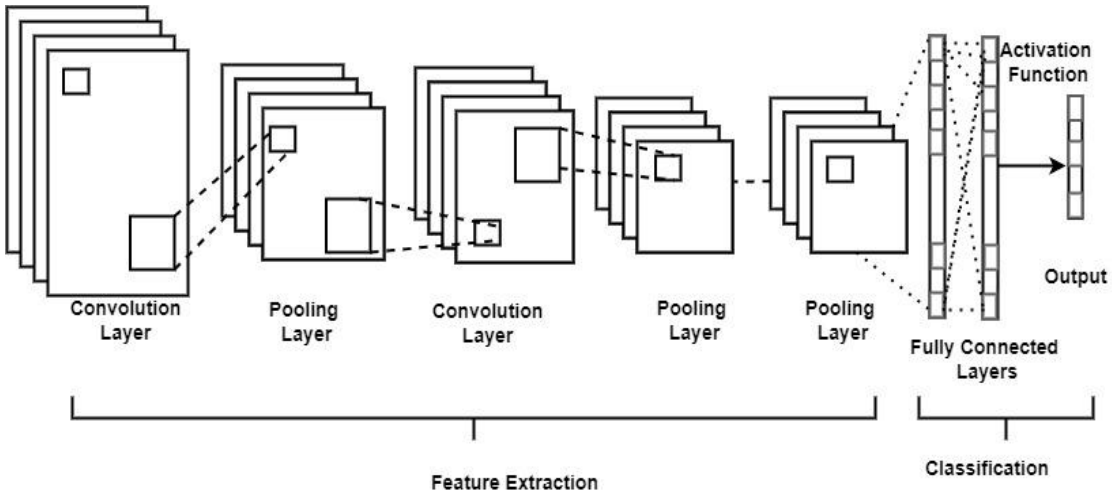


Figure 8 General Architecture of Convolutional Neural network

2.5.4.1. Optimizers

An optimizer is an algorithm that minimizes the loss function and maximizes the model's efficiency. It is dependent on the learnable parameters of the model, such as Weights and Biases, and updates them to reduce losses and provide the most accurate results possible. The loss function, which is based on backpropagation learning techniques, is used to assess its performance. Backpropagation is used by calculating the partial derivative of the loss with respect to the last layer weights recursively. The loss can be managed by cross-checking the trained output with the actual output in either classification or regression. There are numerous

popular optimizers used in deep learning model training. Gradient descent optimizers include SGD (Stochastic Gradient Descent), MBGD (Mini-butch gradient descent), Adagrad, Adam, and momentum (Ruder, S., 2016). Stochastic gradient descent optimizers are commonly used in deep learning, but they have a problem with tuning the parameter with its learning rate. However, Adam, Adagrad, and Adadelta handled the SGD issue. Batch Normalization is used to accelerate training and make learning easier by stabilizing the learning process and reducing the number of epochs after each convolution.

2.5.4.2. Training and Testing

Training is the process of taking the characteristic properties of an image class and forming a unique description for each particular class by adjusting weights through the backpropagation process. Testing is the process of classifying the test image into various classes based on the trained model. Model training is used for all classes in the classification task, whether binary or multi-class (Deepika, Jaswal, Sowmya. V, K.P., & Sowman , 2014). The training process's goal is to teach a model to learn and minimize the possible difference between the desired output and network output.

2.6.Related Work on Disease and Severity Classification

In agriculture, disease and severity classification is critical for increased yield production. As a result, it has emerged as an important artificial intelligence research topic for improved technical support. It also has a significant impact on early detection of leaf disease by detecting disease symptoms, allowing better preventive mechanisms to be implemented as soon as the symptom is visible. Over the last decade, the advancement of image-based approaches using Machine learning and then an improved sub-field of it, deep learning, has been applied to improve the agricultural sector. Using Deep Learning technology to improve crop quality and yield through automatic disease detection with a computer has become standard procedure. (Darwin, B., Dharmaraj, P., Prince, S., Popescu, D. E., & Hemanth, D. J. , 2021). However, for a variety of reasons, the achievement of this technological advancement is still insufficient to be applied in the real world in order to apply it at the earliest stage of crop development. The paragraphs that follow summarize some of the related research papers that were reviewed.

(Olana, M. D., Rajesh Sharma, R., Sungheetha, A., & Chung, Y. K. , 2019) Develop a deep learning model that can detect rust-affected wheat crops by modifying CNN on images of wheat crop dataset. The researcher worked with real-world datasets that had minor variations. The author implemented a binary classification of disease occurrence. After extracting features using color code segmentation to distinguish rust-affected crops from healthy crops, this study achieved 99.76% accuracy. However, the model was not complex enough to categorize the type of disease, so more research is required to perform disease and severity classification.

(Liang, Q., Xiang, S., Hu, Y., Coppola, G., Zhang, D., & Sun, W., 2019) The authors' goal was to create a network for detecting plant diseases and estimating their severity. They attempted to combine the shuffleNet block structure with the ResNet 50 block structure by adding ShuffleNetV2 blocks to each of the residual blocks. The goal of performing the combination was to reduce the computational cost while improving the model's performance in the three classification tasks. They were able to accomplish three tasks within a model by doing so. They used a collection of synthetic datasets from the AI Challenger Global AI Contest to train the model and applied a Generative Adversarial Nets (GAN)-based image augmentation technique. As a result, the ResNet 50 model performed three classifications with 0.99, 0.98, and 0.91 accuracy for species classification, disease diagnosis, and severity estimation, respectively. Despite the study's higher classification results, the classification accuracy for severity can still be improved. Another limitation of this study is that the use of synthetic dataset to train the model may reduce its robustness because it is not practicable in a real farm field.

Research on Deep Learning CNN Models to detect disease by segmenting images (Sharma, P., Berwal, Y. P. S., & Ghai, W, 2020) focuses on resolving a significant performance drop of most deep learning when tested on independent data by training models with segmented image data. They used image segmentation to extract relevant features, random transforms to the extracted images to vary brightness/contrast/blur, and images of different lighting conditions and backgrounds to solve problems related to similar backgrounds and lighting conditions. When tested on independent data, images after comparing the CNN model using full image (F-CNN) and segmented image (S-CNN) show that they were able to increase performance by more than two times. As a result, they were able to significantly boost the confidence of S-CNN over F-

CNN. However, S-CNN can fail in leave regions with complex disease symptoms and sensitive to segmentation quality.

(Pallagani, V., Khandelwal, V., Chandra, B., Udutalapally, V., Das, D., & Mohanty, S. P., 2019) Develop a smartphone app that uses computer vision technology and a Deep Learning method to identify 14 crop species and 26 crop diseases. Convolutional Neural Network model trained over four epochs on 53,306 image datasets. They were able to achieve a 99.24% accuracy. They created a usable model for the farmer by embedding a *Tensorflow.pb* weight file with a disease prediction classifier. The performance is still dependent on minor data variations, and the computational cost was extremely high.

Researchers also attempted to work on severity classification (Wang, G., Sun, Y., & Wang, J., 2017) by utilizing three pre-trained Convolutional Neural Network architectures, namely VGG16, ResNet50, and Inception-v3, which were trained from the ground up with a shallow network. VGG trained with transfer learning and exploration to achieve 90.4% accuracy in classifying the severity of four classes. Deep learning models were used in the study to diagnose disease severity using an image dataset. However, the study was limited by the use of laboratory images, which are not practical for training, and data scarcity also affects model accuracy.

(Sood, S., & Singh, H, 2020) Two wheat crop diseases were identified using the ResNet50 and VGG16 architectures. The dataset was compiled using Google and real-field crop images from farm areas in Ethiopia and Tanzania. Despite the fact that they faced overfitting due to a lack of data sets, they were able to achieve a better result with VGG16. However, a better approach to reducing overfitting should be used to improve its performance and prevent the model from degrading on independent data.

Also, a study on Maize disease identification using modified Cifar10 and GoogLeNet proposed classifying 9 disease types by reducing the number of parameters. Images were gathered from various open data sources, including Plant Village and the Google site (Zhang, X., Qiao, Y., Meng, F., Fan, C., & Zhang, M., 2018). The SGD optimizer was used to optimize the architecture, yielding an accuracy of 98.9%. GoogLeNet was completed after 50000 iterations. Despite its performance and accuracy results, the computational cost for training was very high, indicating that this work needs to be improved.

Some researchers (Xiang, S., Liang, Q., Sun, W., Zhang, D., & Wang, Y., 2021) argue that this area has successfully achieved higher accuracy and can be used in practical applications such as plant diseases and species recognition; however, others (Sharma, P., Berwal, Y. P. S., & Ghai, W, 2020) demonstrate that there is still a significant problem with models achieving high accuracy on disease detection but failing miserably on independent data due to low variability and similar backgrounds of the image dataset made it far from being used in real-world situations. Despite extensive research into plant disease detection, there are still challenges to overcome. Different lighting conditions, foliage occlusions, species variation, and crop growth stages are all unresolved issues. In terms of disease, tests on independent datasets fail due to the lower variability and background of image datasets (Vijayakumar, T., & Vinothkanna, R., 2020).

Table 3 Review of Literature

No.	Title	Problem	Objective	Methods	Result	Gap	Author
1	Applying a Deep Learning Approach for Wheat Rust Disease Detection	Rust affected wheat crop classification performance	To detect rust-affected wheat crop	• CNN model	• Score 99.76% accuracy of rust-affected crop detection	• Does not classify type of disease • Does not classify severity of the disease	(Olana, M. D., Rajesh Sharma, R., Sungheetha, A., & Chung, Y. K. , 2019)
2	PD2 SE-Net: Computer-assisted plant disease diagnosis and severity estimation network	Problem of single diagnosis task higher computational expense of previous models	To develop a computer-aided approach for disease and severity classification of various plants.	• ResNet 50 and ShuffleNet Structure.	• Score accuracies of 0.91 and 0.98 for severity and plant disease classification	• Used laboratory data set with less data variation and simple plain background. • Modest Accuracy for disease classification	(Liang, Q., Xiang, S., Hu, Y., Coppola, G., Zhang, D., & Sun, W., 2019)
3	Performance analysis of deep learning CNN	Solving a significant performance	Segmentation-based CNN model	• Full and segmented	• The performance increase	• Fail on leave regions containing	(Sharma, P., Berwal, Y. P.

	models for disease detection in plants using image segmentation	drop of most deep learning when tested on independent data		image with CNN model	more than two times when tested on independent data	complex disease symptoms • Sensitive to the quality of semantic segmentation.	S., & Ghai, W, 2020)
4	Dcrop: A deep-learning-based framework for accurate prediction of diseases of crops in smart agriculture	Limited availability of infrastructure for identification of crop disease	To identify 14 crop species and 26 diseases of crops using a smartphone	• Computer vision technology • ResNet	• Achieve an accuracy of 99.24 %	• Trained on unrealistic laboratory datasets • and costly	(Pallagani, V., Khandelwal, V., Chandra, B., Udutalapally, V., Das, D., & Mohanty, S. P., 2019)
5	An implementation and analysis of deep learning models for the detection of wheat rust disease	Disease on Wheat crop	To classify the wheat disease into three classes pure and two disease affection	• VGG16 • ResNet 50	• VGG16 achieved a higher performance of 98.33% ResNet 75.76%	• Less training dataset • Noisy data • Overfitting • High computational cost	(Sood, S., & Singh, H, 2020)

6	Identification of Maize Leaf Diseases Using Improved Deep Convolutional Neural Networks	Disease on Maize crop	To identify maize disease by comparing two architectures	• Improved GoogLeNet • Cifar10	• GoogleLeNet achieves 98.9% • Cifar10 98.8%	• Training cost was very high and the • Dataset do not represent the real field	(Zhang, X., Qiao, Y., Meng, F., Fan, C., & Zhang, M., 2018)
7	Automatic Image-Based Plant Disease Severity Estimation Using Deep Learning	Expensive cost and low efficiency of human disease assessment	To diagnose disease severity with an image dataset	• VGG16, • ResNet50, • Inception-v3	• VGG with transfer learning outperform	• Insufficient data • Unrealistic data • High computation cost	(Wang, G., Sun, Y., & Wang, J., 2017)

The majority of the related work reviewed used predefined CNN models without taking into account the computational cost or the context of the data set. As a result, the main gap to be identified is the lack of a modified version of DL and illumination conditions consideration. Because the severity of plant diseases varies over time, DL should be enhanced to classify diseases throughout the life cycle. However, many research papers do not include the crop's different growth cycles in their dataset.

Except for (Olana, M. D., Rajesh Sharma, R., Sungheetha, A., & Chung, Y. K. , 2019) almost all papers used the Plant Village dataset and synthetic plant leaf images. However, the dataset is limited in variety and has a plain background; to overcome this limitation, a real field image dataset with high variation and different growth stages should be considered. In order to include a diverse set of data, various augmentation techniques must be used. Additionally, optimizing image training pre-processing with CNN can improve real-world performance. By conducting various experiments, this study attempts to fill the specified gaps, detect plant diseases at an early stage, and classify the severity level based on the leaf symptom using a multi-labeling model.

CHAPTER THREE

3. RESEARCH METHODOLOGY

3.1. Chapter Overview

The process and method of developing disease and severity image classification are discussed in this chapter. Following the description of the research approach, these procedures are broken down into seven sub-sections. The first two sub sections describes the data collection and organization procedure, along with some related data organization techniques for multi labeling classification model. Then the next section demonstrate the preprocessing and augmentation techniques implemented. Continuing the discussion of augmentation techniques, the justification for model selection in the modeling subsection is revealed, followed by the tools and metrics used to implement the proposed model. Finally, the last sub-section concentrates on methods for combating overfitting.

3.2. Research Approach and Procedure

Based on the problem statement and questions to be answered, the research can be classified as an applied research. Given that, it aims to solve a specific problem that necessitates the use of a quantitative research approach with experiments. In order to draw a conclusion, a research design approach that includes data collection, processing, modeling, and experimentation is required. To carry out this research, several sequential procedures are carried out, beginning with the identification of the problem, followed by the development of the concept and the review of related work on it (Kumar, R., 2018). The gap identification and formulation of research questions is accomplished by reviewing literature. The methodology and tools for carrying out the research experiment are chosen based on the identified research questions. Simultaneously, data collection and organization have occurred. The proposed solution is then designed, implemented, and evaluated. The following figure depicts the overall general research approach taken, along with the main steps.

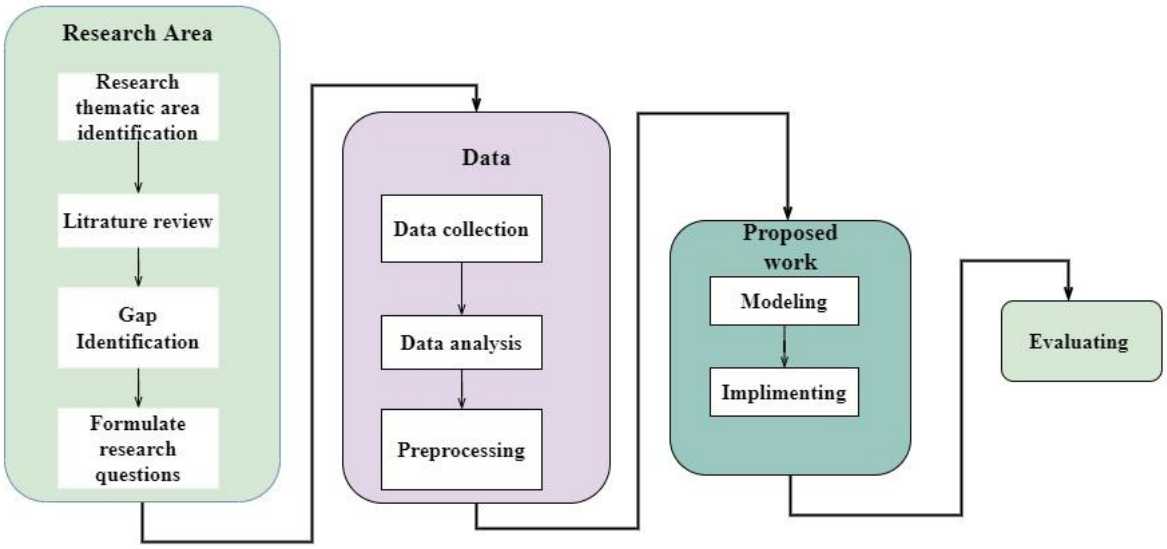


Figure 9 General approach of the research

3.3.Data Source

To collect training data from various sources, three main targets are considered in order to conduct this research. Because data size and quality are known to be the cornerstone resources for deep learning performance, it has been discovered that evaluating various possible resources is required. As a result, those three goals were thought to be the most efficient way to meet this requirement. The Ethiopian Institute of Agricultural Research (EIAR) was the primary data source, with secondary data sources including the Bako, Melkasa, Kullumsa, Debrezeit, and Ambo Agricultural Research Institutes. Despite the fact that there are numerous research centers, the Head of those research centers, EIAR, and six branches were chosen due to the mandate given by EIAR for maize and wheat research. The second target was some privately owned data from previous researchers (Olana, M. D., Rajesh Sharma, R., Sungheetha, A., & Chung, Y. K. , 2019) (Aboneh, T., Rorissa, A., Srinivasagan, R., & Gemechu, A., 2021) who were asked to voluntarily share their data after assessing their previous related work. The third type of data source is primary data, which is obtained directly from greenhouses prepared for cloning by the agricultural research community and represents some of the disease features.

3.4.Data Collection and Organization

The data collected and used to conduct this research is a combination of 90% secondary data and 10% primary data from various data sources, which include healthy and diseased wheat and Maize leaf images, as stated in the data source section. The collected data sets are organized into healthy and diseased crops, and each one is classified on a scale of 0 to 4. The data sets are organized through collaboration between the researcher and professional pathologists from the Debrezeit, Kulumsa, and Melkasa research institutes.

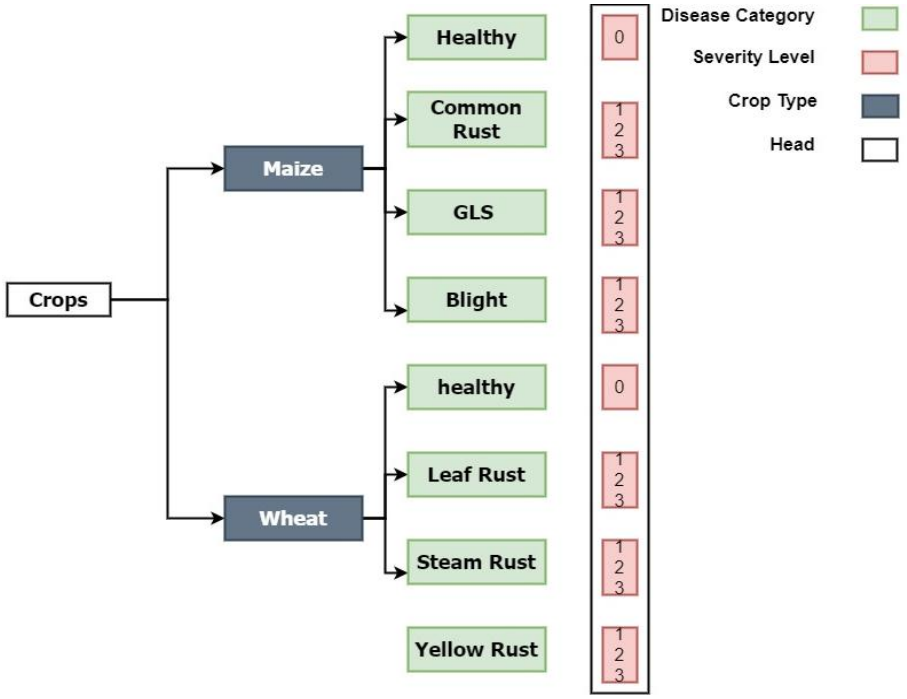


Figure 10 Data Set organization

The data used in this experiment have two crop species and both crops have healthy and three classes of diseases with corresponding severity levels, as shown in the diagram. The severity level is divided into three stages: early, middle, and series. As a result, the proposed model implemented multi-classification, which are disease and severity classifications, in order to obtain eight disease classifications and four severity classes, including the healthy one. Despite the fact that all possible data sources were used to collect the data, some of the classes have an unbalanced distribution, which necessitates the use of both under-sampling and over-sampling techniques. The total number of images collected from all data sources was 11,467 before

cleaning. Appendix A contains more information about the dataset, including statistical data. The final output obtained is six classifications of two crops diseases, with each crop containing three disease categories and three severity categories for corresponding disease categories, ignoring the healthy severity level of zero.

After reviewing the literature and interviewing pathology experts, five features were identified to classify the type of disease and its incidence: color, texture, morphology, edges, and symptomatic covered leaf area. Field images are considered the primary source in order to make the model robust against testing on real image data. Because the process of collecting and training the model only using primary real-world image data takes longer and requires waiting for the cultivation season, secondary data sources were used as the primary source. Some of the real-world images collected are listed below.

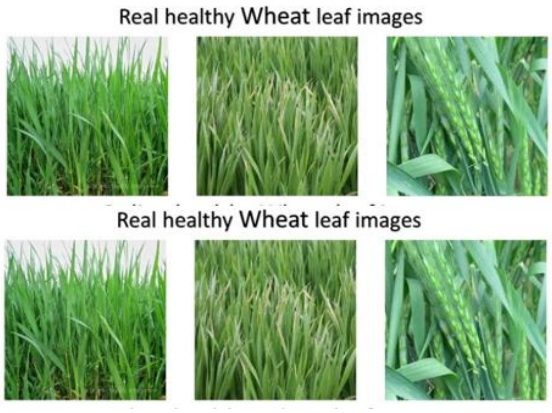


Figure 11 Healthy Wheat and Maize sample images

3.5.Preprocessing

The primary goal of image preprocessing is to convert collected data into suitable data quality before performing any experimental work, particularly in deep learning. Since these techniques differ depending on the purpose and type of data, the types of preprocessing techniques used in various studies vary. The quality of the images in this study case varies, which may have a significant impact on the model's performance. As a result, various preprocessing methods are used. Each of them is revealed step by step in the subsections that follow.

3.5.1. Data cleaning

The first step in implementing the image preprocessing was data cleaning. In data cleaning preparation of collected image data for training and testing of the model is takes place. Irrelevant images that consists of redundancy or duplication and images of lower quality were removed before it passes to further data analysis and fitting. Since merging multiple sourced data sets into one may result in redundancies and duplication, which may lead to model accuracy reduction, cleaning was one of the key preprocessing steps that have been implemented. In the case of this study, the collected data was the most tedious data, with high redundancy, irrelevant images, and poor organization. Filtering out all unwanted images, labeling mislabeled samples that are irrelevant to the research concept, and validating the data accuracy results in approximately 17% of the collected data being removed from the dataset. The total number of images collected before cleaning was 11,467; after cleaning, the total number of images dropped to 9,518.

3.5.2. Resizing Images

The first step was to resize the entire dataset in order to maintain the same size and aspect ratio. Most CNN models assume that the size of the input images is square. To make the dataset uniform in size, images were resized to 128x128 pixels. To reduce the computational complexity caused by training with large image sizes, a square of 128 image size is chosen. The type of image compression or extension is also altered by using the current standard image format, which is a .jpg file (Dhawan, Sachin, 2011). Because the jpg image format is preferable for realistic images since it stores smaller file sizes while maintaining acceptable quality than other formats (Abuzaher, M., & Jamil, A. A., 2017), it is chosen for the purpose of lowering computational costs.

3.5.3. Encoding

Since deep learning cannot work with categorical data they should be fed with encoded real numbers as a vector to improve the model classification accuracy (Hancock, J. T., & Khoshgoftaar, T. M., 2020). Before being fed directly into the model, the input dataset must be encoded. To accomplish this, two types of encoding are used sequentially, label encoding with integers and one-hot encoding. Label encoding is used for multi-label classification and involves the use of two types of labels. Severity has four classes that take 0 to 3 integers to encode severity

levels ranging from 0 to 3. Then, for eight classes ranging from 0 to 7, disease categories are encoded. Following the use of label encoding, categorical integers are encoded to a one-hot scheme using one-hot encoding. For each possible category, it generates a binary number. The following table shows a sample data frame with label encoding.

	severity	disease	file
0	healthy	healthyMaize	/home/cv-sig/Research Data/CombinedTrainF/0_0_...
1	level2	GLSMaize	/home/cv-sig/Research Data/CombinedTrainF/2_3_...
2	level3	commonRustMaize	/home/cv-sig/Research Data/CombinedTrainF/3_1_...
3	level2	GLSMaize	/home/cv-sig/Research Data/CombinedTrainF/2_3_...
4	healthy	healthyMaize	/home/cv-sig/Research Data/CombinedTrainF/0_0_...
...	...	...	...

Figure 12 Data frame sample before encoding

	severity	disease	file
0	0	4	/home/cv-sig/Research Data/CombinedTest/0_4_le...
1	1	1	/home/cv-sig/Research Data/CombinedTest/1_1_le...
2	0	0	/home/cv-sig/Research Data/CombinedTest/0_0_le...
3	3	3	/home/cv-sig/Research Data/CombinedTest/3_3_le...
4	2	1	/home/cv-sig/Research Data/CombinedTest/2_1_le...
...	...	...	...

Figure 13 Data frame sample after label encoding

3.5.4. Normalization

The goal of normalization is to translate features to a common scale so that the training stability, computational efficiency and model performance will be improved. It will convert the input image into normalized distribution by converting the natural feature value to a range between 0 and 1.Therefore it makes the images better visualized for the model. Since the images collected for this experiment are from different source with different format normalization is applied to

handle problems related to images with poor contrast therefore it is used to improve image intensity.

$$Zi = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad \text{eq. (2)}$$

Where input xi=(x1, x2…xn), min (x), max(x) are the min and max of x and Zi normalized output

3.5.5. Data splitting

Data splitting is the process of dividing a dataset into three distinct subsets based on the splitting ratio for each task. In the preprocessing stage, the dataset is prepared for the deep learning model. It is used to separate data for training, validation, and testing. Despite the fact that there are two common methods for splitting train and test data sets (70:30 and 80:20), an 80:20 splitting ratio is used in this study. Before performing augmentation techniques on the entire dataset, the data set that has been cleaned and transformed is divided into training, validating, and testing sets in an 80:10:10 ratio. As stated in data collection and data cleaning, the total dataset collected after cleaning was *9,518* after applying the train test split, with the training set containing 7,624 images and the validation and testing sets containing check the number of images 953and 941 images, respectively.

One of the most difficult aspects of working on this experiment is the imbalanced class distribution in the dataset. The original data set has a high imbalance for training the model, with a 1:15 ratio between the minimum and maximum dataset. This may cause the algorithm to ignore the minority class entirely, lowering the expected model performance. Random sampling, specifically oversampling, is one approach that can be used to overcome this problem. Oversampling is a technique for increasing the data size of a minority class by randomly duplicating the exact copy of the minority class or augmentation. Oversampling by duplicating the same leaf image will result in a bias in the model, which may worsen model performance because the minority class data size contains very few leaf images. As a result, the best option was to augment the original images using different varying techniques to generate slightly different images that can represent the real-world leaf variation.

3.5.6. Data Augmentation

Augmentation is a technique for increasing data size for deep learning in order to reduce overfitting and assist the model in generalizing rather than memorizing small data sizes. Augmenting the collected real dataset with a different version of the existing images of fitted transformation for the intended purpose is a low-cost, controllable, and accurate way to improve testing accuracy (Khalifa, N. E., Loey, M., & Mirjalili, S., 2021). It enables showing neural networks a wide range of dataset. Because the collected data set has a large imbalance, augmentation plays a significant role in overcoming this issue by creating a different variation to the original data to represent real-world variation. Some of the random transformations used in this experiment, which were chosen after experimenting with all augmentation techniques, are classified as positional and color augmentation. To expand the training data set, rotation, height and width shift, shear, brightness, zooming, horizontal and vertical flipping, and channel shift are applied.

1. **Position augmentation/Geometric**: Cropping or zooming to magnify the original image in a sort of scale, rotating images 45-degree angle (Sifre, L., & Mallat, S., 2013). A translation moves the leaf image from one position to another in the x or y direction and flips by reflecting images vertically and horizontally and shear which is changing the shape along the x and y axis (Vyas, A., Yu, S., & Paik, J, 2018). Shifting image pixels horizontally and vertically in some range moves the affected leaf area to the left-right and top-bottom.
2. **Color augmentation/Photometric:** increasing the brightness uniformly by increasing the pixel value by a certain amount to vary image visibility As a result, it appears darker and lighter in comparison to the original images. When color augmentation is used to feed the model color features regardless of saturation value and adding variability on the intensity value, channel shift occurs. As a result, it can represent some leaf images that may arise due to the acquisition protocol (Khalifa, N. E., Loey, M., & Mirjalili, S., 2021).

3.6.Modeling

Leaf disease and severity classification has been a well-known research topic for over a decade, utilizing image processing, machine learning, and recently deep learning. Typically, the disease and severity classification tasks are completed independently as a multi-classification task. In

this study, however, both tasks are combined into a single multi-labeling task by utilizing a single dataset for both classifications. Deep learning is selected to carry out Multi-label classification tasks. Unlike traditional Machine Learning, it works with data to extract features on its own. Whereas traditional machine learning anticipates extracted features through handicraft. The primary reason for choosing deep learning over other disciplines is its demonstrated superior performance in complex task including leaf disease classification. In this study, for example, extracting variable features that are raised due to differences in lighting, background, and data set variation is more than a complication with manual extraction (Wu, H., Liu, Q., & Liu, X., 2019). CNN architecture is chosen from Deep Learning specifically for modeling.

3.6.1. Why Convolutional Neural Network (CNN)?

A Convolutional Neural Network is a popular Deep learning architecture in the last decade while the world computational resource is not a problem anymore. It is designed for learning spatial hierarchal features automatically from the example dataset (Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K., 2018). It update its learnable parameters through backpropagation with a gradient decent optimizer which made it an effective feature extractor for images (Wu, H., Liu, Q., & Liu, X., 2019). Therefore it ease the researcher's task by extracting edges, textures and some variation on image position and color that makes it location invariant with an advantage of reduced number of parameters to learn features (Wu, H., Liu, Q., & Liu, X., 2019). The CNN model was chosen over other DL algorithms because of its weight sharing characteristics and convolutional operation, which make it efficient for leaf disease and severity feature extraction with images. It also fits in image classification tasks because it reduces the number of parameters, making it preferable for images, which have high dimensions and need large number of parameters.

3.7.Base Line Work

In order to compare the proposed model performance it is compared with VGG 16 (Visual Geometry Group) which is a Convolutional Neural Network architecture. Recently it become popular image recognition architecture which shows high performance and state-of the art result in different tasks. It contain total 16 number of layers with large number of parameters (138

million) and large size of weights which have high computational expense results in large training time. It have an advantage of high discrimination due to its blocks non linearity (Ajit, A., Acharya, K., Samanta, A. , 2020).It have been implemented in different image classification tasks including Disease and severity classification some and shows good performance (Syarief, M., & Setiawan, W., 2020) (Rangarajan, A. K., Purushothaman, R., & Ramesh, A. , 2018) (Ganatra, N., & Patel, A., 2020).

3.8.Development Tools

Deep learning relies heavily on development tools. This could include all of the hardware and software design tools used in this experiment, such as web applications, libraries, and platforms. These tools and software make development easier, faster, and more efficient. Tools for developing deep learning-related research are being released at a faster rate than ever before, but only a few of them have been chosen for this study and are listed below in the HW and SW categories.

3.8.1. Software Tools

Software tools are programs or libraries that are used to create, train, and test the model while also visualizing the entire experiment procedure.

- **Jupyter Notebook:** is a freely available mathematical framework (Randles, B. M., Pasquetto, I. V., Golshan, M. S., & Borgman, C. L., 2017) that is maintained by Project Jupiter members. It is integrated with an Anaconda environment and is used for code creation and running to implement the proposed work.
- **Colab**[1]: is a Python IDE for machine learning development that integrates with cloud storage released by Google. It is designed for Python and supports multiple libraries with no setup required. It is used for part-time testing with free GPU.
- **TensorFlow** [2]: an open-source Python library for numerical computation by Google. It is used for developing and training Deep Learning models .TensorFlow is flexible and

[1] https://colab.research.google.com/
[2] https://www.tensorflow.org/

powerful. It can also run from a highly distributed system to a low distributed system. It is used as a back end for the Keras interface in this study to develop the model, specifically for building the proposed functional model and implementing augmentation.

- **Keras [3]:** is a high-level library written in python programing language and can run on Theano, TensorFlow, Microsoft Cognitive (CNTK)Toolkit, and PlaidML back end. It is used to build Machine learning models. Keras is used as an interface with TensorFlow back ends for developing the proposed model. Functional API offered by Keras was used to create flexible models since it supports multiple outputs and sheared layers.
- **Google Drive[4]**is a free cloud service that allows to storage, sharing, backup, and use of data anytime and anywhere the user intended to use it by synchronizing multiple devices. It is developed by Google for the creation of documents, spreadsheets, and presentations. Since the collected dataset contain a large number of images google drive free storage, access, and backup services are needed.
- **Diagrams.net[5]:** is an online diagram tool that can be used to draw charts, diagrams, and boards for different purposes. It provides different shapes connecting lines and editing tools to make the diagram visually qualified by just dragging and dropping which is labor-saving and economical. In this experiment, diagrams.net have been used to draw block diagrams, flow charts, and tree graphs
- **CUDA[6]:** is an API and parallel computing platform developed by a company named Nvidia. This allows the use of GPU for computation to work with different programing language GPUs. It is mainly used as a platform for research and executing deep learning and parallel computing algorithms by enabling the use of GPU architecture (NIVIDIA, 2022).

3.8.2. Hardware tools

Hardware tools used for the research are camera, GPU, and Hard Disc are listed with their description and specification below

[3] https://keras.io/
[4] https://drive.google.com/
[5] https://www.diagrams.net/
[6] https://developer.nvidia.com/cuda-toolkit

No.	Device	Specification	Description
1	GPU	GeForce RTX 2070	Graphical Processing Unit for better computation
2	Hard Disc	500 GB	To store collected data set
3	Camera	13 MP	To capture leaf images

3.9.Evaluation Methods

Evaluating methods is one assured way to show the model performance in any traditional Machine learning Models and deep learning. This includes some metrics to measure the performance of the proposed model. In this research three metrics are used to evaluate performance, precision (P), recall (R), and F1 Score (Grandini, M., Bagli, E., & Visani, G. , 2020).

- **Precision (P):** Positive Predictive Value (PPV)): is a measure of classifiers' exactness. A low precision indicate a large number of False Positives.

$$P = TP / TP + FP \quad (1)$$

- **Recall(R):** is the number of positively classified targets divided by the number of positive class values in the test data called sensitivity or the True Positive, used to measure classifiers' completeness.

$$R = TP / TP + FN \quad (2)$$

- **F1 Score:** combines precision and recall relative to a specific positive class. It is a weighted average of the precision and recall, its value ranges between 0 and 1

$$F1\ Score = 2 *(precision*recall) / (Precision+ recall) \quad (3)$$

- **A confusion matrix:** - is also an evaluation metric that is used to describe the performance of a classification task. Precision, Recall, and Accuracy can all be calculated with the help of a confusion matrix

Table 4 Confusion matrix terminology

	Actual(+ve)	**Actual(-ve)**
Prediction(+ve)	True positive(TP)	False Positive (FP)
Prediction(-ve)	False Negative (FN)	True Negative(TN)

3.10. Overfitting problem

Overfitting occurs in a model when the proposed model is fed with a limited training dataset therefore the model is used to memorize the dependent dataset only and not generalize on unseen datasets. The presence of noise in data, the limited size of the training set, and the complexity of the model are the main reason behind Overfitting. The solution to prevent this overfitting problem is applying different regularization techniques to tune the model depending on the specific problem context.

In order to prevent the overfitting problem, four main strategies are targeted. The first one is early stopping the training before the performance drops optimization. The second choice is reducing the complexity of the network to prevent memorization of noises in the training set. The third and fourth methods are data expansion and applying regularization by selecting main features and removing irrelevant features for that specific problem with L1 and L2 (Ying, 2019)

3.10.1. Targeted techniques to avoid overfitting

There are different methods to avoid and reduce the overfitting problem some of the major methods are discussed bellow

Regularization techniques are methods that add information to avoid the occurrence of overfitting by reducing validation loss or generalization error keeping the training performance.

- Playing with Loss functions such as applying L1, L2, and entropy regularization to work with the parameters of the model.
- Focusing on the dataset sampling by applying data augmentation if the overfitting occurs due to the lesser dataset. Training with a k-fold cross validation method

- Modifying the model by dropping out some neurons activation to reduce memorizing the pattern and applying noise on a dataset to make the model robust for different variations that will arise in the testing dataset(training approach method)

Expand training dataset: performance of classification can be affected by both the quality and quantity of the data sample. Particularly for a complicated model, the number of parameters to be tuned is more, and to tune those parameters a proportional data sample is required. Collecting more datasets or augmenting the existing dataset by introducing some noise and variation to it shall expand the training set.

Early stopping: is the other strategy to prevent overfitting by stopping the training before the validation loss seas to improve instead even go worth, while training loss still decreases. This can be done by computing accuracy after each epoch and quitting training when validation accuracy stops improving. In training finding the right weights and bias is the main aim this can be done by determining the partial derivative of the cost function.

$$\frac{\partial c}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma z - y) \qquad \frac{\partial c}{\partial \mathrm{b}} = \frac{1}{n}\sum_x (\sigma z - y) \tag{4}$$

(Ying, 2019)

Where w, b, c, x_j, y, z is weight, bias cost, jth input, and output, respectively. The speed of learning and the performance of the mode depends on that partial derivative.

Dropout: randomly dropping the activation of some neurons from the network will avoid too much adaptation and memorization. It is the preferred solution for a complex neural network model that needs a large dataset to be trained to reduce both computations and is prone to overfitting.

K-fold cross-validation: Using a single parameter K, the K-fold cross-validation method is used to evaluate models in a small number of data samples. K value is a number that is assigned to sample groups and is used to divide the entire data sample. This method is used to evaluate and estimate the performance of a model on unseen data. It has a significant impact on reducing overfitting and providing unbiased and less optimistic promises. First, the data set was divided into train and test datasets. The training set is divided into k folds and trained on each fold

using the previously optimized weights. Finally, the model is tested using an independent testing dataset.

CHAPTER FOUR

4. PROPOSED SOLUTION

4.1. Chapter Overview

As discussed in the previous chapter the tools, techniques, and methods are discussed in detail. In this chapter, the proposed model architecture and data preparation procedure are expressed in a more elaborated and detailed fashion. To illustrate the whole content this chapter is divided into three sections. In the first section, the overall architecture of the proposed model to implement Disease and Severity classification or identification is described. The second part discusses the detailed architecture and building blocks with base hyper parameters are described. Finally the comparison model which is adapted VGG model architecture is described.

4.2. Work Architecture

To build this model, a specific work architecture was followed. First, the training data is collected as illustrated in chapter three before implementing the data on the classification model. Data passed through a different path to make it fit the model and the model is evaluated and compared with the baseline work. Then, the classification model which is the proposed model will be discussed in the following sub-sections in detail. Finally, the classification model is trained by augmented training and validation dataset and predicted by testing data set as shown in the following proposed solution architecture diagram.

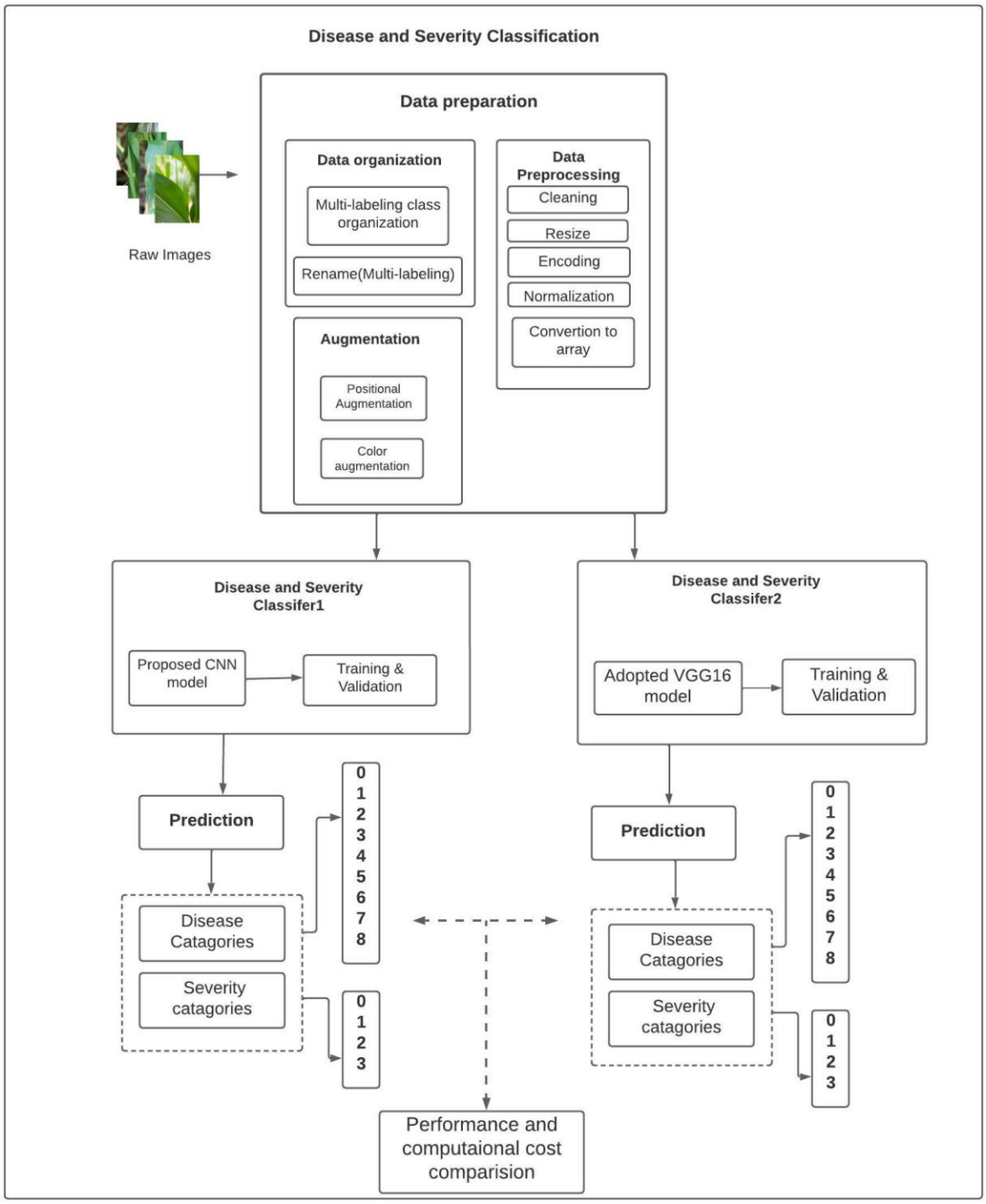


Figure 14 Proposed work overall architecture

4.3. Proposed Model

In order to perform both disease and severity classification for two crops, a multi-label classification with a Deep learning approach is provided. Several convolutional architectures have been implemented in the past for disease and severity classification separately. However, this research provides a model that can perform the two tasks simultaneously with a single dataset. Therefore in the end the model is expected to generate two types of classification. In this research, a modified version of the CNN model is provided to improve the performance and the function. It includes a combination of VGG and ShuffleNet-V2 structures. The overall architecture of the proposed model is shown below.

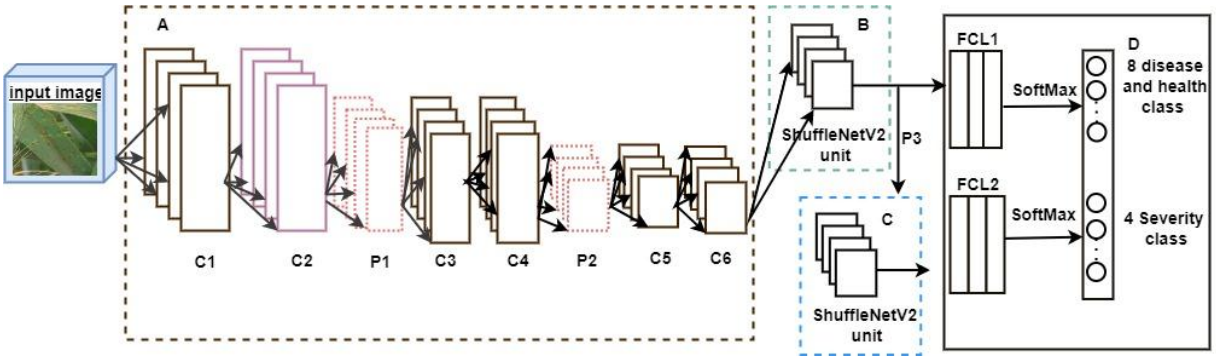


Figure 15 Proposed architecture for disease and severity classification

The proposed model is the modified version of basic VGG architecture by reducing the number of layers to fit the reduced input size of images and applying Chunnel shuffle units at the output layer before the three fully connected layers. Chunnel shuffle units are used as auxiliary structures to split disease extracted features into two which enable the model to do two functions. The proposed architecture has four blocks indicated by A, B, C, and D as shown in the diagram (Figure 3). Where Block A represents the feature sharing layers module that contains 9 layers, 6 convolutional layers (C1-C6), and 3 pooling layers (P1, P2, and P3).The last layer which is P3 transfer features to Block B containing shuffle block by applying convolution. Block B contain features for disease classification. After extracting disease features disease classifier classifies the type of disease while Block C severity feature extractor reuse extracted disease features for further severity level classification purpose. Finally, Block D contains a pair of three fully connected layers followed by SoftMax for each output that will

be delivered from shuffle blocks. Therefore two types of classification which are disease and severity were expected.

Batch Normalization and ReLu activation functions are applied after each convolution and an Adam gradient descent algorithm has been used to optimize the CNN learning rates. Adam evaluates adaptive learning rate from moments of the gradient for various parameters combining RMSprop and AdaGrad optimizer's advantage. ReLu activation, a non-linear function that will give an output that is greater than zero used in a neural network to ease the training and improve performance by overcoming the vanishing gradient problem**.** The pooling layer after ReLu activation is used to reduce the spatial size of the feature map to reduce computational cost and complexity**.** Each convolution stride is a parameter of the convolution neural network filter. Batch Normalization is implemented to speed up training and use a higher learning rate to make learning easier by stabilizing the learning process and reducing the number of epochs after each convolution.

4.4.ShuffleNet

ShuffleNet is one type of Convolutional Neural Network specifically made for low computational devices such as mobile due to its weight and lightness with very low computing power (Zhang, X., Zhou, X., Lin, M., & Sun, J., 2018).It measures computational complexity using direct metrics such as speed or memory access cost rather than indirect metrics such as flops. It have cutting-edge in terms of the speed-accuracy tradeoff. (Ma, N., Zhang, X., Zheng, H. T., & Sun, J. , 2018) Figure 16 depicts its structure clearly. Due to its effective in reducing computation cost while achieving better accuracy by applying channel shuffle and pointwise convolution operation it is implemented in the proposed model.

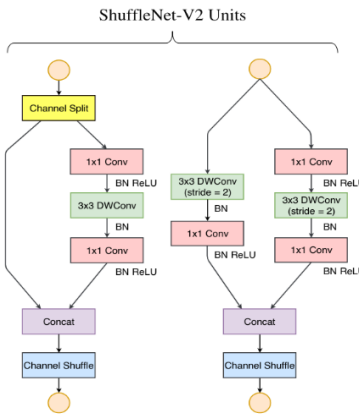


Figure 16 ShuffleNet-V2 structure (Ma, N., Zhang, X., Zheng, H. T., & Sun, J. , 2018)

4.4.1. Building Blocks of Proposed Model Architecture

The proposed model consists of multiple building blocks such as convolution layers, pooling layers, ShuffleNet block, and fully connected layers. All the details of each building block and operations applied are explained below.

4.4.2. ShuffleNet unit

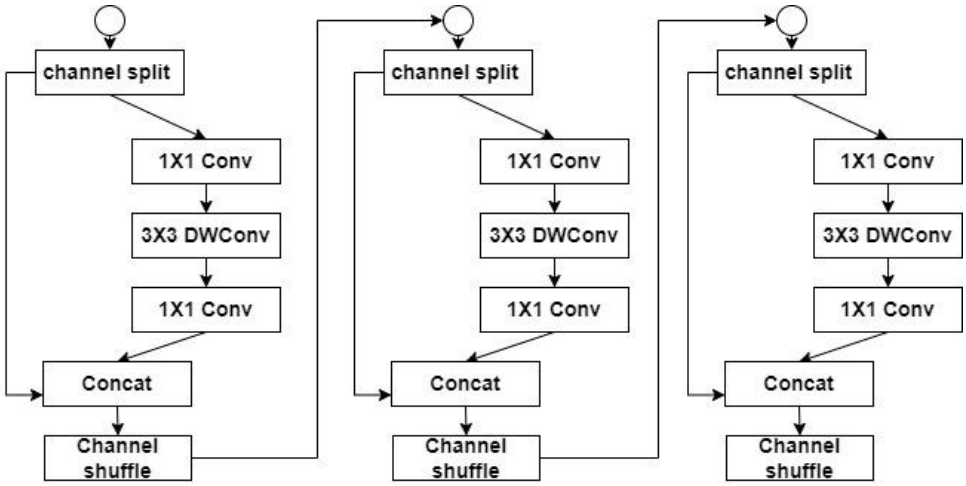


Figure 17 ShuffleNet unit detail structure

4.4.2.1. ShuffleNet V2 block

ShuffleNet V2 block is a shuffle block that uses channel shuffle operation with depth-wise convolution to design an efficient image architectural model. First, the input channels split into

two branches. Branch one channel has the original shape of the channels for keeping the same shape[7]. Branch two passes through different operations, convolved with a 1x1 kernel which is a 1x1 convolution[8], and then a depth-wise convolution with a 3x3 kernel. The depth-wise convolution output again will convolve with a 1x1 kernel. Then the two different branches will be concatenated[9] to form the same shape. Finally, a shuffle operation is implemented.

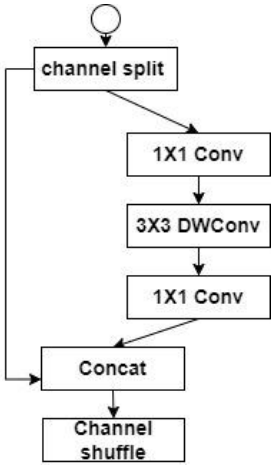


Figure 18 Single ShuffleNet Block *(Ma, N., Zhang, X., Zheng, H. T., & Sun, J. , 2018)*

4.4.2.2. Channel Shuffle

Channel shuffle is an operation performed on the input tensor's channels. The goal of using this operation is to allow the channels to communicate with one another for more group convolution layers. Because it allows group channel communication, it allows communication between all channels, which is ideal for this study's unbalanced dataset.In the proposed model case 8 groups is implemented for Channel shuffling.

---

[7] 1st guide line:" Equal channel width minimizes the memory access cost" applying balanced convolution with the same number of input channels and output channels (1:1) reduces memory cost.

[8] 2nd guide line: "Excessive group convolution increases memory access cost" The group number selection should be taken wisely since it may reduce running speed which lead to high Memory access cost.

[9] 3rd guide line: "Network fragmentation reduces degree of parallelism" The benefit of Fragmentation in accuracy costs efficiency for parallel computing devices.

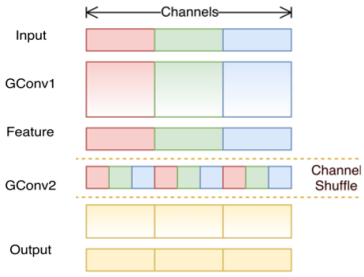


Figure 19 Channel shuffle operation *(Zhang, X., Zhou, X., Lin, M., & Sun, J., 2018)*

- Two stacked convolutional layers where GConv is Group convolution and channels are only related to the corresponding channel in the group.
- Channels at the output are related after GConv2.
- Relating output channels with channel shuffling

Algorithm

Take input tensor as x and group as g

Get the dimension of the input tensor x (width, height, and channel)

Calculate channel numbers in each group by dividing the number of channels by groups

Reshape tensor to (width, height, group-channel, and group)

Permute/Transpose the last two dimensions group-channel, group->group, group-channel

Then reshape it to the previous shape

Finally, return tensor

Depth-wise-Separable convolutions: is a combination of pointwise (spatial) convolution keeping the channels separate and depth-wise convolution (Chollet, 2017). Therefore its computation is split into two rather than in a single step as a standard convolution. Where point-wise is used to create a linear combination of depth-wise convolution. Performing a single convolutional filter for each input channel by depth-wise and then point-wise convolution creates a linear combination of its output. The main advantage of using this type of convolution is to reduce the number of parameters in convolution.

1x1 Convolution: is a point-wise type of convolution involving a 1x1 filter convolving over the whole image pixel by pixel used for reducing dimensions depth-wise that enabling to increase in the depth, and width of the network by removing significant computational bottlenecks. It also has the advantage of efficient low-dimensional embedding with sufficient information and the ability to apply nonlinearity after convolution. That allows more complex functions to be learned by the network (Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A., 2015).

Depth-wise Convolution is a type of convolution where a single filter is applied for each input channel. It keeps each channel separately by splitting the input and filter channels then convolving each input channel with the respective filters independently and finally stacking the output together. It changes both the spatial dimension and depth of the image. Due to its advantage in reducing the number of parameters and neural network computation it is preferable to keep the model generalizing instead of being prone to overfitting.

4.4.3. Basic Building Blocks

Input layer: The input layer of the proposed model is fed with RGB images having a size of 128x128 with multi-labeled input of 8 diseases and 4 severity classes. It contains 3 dimensions of shape that include width, height, and the number of channels which equals 3. The input layer passes input images to the next convolutional layer with zero number of parameters and no computation.

Convolutional Layer: In a CNN model convolutional layer is a key part of the architecture. It performs multi-level convolution to extract spatial hierarchies of feature patterns and characters from input images. A convolutional layer is composed of convolutional and linear mathematical operations. Extraction is done by using a specific kernel, with a small dimension passing through each of the image positions to extract special features which can be found in any of the image positions. The kernel contains a small grid of parameters that can be optimized. To build this CNN model seven 2-D convolutional layers, eighteen pairs of 1x1 convolutional layers, and 3x3 depth-wise convolutional layers are used. The following paragraphs show the detail of each layer.

In the feature sharing block, seven 2-d convolutional layers are included with 3 pooling layers after each pair of convolutions to down-sample the feature maps. In ShuffleNet unit pair of 1x1 convolutional layers and single 3x3 depth wise convolutional layers are included in each ShuffleNet block. Since we have three ShuffleNet units for each branch containing three ShuffleNet blocks total of 3 pairs of 1x1 convolutional layers with three single depth-wise convolutional layers in one block. Therefor there are nine pairs of 1x1 convolutional layers and nine 3x3 depth-wise convolutional layers in each branch (severity and disease feature extractor).To sum up the hole convolutional layers in both severity and disease feature extractors there are 18 pairs of 1x1 convolutional layers and 18 3x3 depth wise convolutional layers used for feature extraction after the feature sharing convolutional layers module.

Pooling Layer: a pooling layer is a layer used mainly for compressing images by taking the average and maximum value of the image filter and delivering its matrix with a compressed output. It aggregates the neighborhood information by down sampling the region information. Therefore it reduce the size of feature maps to reduce model complexity and prevent overfitting also quicker computation. Max-polling and average polling layers are used for this model. Three max-pooling layers are applied for feature-sharing convolutional layers for the reason of their ability to select the most prominent features and two average pooling is used between shuffleNet unit and dense layers for each classification branch.

Fully Connected Layer (FCL): There are three pairs of fully connected layers including the output layer in the proposed model which are connected for both disease and severity classifier branches. The first FCL consists of two pair of fully connected layers having 1024 neurons and the last fully connected layer contain classification neurons. The first FC layer accepts the output of shuffleNet unit down-sampled by average pooling and converted to a flattened one-dimensional (1D) array of numbers or vector values.

Output Layer: The third pair of a fully connected layer with Softmax activation from a different branch is the output layer with 4 and 8 neurons or classes for severity and disease multi-label classification tasks. Softmax activation function scales logits to the probability where the higher probability can decide which class should fall. The input image therefore one of the classes will be assigned as a predicted class. From this layer, the model is expected to give multi-labeled classification output.

Activation Function: activation function is a nonlinear function that takes the output of a linear function such as convolution. It is used to introduce non-linearity before firing the previous layer output to the next neuron. So that it can help to reduce linearity and help the model to learn more complex patterns by promoting a Deep Neural Network. There are various activation function in a neural network the most common one is Sigmoid, Tanh, SoftMax, ReLU, and Leaky ReLU activation functions that can be used as neuron activators. Among the common functions SOftmax and Relu activation functions are used to build the model. Where SoftMax is used in the two output layers while Relu is used in the rest of the model building blocks. Where ReLU computes the maximum value between zero and input (Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K., 2018).

$f(x) = max(0, x) \quad , f(x) = \begin{cases} 0, if\ x < 0 \\ x\ if\ x \geq 0 \end{cases}$ ReLU activation function where x is an input value *(5)*

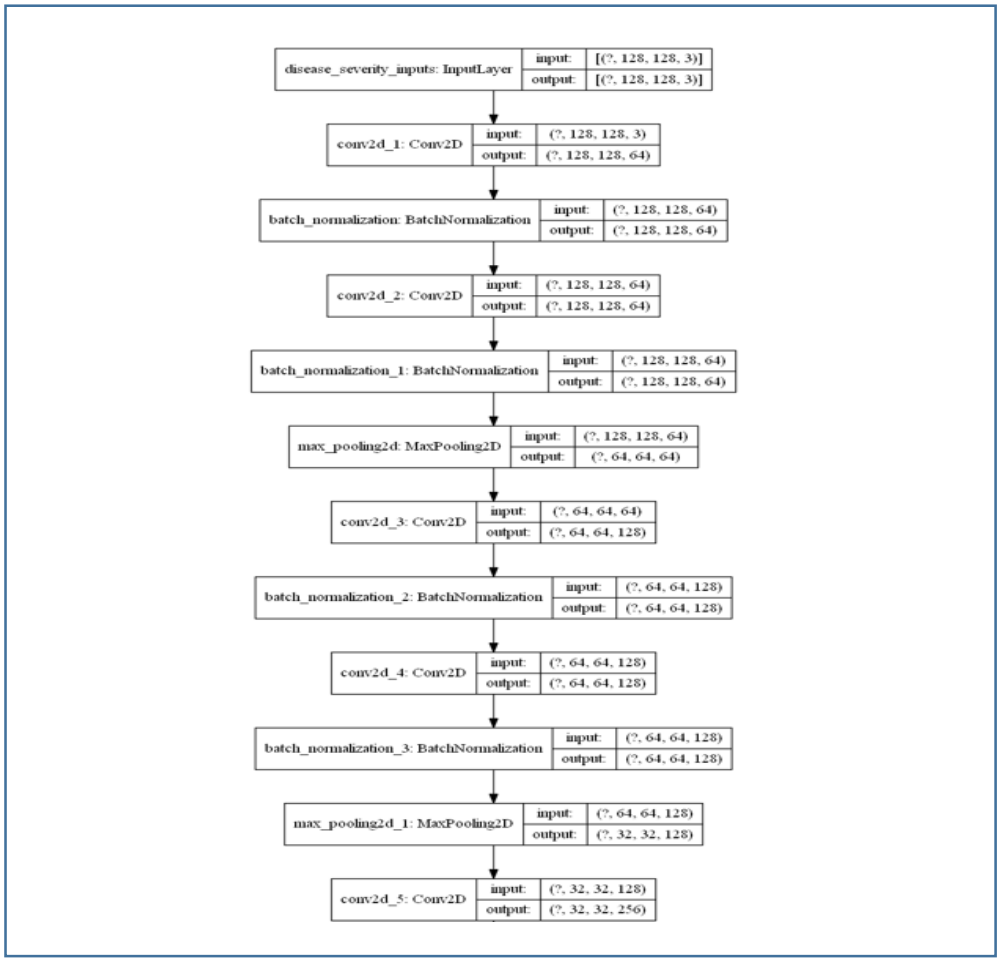
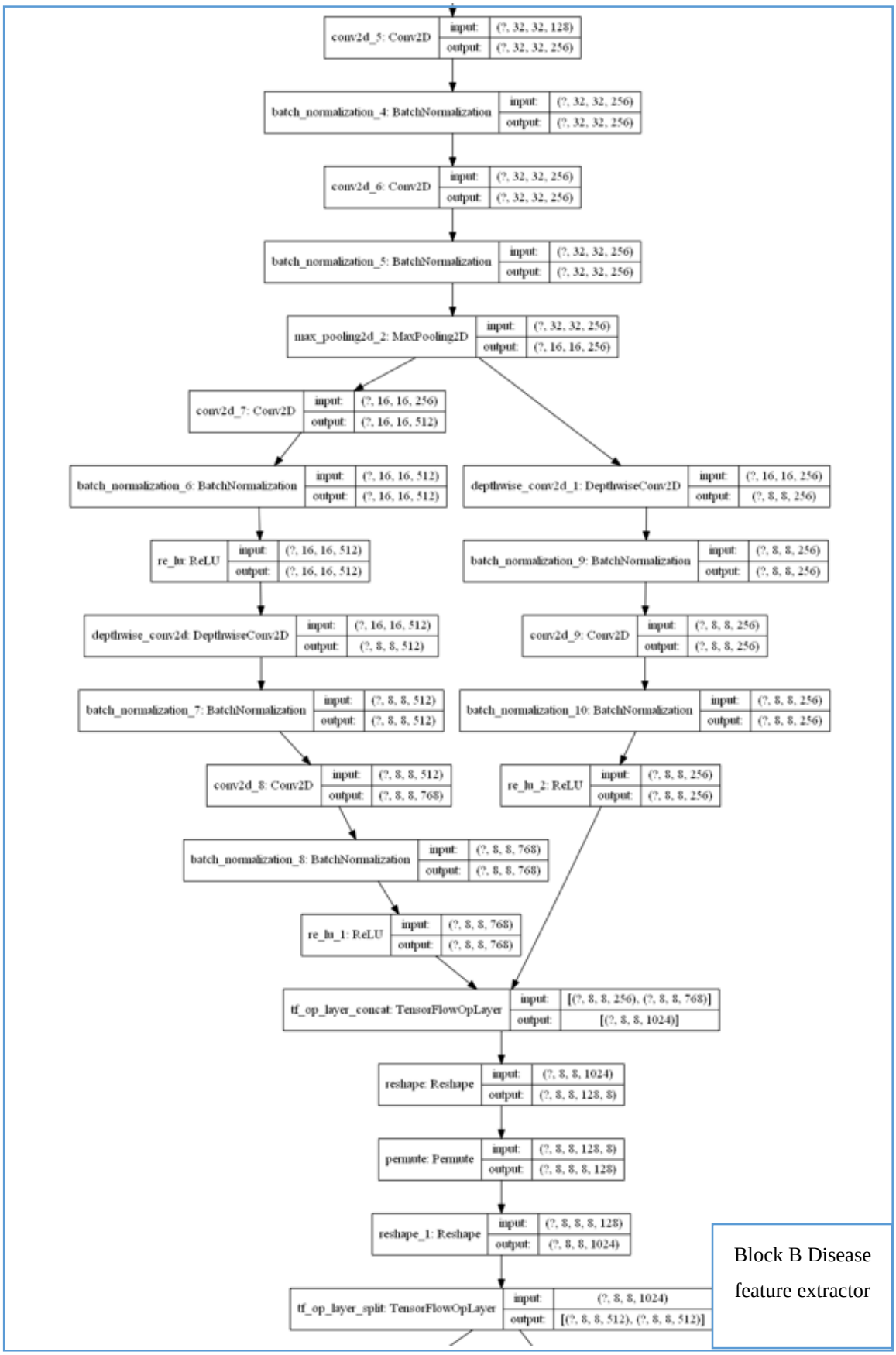


Figure 20 Block A feature sharing schematic diagram



Block B Disease feature extractor

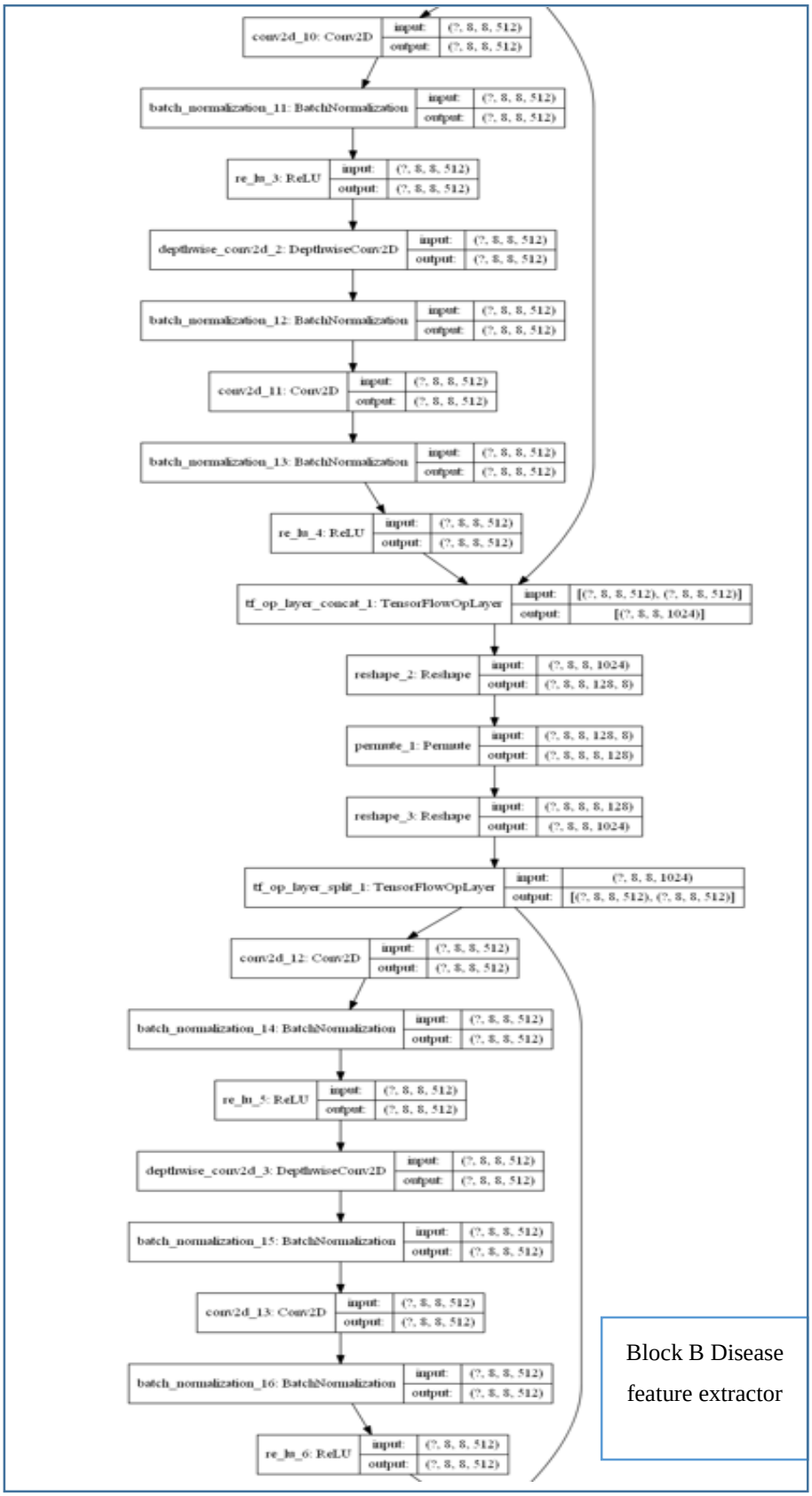


Figure 21Block B disease feature extractor schematic diagram

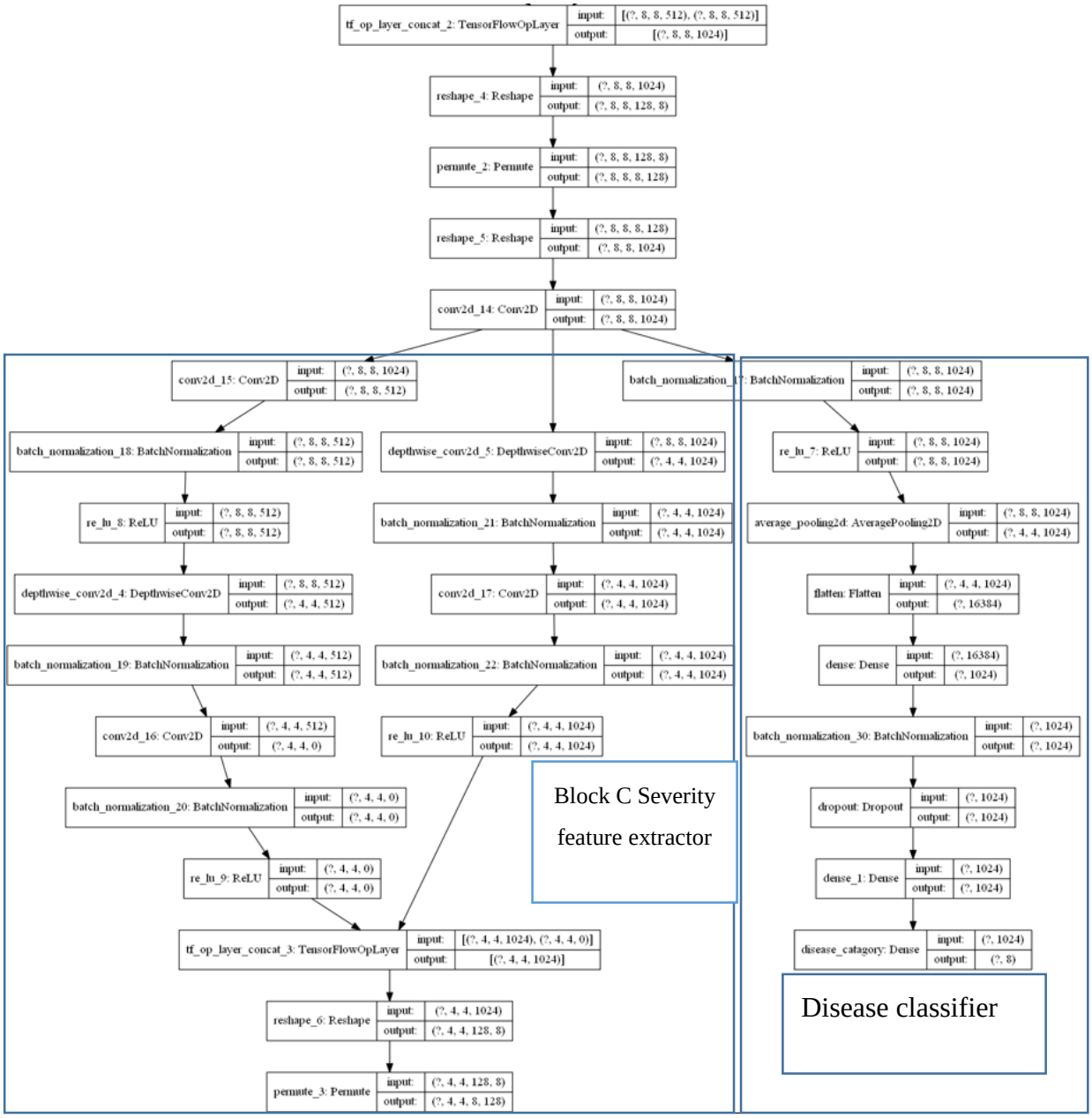
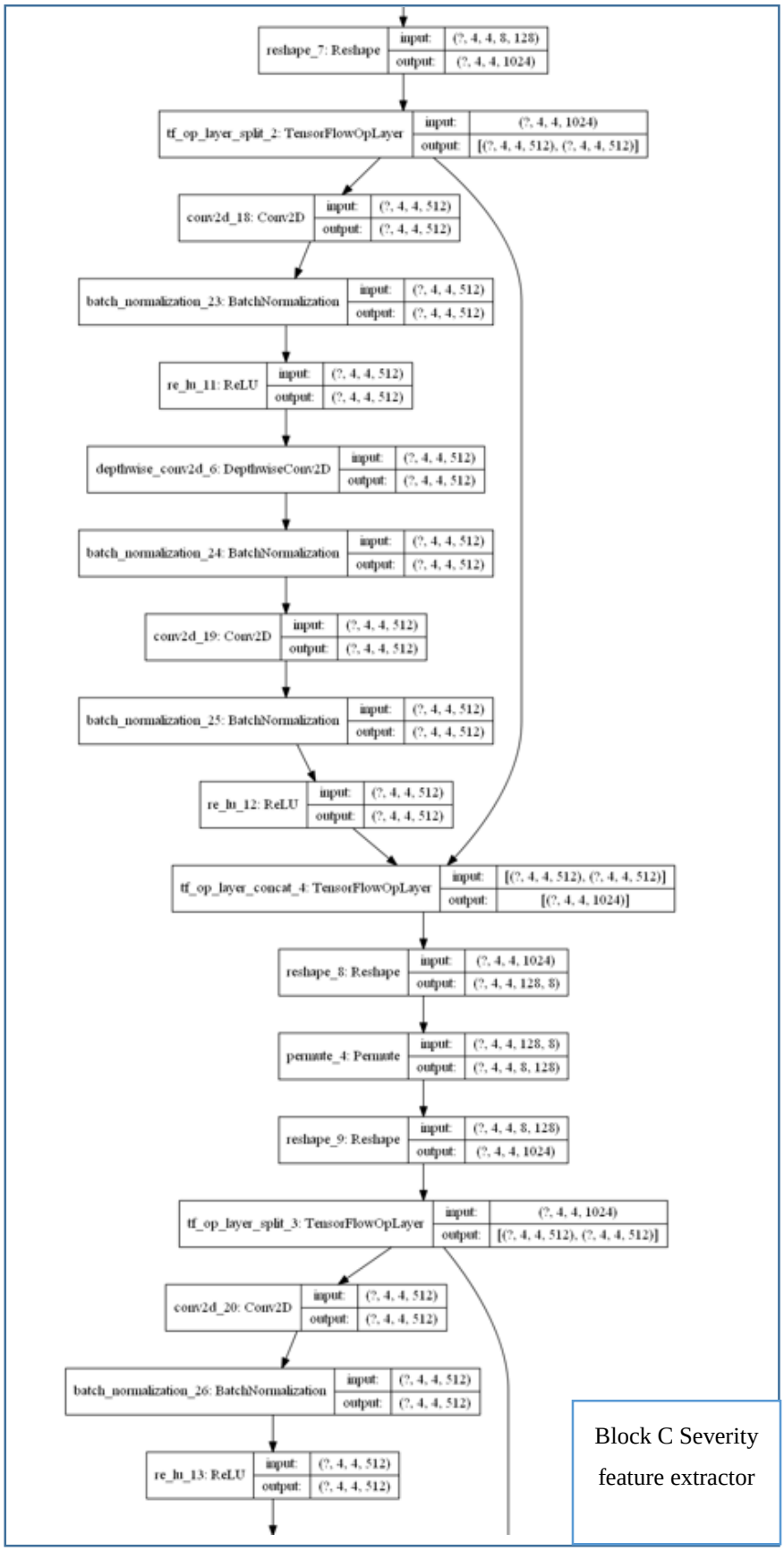


Figure 22 Block c and Disease classifier schematic diagram


reshape_7: Reshape
input: (?, 4, 4, 8, 128)
output: (?, 4, 4, 1024)
tf_op_layer_split_2: TensorFlowOpLayer
input: (?, 4, 4, 1024)
output: [(?, 4, 4, 512), (?, 4, 4, 512)]
conv2d_18: Conv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_23: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
re_lu_11: ReLU
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
depthwise_conv2d_6: DepthwiseConv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_24: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
conv2d_19: Conv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_25: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
re_lu_12: ReLU
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
tf_op_layer_concat_4: TensorFlowOpLayer
input: [(?, 4, 4, 512), (?, 4, 4, 512)]
output: [(?, 4, 4, 1024)]
reshape_8: Reshape
input: (?, 4, 4, 1024)
output: (?, 4, 4, 128, 8)
permute_4: Permute
input: (?, 4, 4, 128, 8)
output: (?, 4, 4, 8, 128)
reshape_9: Reshape
input: (?, 4, 4, 8, 128)
output: (?, 4, 4, 1024)
tf_op_layer_split_3: TensorFlowOpLayer
input: (?, 4, 4, 1024)
output: [(?, 4, 4, 512), (?, 4, 4, 512)]
conv2d_20: Conv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_26: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
re_lu_13: ReLU
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
Block C Severity feature extractor

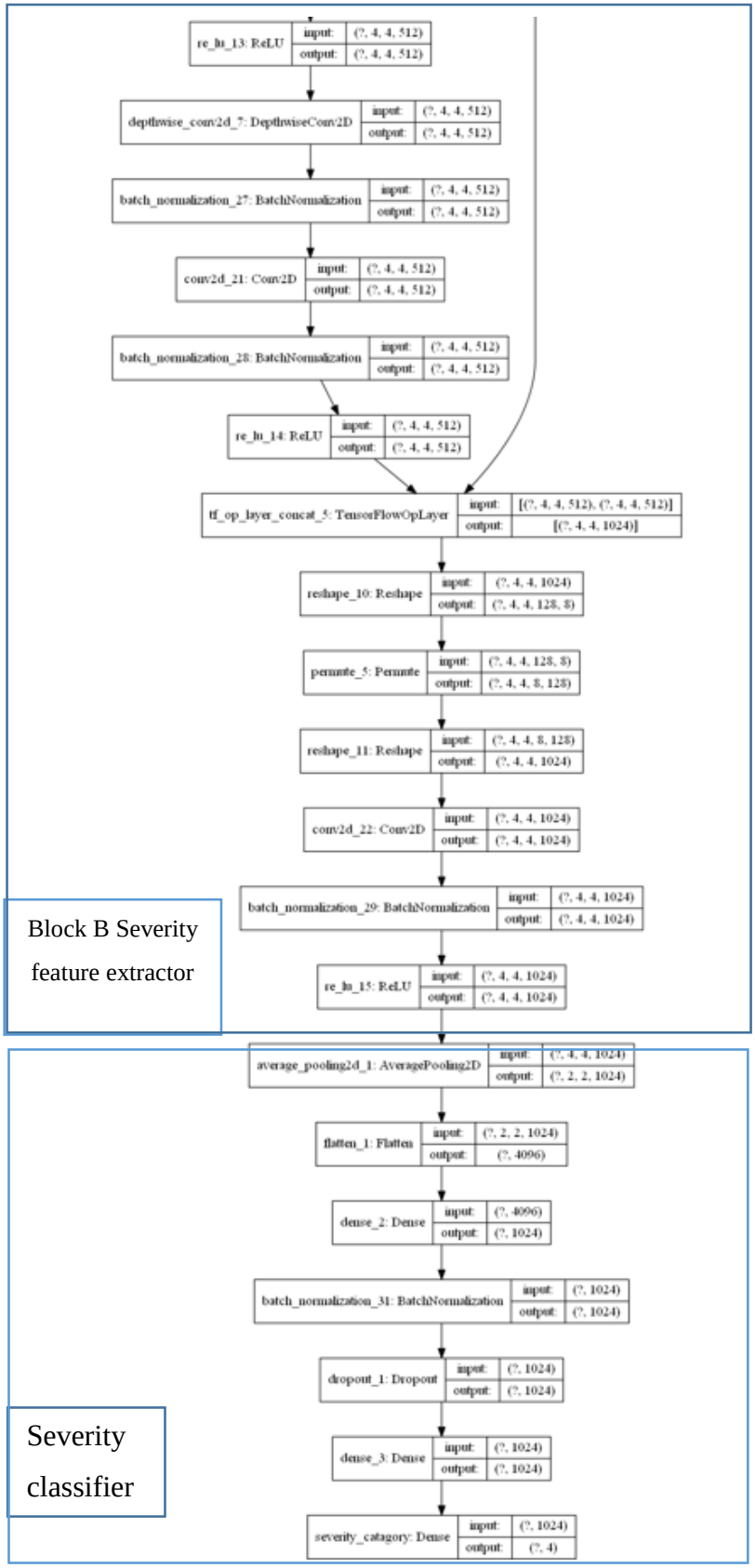


Figure 23 Block C severity feature extractor and severity classifier schematic diagram

4.5. Adapted VGG model

VGG model is the base CNN model for the proposed model improvement therefore an effort has been made to convert it for multiple outputs for comparison of the proposed model. Therefore it becomes possible to compare whether the proposed work shows an improvement both in the computational expense, and performance with multiple functionalities. The following image, *Figure 24* and *Figure 25* shows the adopted VGG model architecture which is converted to a functional model after modifying it for multiple outputs by including an additional classifier module consisting of three fully connected layers at the end of the ShuffleNet feature extractor.

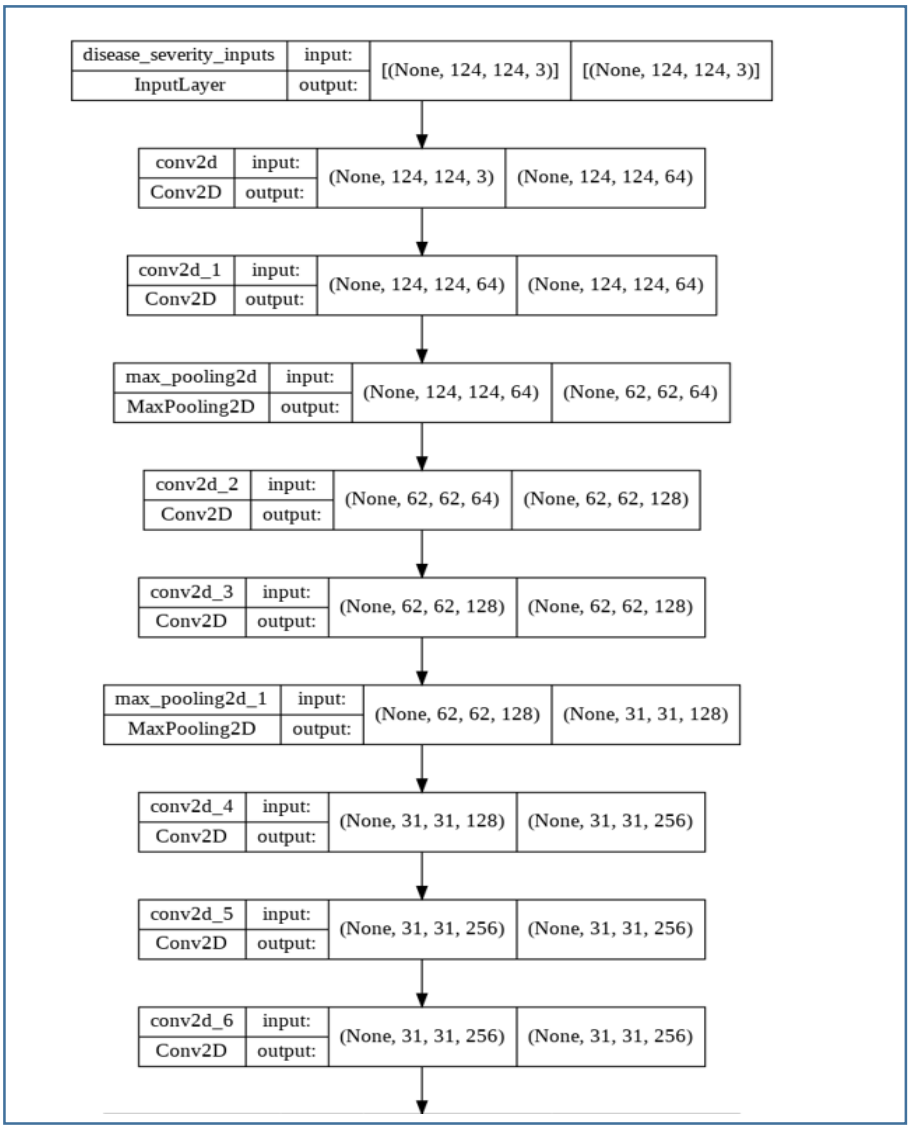


Figure 24 Adopted VGG Model schematic diagram first section

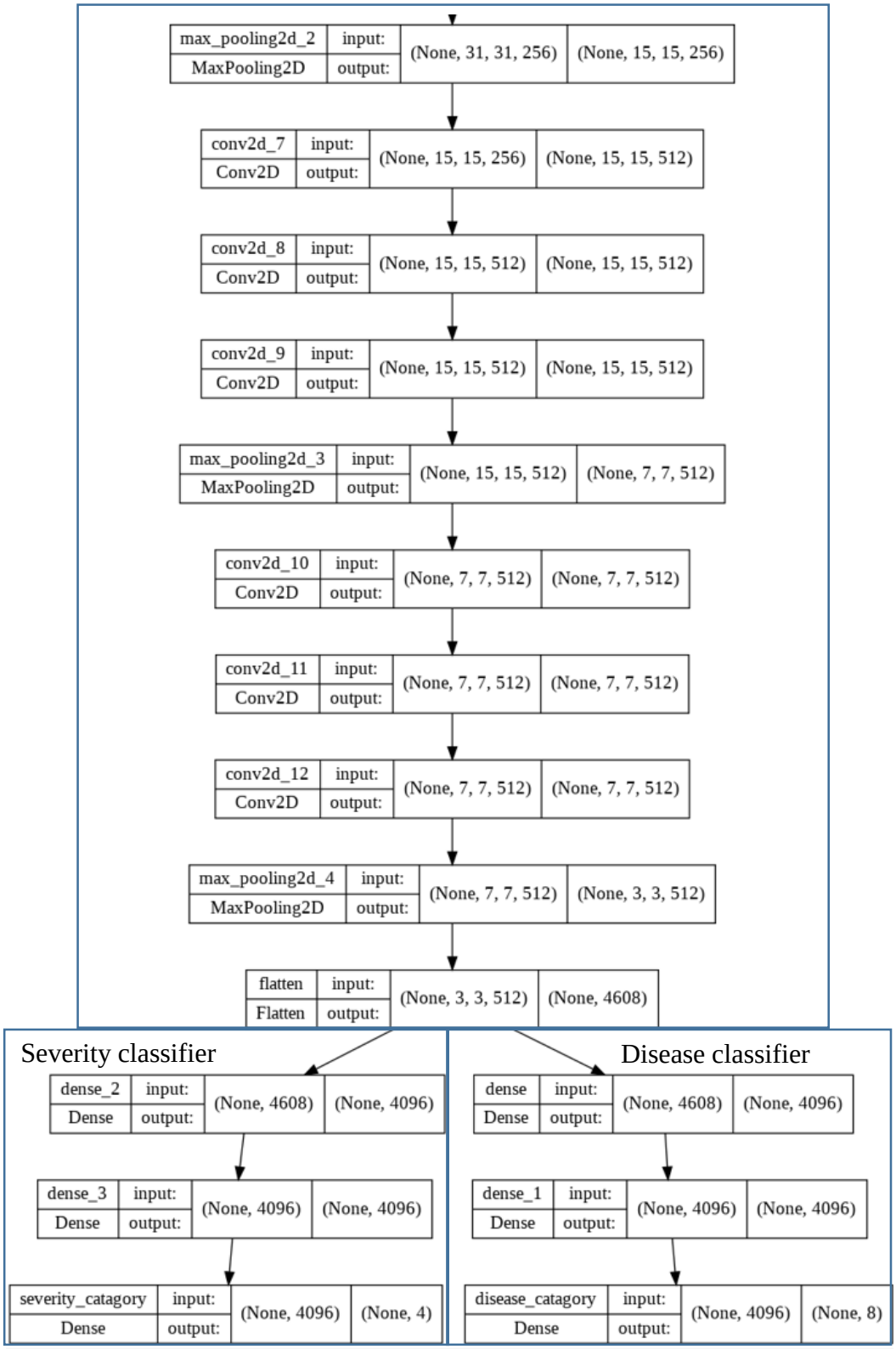


Figure 25 Adopted VGG Model schematic diagram second section

CHAPTER FIVE

5. IMPLEMENTATION

5.1. Chapter Overview

In this chapter, the implementation of the proposed solution and deployment environment specification is reported. The chapter content includes pre-processing, augmentation, proposed model building, training, testing, and evaluation code implementation. Furthermore, the detail of each implementation is stated in a descriptive form. Also, the tools that have been used and their specification in combination with why it is preferred are described in detail.

5.2. Deployment Environment

A combination of a local Jupyter notebook platform and a google Colab platform connected with local resources were used for the implementation of this study. A desktop computer with an Intel ® core I5 -6500 CPU @ 3.20 GHz processor, GeForce RTX 2070 GPU with 12 GB of RAM GPU, 4 Logical core Processor(s), 16 Gigabyte of physical memory, and Ubuntu 22.04 platform, supporting 64bit operating system have been used to deploy the following tools and packages for the implementation of the model.

Python is a high-level interpreted programming language, used to prepare, preprocess datasets and implement the model. The main reason behind choosing python as an implementation of this model is due to its richness of resources for AI specifically Machine learning, deep learning, and Data science tools and packages which reduces the implementation task. Recently most of those modules and libraries become python based. Also, it enables low-level library usage and clean high-level APIs (Raschka, S., Patterson, J., & Nolet, C., 2020). Due to this, it become a preferred tool for scientific computing and machine with its better performance and productivity.

5.2.1. Libraries and Toolkit

The model implementation modules include helpful libraries, header files, and toolkits as shown in the following table with their respective version.

Table 5 Tools and libraries

Tools	Specification	Description
Python	3.10.4	Programing language for creating deep learning algorithm
Anaconda	4.13.0	Python distribution for computation
Keras	2.9.0	API for deep learning model building
Keras preprocessing	1.1.2	Keras data preprocessing module
Matplotlib	3.5.1	Graphical plotting for data visualization
TensorFlow-GPU	2.9.1	Keras back end for GPU enabled deep learning functionality
Diagram.net	-	Diagram creation
Colab	-	Python code writing and execution
Jupyter Notebook	6.4.11	Web app to create and share documents
Cuda	11.7.64	Parallel computing platform for GPU
Sklearn. metrics	1.1.1	Measures classification performance
Seaborn	0.11.2	Data visualization
Pandas	1.43	Analyze data
Numpy	1.23.1	Array manipulation

5.3.Disease and Severity Classification Implementation

Recalling the model architecture and description presented in section 4.2 and 4.3 this section deals with its implementation. In-depth discussion has been had regarding the suggested model design and its working principle. The implementation of the mentioned model architecture is then described in this section along with associated Python code. Additionally, preparation and preprocessing techniques implementation are covered.

5.3.1. Data preparation and Preprocessing

While preparing the dataset for training and testing the first step taken was loading image files and then extracting file information from the dataset through iteration of each image.

5.3.1.1. Image loading and Renaming

To convert the annotated image into a proper format for multi-labeling classification each class annotation was renamed with a new label containing both disease and severity category as shown in the following sample code for an image representing 3rd severity level of yellow rust wheat disease. To implement renaming the training data set have been loaded from google drive and the OS library is used to rename the file to the corresponding class labels. As the sample code shows the first two integers in 3_7_level3yellowRustWheat label separated by underscore represent severity level and disease category respectively.

```
import os
os.getcwd()
collection = "/content/drive/MyDrive/TrainData/3_7_level3yellowRustWheat/"
for i, filename in enumerate(os.listdir(collection)):
    os.rename("/content/drive/MyDrive/TrainData/3_7_level3yellowRustWheat/" + filename,
     "/content/drive/MyDrive/TrainData/3_7_level3yellowRustWheat/"+ "3_7_level3yellowRustWheat" + str(i) + ".jpg"
```

Sample code 1 Renaming images

5.3.1.2. Multi-label File Reading

Dictionary of disease and severity classes category names were used to encode related multi-label for each class-specific file. Therefore categorized datasets were encoded into label encoding integer values which have a form of vector encoded integers for each class. For this purpose, a dictionary of disease and severity categorical names are created as shown in the following sample code.

```
dataset_dict = {

      'disease_id': {
        0: 'healthyMaize',
        1: 'commonRustMaize',
        2: 'blightMaize',
        3: 'GLSMaize',
        4: 'healthyWheat',
        5: 'leafRustWheat',
        6: 'steamRustWheat',
        7: 'yellowRustWheat'

    },   'severity_id': {

        0: 'healthy',
        1: 'level1',
        2: 'level2',
        3: 'level3'
         }


}
dataset_dict['severity_alias'] = dict((g, i) for i, g in dataset_dict['severity_id'].items())
print(dataset_dict['severity_alias'] )
dataset_dict['disease_alias'] = dict((r, j) for j, r in dataset_dict['disease_id'].items())
print(dataset_dict['disease_alias'] )
```

Figure 26 Dictionary of disease and severity labels

5.3.1.3. Multi-label Data Structuring Implementation

Python's Pandas library was used to create a data frame in the form of a tabular data structure containing rows and three columns of labels. Then the extracted metadata was returned as a data frame containing three columns with an image file, and labels of disease and severity labels. Therefore, multi-labeled information of all files is parsed from each file with parse_info_from_file method to extract multi-label information from a single file for each respective label. The meta-data retrieval implementation is shown in the following code sample.

```
def parse_dataset(dataset_path, ext='jpg'):
    def parse_info_from_file(path):
        try:
            filename = os.path.split(path)[1]
            filename = os.path.splitext(filename)[0]
            severity, disease, _ = filename.split('_')
            return dataset_dict['severity_id'][int(severity)], dataset_dict['disease_id'][int(disease)]
            print("severity  id:",severity_id)
        except Exception as ex:
            return None, None
    files = glob.glob(os.path.join(dataset_path, "*.%s" % ext))
    records = []
    for file in files:
        info = parse_info_from_file(file)
        records.append(info)
    df = pd.DataFrame(records)
    df['file'] = files
    df.columns =[ 'severity', 'disease','file']
    df = df.dropna()
    return df
```

Sample code 2 Parsing data set information

5.3.1.4. Image Preprocessing Implementation

The next procedure was image preprocessing implementation to get the intended effect on the model. Different image preprocessing techniques were implemented which include cleaning duplicated data and transformation. After cleaning redundant data, loading annotated images was the first step to perform those techniques. Loaded images were renamed with their respective multi-labeled annotation and they are encoded after extracting the labels in the data frame as explained in section 5.3.1.3. Then resizing images to 128 x128 square size and conversion to an array is applied.

Data Re-Scaling: Data rescaling is implemented by normalizing images to project image pixels to a (0, 1) range. Image normalization is crucial for all images to contribute fairly to the total loss instead of varying loss due to different ranges of pixel values. Also, it is used to maintain a standard learning rate for all images since a different range of pixels requires a different learning rate.

```
#Minor Image Preprocessing
def preprocess_image(self, img_path):
        im = Image.open(img_path)
        im = tf.keras.utils.load_img(img_path,target_size=(128,128,3), color_mode = "rgb")
        im = tf.keras.utils.img_to_array(im)
        im = cv2.normalize(im, None, alpha=0,beta=255, norm_type=cv2.NORM_MINMAX)
        return im
```

Sample code 3 image preprocessing

5.3.1.5. Image Augmentation

Image augmentation is applied to preserve balanced class distribution of image data. After splitting the image dataset by 80: 20 data splitting ratio to training and testing set before performing data augmentation since the testing set should not be augmented. The selected augmentation techniques are rotation, height and width shift, shear, brightness, zooming, flipping horizontally and vertically, and channel shift to expand the training dataset with more variation. To perform all the augmentation techniques Keras image preprocessing library is used that has a class called ImageDataGenerator which enables the whole augmentation techniques in it.

Shifting: from selected augmentation techniques one of them is shifting image pixels in a particular direction. There are two types of shifting augmentation horizontal and vertical shifting and both of them are implemented. The sample image implemented shifted horizontally and vertically is shown in the implementation result in chapter six. The implementation returns an image pixel shifted in the x and y direction and clipped while specified new pixels in some regions.

Rotation: a random rotation of the image in a certain direction in this case 45 degrees in a clockwise direction. It gives the site of an image from a different angle of a site without changing the image information. An empty area of images due to rotation is filled by the reflected pixel values.

Flipping: images are flipped in both horizontal and vertical directions by reversing rows vs columns of image pixels. This implementation results in a mirror reversal of the original image on the x or y axis based on the given parameter.

Channel shift: random change or shift of channel values by channel shift range equals 80 as shown in the next sample code implementation.

Zooming: image is randomly zoomed in a range of 0.5 to 1.0 by interpolating pixel values of images. Therefore images show a variation in both width and height dimensions and different aspect ratios of the leaf image.

The implementation of all the augmentation techniques explained above is shown in the following sample code.

```
datagen = ImageDataGenerator(
        rotation_range=45,# 45 degree rotation
        width_shift_range=0.1,  #horizontal shift by  10 %
        height_shift_range=0.1, #vertical shift by 10 %
        shear_range=30.0,
        brightness_range=[0.2,0.9],# random brightness change b/n 0.2-0.9
        zoom_range=[0.5, 1.0],# random zoom b/n 0.5 and 1.0
        horizontal_flip=True,
        vertical_flip=True,
        channel_shift_range=80.0,#shifting image  pixels by 80 pixels
        fill_mode='reflect')#filling augmented images by reflection
```

Sample code 4 Image data augmentation

5.4. Proposed Model Implementation

The proposed model contains three main module blocks as explained in Figure 15. The model architecture block A include an input layer and six convolutional layers with a pooling layer followed after each pair of convolutional layer total of 3 pooling layer are used to construct the feature sharing layers. Block A layers handle the feature extraction part for both severity and disease classification. Then it shares the extracted features to both specific feature extractor shuffleNet units. ReLU activation function and batch normalization are applied while constructing the model as explained in CHAPTER FOUR.

Keras functional API with Tensorflow backend was used to build the proposed deep learning model algorithms. Functional API enables the design of more flexible models to create complex

models other than a sequential model that does not allow for the creation of models that share layers or have multiple input or output layers. The model structure contains shared layers for feature-sharing modules and has multiple outputs by using Keras Functional API. Python code sample to implement the proposed model is listed below.

- A feature-sharing module consisting of six 2D convolutional layers and three poling layers in block A as shown in *Sample code 5*

```
inputs = Input(shape=(128, 128, 3), name="disease_severity_inputs")
inputs1 = Conv2D(3, (3, 3), padding='same', activation="relu")(inputs)
conv1  = Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(inputs)
conv1=BatchNormalization()(conv1)
conv2  = Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(conv1)
conv2=BatchNormalization()(conv2)
pool1  = MaxPooling2D((2, 2))(conv2)
conv3  = Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(pool1)
conv3=BatchNormalization()(conv3)
conv4  = Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(conv3)
conv4=BatchNormalization()(conv4)
pool2  = MaxPooling2D((2, 2))(conv4)
conv6  = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(pool2)
conv6=BatchNormalization()(conv6)
conv7  = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(conv6)
conv7=BatchNormalization()(conv7)
pool3  = MaxPooling2D((2, 2))(conv7)
```

Sample code 5 feature sharing layers

- A more complex convolutional structure for shuffleNet blocks is also made with the functional API that enables defining different layers and connecting them in pairs.

```
def shaflenet_block(inputs,channels,strides):
    if(strides==1):
        input1,input2=tf.split(inputs,2, axis=3)
    else:
        input1,input2=inputs,inputs
    input1_channel=int(input1.shape[3])
    input2=Conv2D(filters=channels//2, kernel_size=(1,1),padding="same")(input2)
    input2=BatchNormalization()(input2)
    input2=ReLU()(input2)
    input2=DepthwiseConv2D(kernel_size=3,strides=strides,padding='same')(input2)
    input2=BatchNormalization()(input2)
    input2=Conv2D(filters=channels-input1_channel, kernel_size=(1,1), padding="same")(input2)
    input2=BatchNormalization()(input2)
    input2=ReLU()(input2)
    if (strides==2):
        input1=DepthwiseConv2D(kernel_size=3,strides=strides,padding='same')(input1)#,
        input1=BatchNormalization()(input1)
        input1=Conv2D(filters=input1_channel, kernel_size=(1,1),padding='same')(input1)
        input1=BatchNormalization()(input1)
        input1=ReLU()(input1)#e
    output= tf.concat([input1,input2], axis=3)#Concatenate input1 and input 2
    output=channel_shuffle(output)
    output_channel=int(output.shape[3])
    return output
```

Sample code 6 Shuffle Net Block

- A single stage contains 3 blocks which is shuffleNet block where the first block should always contain a stride of 1 and the rest have a stride of 2 which is one of the shuffleNet V2 rules

```
def stages(inputs, channel, num_blocks, group):
    x = inputs
    for i in range (num_blocks):
        x= shaflenet_block(x, channel, 2 if i == 0 else 1)
    return x
```

Sample code 7 Stage

- Block B and C perform feature extraction separately for disease and severity respectively before classification.

```
#Block B Shuffle unit
x1t = stages(pool3,512, 3, group)
x1f = Conv2D(1024, 1)(x1t)
x1=BatchNormalization()(x1f)
x1=ReLU()(x1)#el
x1=AvgPool2D()(x1)
flat1  = Flatten()(x1)

#Block C Shuffle unit
x2 = stages(x1t, 1024, 3, group)
x2= Conv2D(1024, 1)(x2)
x2=BatchNormalization()(x2)
x2=ReLU()(x2)#el
x2=AvgPool2D()(x2)
flat2   = Flatten()(x2)
```

Sample code 8 Block B and C Auxiliary Structure

- The last block containing fully connected layers for both disease and severity classification implementation is shown in the following sample code

```
[ ]  #Disease classification
     dense1 = Dense(1024, activation="relu")(flat1)
     dense1b=BatchNormalization()(dense1)
     dense1d = Dropout(0.8)(dese1b)
     dense2 = Dense(1024, activation="relu")(dense1d)
     D_output = Dense(8, activation="softmax", name="disease_catagory")(dense2)#class=7
     #Severity classification
     dense1_2 = Dense(1024, activation="relu")(flat2)
     dense1_2b=BatchNormalization()(dense1_2)
     dense1_2d = Dropout(0.8)(dense1_2b)
     dense2_2 = Dense(1024, activation="relu")(dense1_2d)
     S_output = Dense(4, activation="softmax",name="severity_catagory")(dense2_2)#classes=3
     model  = Model(inputs=inputs, outputs=[S_output,D_output],name="ResearchModel")
```

Sample code 9 Block D Classification layers

5.4.1. Model Compilation and Training

The constructed model has been compiled with categorical-cross entropy. An Adam optimizer is also used for optimizing the model with accuracy metrics. The model is compiled with different parameters sated to train the model and evaluate the ability to generalize with different weight loss for each classification.

```
#Model compilation
learning_rate = 0.001
epochs =80
opt = Adam(learning_rate=learning_rate, decay=learning_rate / epochs)
model.compile(optimizer=opt,
              loss={
                 'severity_catagory': 'categorical_crossentropy',
                 'disease_catagory': 'categorical_crossentropy',},
              loss_weights={
                  'severity_catagory': 0.5,
                  'disease_catagory': 1.2},
              metrics={
                   'severity_catagory': 'accuracy',
                  'disease_catagory': 'accuracy'
                 })
```

Sample code 10 Model Compilation

Then Keras library data generator is used to generate training, validation, and testing dataset. The model is fitted with a training and validation set to train the model and finally, evaluated with the testing dataset to evaluate the performance of the model other than the dependent dataset. Before generating images with their corresponding disease and severity encoded labels, one hot encoding is implemented by changing labels to categorical.

```
def generate_images_training(self, image_idx, is_training, batch_size=16):
    images, diseases, severitys = [], [], []
    while True:
        for idx in image_idx:
            leaf = self.df1.iloc[idx]
            disease = leaf['disease_id']
            severity = leaf['severity_id']
            file = leaf['file']
            im = self.preprocess_image(file)
            diseases.append(to_categorical(disease, len(dataset_dict['disease_id'])))
            severitys.append(to_categorical(severity, len(dataset_dict['severity_id'])))
            images.append(im)
            if len(images) >= batch_size:
                yield np.array(images), [np.array(severitys),np.array(diseases)]
                images, severitys, diseases= [], [], []
        if not is_training:
            break
```

Sample code 11 Image Generation

Keras' model.fit () enables us to perform training with train data and validate it using validation data for assessing the model's ability in different data. To reduce computational

complexity batch size of 32 is used while fitting the dataset rather than fitting the whole data set at once

```
history = model.fit(train_gen,#model1
                    steps_per_epoch=int(len(train_idx)//batch_size),
                    epochs=epochs,
                    callbacks=[callbacks],
                    validation_data=valid_gen,
                    validation_steps=int(len(valid_idx)//batch_size))
```

Sample code 12 Fitting proposed model

5.5.Model Testing and Evaluation

For model testing and evaluation, K-fold cross-validation is implemented. Holding the performance value in each fold iteration, the model testing and evaluation return a confusion matrix with the corresponding accuracy for each class. Since we use K-fold cross-validation the model training and evaluation iterates for outer and inner each fold.

The evaluation analysis is done by using different evaluation metrics and visualization methods. Metrics such as recall, f1 score, and precision are used for the classification report. Then visualization of the model performance is performed by using a confusion matrix and mat-plot library loss and accuracy graphs to show the performance of the model.

```
#prediction
sevrity_pred,disease_pred = model.predict(test_gen,steps=len(test_idx)//1)
yd_pred = np.argmax(disease_pred , axis=1)
ys_pred = np.argmax(sevrity_pred , axis=1)
```

Sample code 13 Model prediction

```
#classification report
print(confusion_matrix(test_yd.astype(str), yd_pred.astype(str)))
print(confusion_matrix(test_ys.astype(str), ys_pred.astype(str)))
predicted_Dclass_indices=np.argmax(disease_pred ,axis=1)
predicted_Sclass_indices=np.argmax(sevrity_pred,axis=1)
print(classification_report(test_yd.astype(str), predicted_Dclass_indices.astype(str)))
print(classification_report(test_ys.astype(str), predicted_Sclass_indices.astype(str)))
```

Sample code 14 Classification report

Finally, a saved model makes predictions for a set of randomly selected leaf images.

```
model.load_weights(checkpoint_filepath)
model.compile(optimizer=opt,
              loss={
'severity_catagory': 'categorical_crossentropy',
                  'disease_catagory': 'categorical_crossentropy',},
              loss_weights={
                   'severity_catagory': 0.5,
                   'disease_catagory': 1.2},
              metrics={
                    'severity_catagory': 'accuracy',
                   'disease_catagory': 'accuracy'
                  })

sevrity_pred,disease_pred = model.predict(test_gen,steps=len(test_idx)//1)
yd_pred = np.argmax(disease_pred , axis=1)
print("disease pred:",yd_pred)
ys_pred = np.argmax(sevrity_pred , axis=1)
print("severity pred",ys_pred)
```

Sample code 15 evaluation demo for randomly selected images

CHAPTER SIX

6. RESULT AND DISCUSSION

6.1. Chapter Overview

The background of the study and the initiative problem that led to the proposal of this research, the related work, the methodology, and the proposed model with its implementation have all been briefly covered in the previous chapters. The results that have been recorded after putting various experiments into practice are illustrated in this chapter. The data acquired from each training session using various validation techniques is presented along with a comparison of the suggested model. Each experiment carried out with various hyper parameters is displayed using a variety of visualizing techniques, including graphs, tables, and figures.

6.2. Disease and Severity Classification

The protection or pathology department's duty for one of the primary study topics in agriculture is the classification of diseases and their severity. Deep learning in agriculture also has applications for predicting the class of these two classifications from an image. The presence, type, incidence, and severity of a leaf disease can all be predicted or classified using deep learning. The categorization model's output can be used to statistically analyze seasonal and general leaf diseases so that a pathologist can get rural farmers the help they need. In the absence of scouts, rural farmers can also employ a classification model that can be put into practice for accurate disease and severity prediction findings. After conducting numerous tests with the suggested classification model, the results obtained during the model's evaluation and training have been described in the following subsections.

6.2.1. Augmentation Results

Numerous augmentation strategies have been used to expand data size, as discussed in section 3.5.6. While adding more variance to the input so that the model perceives it as new data, the chosen augmentation helps to preserve data that can make sense for the intended purpose. Different augmentation approaches have been considered in order to get better outcomes, and

the impact of each technique implementation has been observed. Multiple procedures are selected, applied to each class, and some of the unbalanced data variations are balanced to maintain more balanced data than the original data. The following figure displays the outcome of a sample picture augmentation implementation.

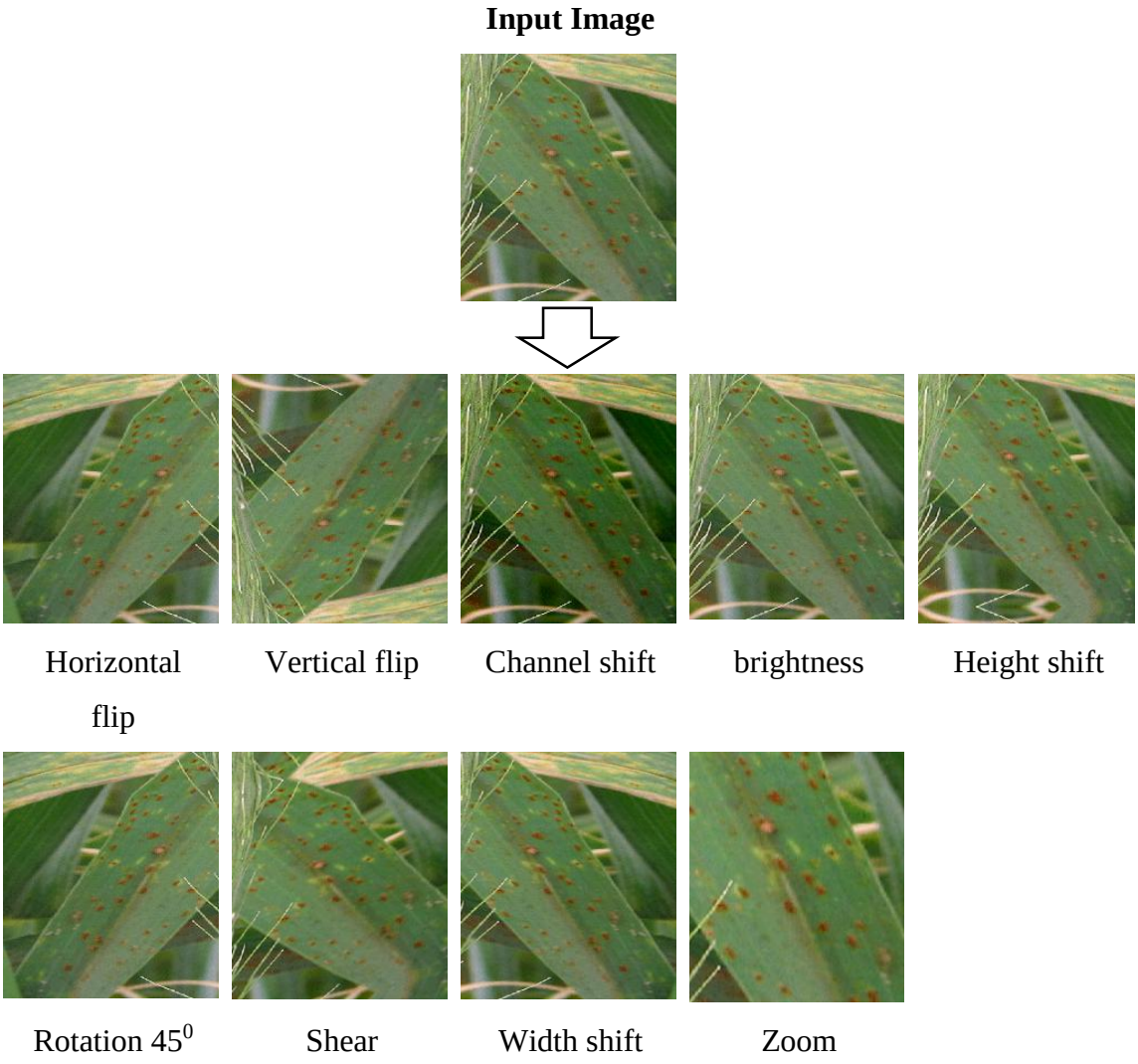


Figure 20 Augmentation result

6.2.2. Hyper Parameters

With certain parameters, the proposed convolutional neural network model is implemented. Hyper parameter optimization is used to train the model, and a productive model with less computational resources is chosen. The following table includes a list of some of the typical

hyper parameters that were used to train the model. By comparing the results from training and evaluation, several parameters are improved to produce a better evaluation of the model.

Table 6 Hyper parameters

Parameters		Value
Optimization		Adam
Learning rate		0.01 ,0.001,1e-4
Epoch		100
Weight loss	severity	0.1, 0.2, 0.4,0.5,
	disease	1.2, 1.5
Size		128x128x3
Group (ShuffleNet)		8
Channel(ShuffleNet)		24
Dropout		0.8
Batch size		32

6.2.2.1. Classification Model Hyper Parameter Tuning

For a better model training effect, model hyper parameter optimization is required. Randomized search hyper parameter tuning is an optimization approach that is used to modify the model. A model is trained using the randomized-search hyper parameter optimization method using randomly chosen hyper parameters. The short time span for other tuning methods is the reason this technique was chosen. There are five primary training rounds based on this tuning procedure, and the results acquired with various combinations of hyper parameters are displayed in the following tables.

Table 7 classification accuracy of severity and disease classification

No.	Learning rate	Disease weight loss	Severity weight loss	Epochs	Drop out	Batch size	Kernel size
Round 1	1e-4	1.5	0.1	60	-	64	3x3
Round 2	1e-4	1.5	0.2	100	-	32	3x3
Round 3	1e-4	1.5	1	50	-	5	*3x3*
Round 4	0.001	1.2	0.4	100	0.9	32	3x3
Round 5	0.01	1.2	0.2	50	0.9	32	3x3

After each training session, the mean validation and training accuracy and loss results are collected, and the best hyper parameter combination is chosen based on the above combination. Table 8 and Table 7. As illustrated in those tables the figures are mean accuracy and mean loss of each round training.

Table 8 Severity and disease classification mean validation and training loss

No.	Disease Train Loss	Disease Val Loss	Severity Train Loss	Severity Val Loss
Round 1	0.0888	1.0141	0.0493	2.0678
Round 2	0.0893	0.8150	0.0350	2.1043
Round 3	0.0900	0.8155	0.9857	2.2074
Round 4	0.1053	0.7780	0.0350	2.1543
Round 5	**0.0614**	**0.6697**	**0.0277**	**1.9754**

Even when the extra-large loss is indicated by the severity classification, it is still possible to select the best set of hyperparameters. High data variance and an overfitting incidence are the primary causes of the poor severity validation loss. Accordingly, round 5 hyper parameters with L1 and L2 regularization in combination with dropout are taken into consideration for the subsequent training trial based on the results.

Table 9 Severity and disease classification mean validation and training accuracy for tuning proposed model with different parameters

No.	Disease Train Accuracy	Disease Valid Accuracy	Severity Train Accuracy	Severity Val Accuracy
Round 1	0.9338	0.7794	0.8776	0.6051
Round 2	0.9324	0.8107	0.9001	**0.6814**
Round 3	0.9209	0.8352	0.8957	0.6440
Round 4	0.9518	0.8452	0.9019	**0.7714**
Round 5	**0.9606**	**0.8501**	**0.9081**	0.7501

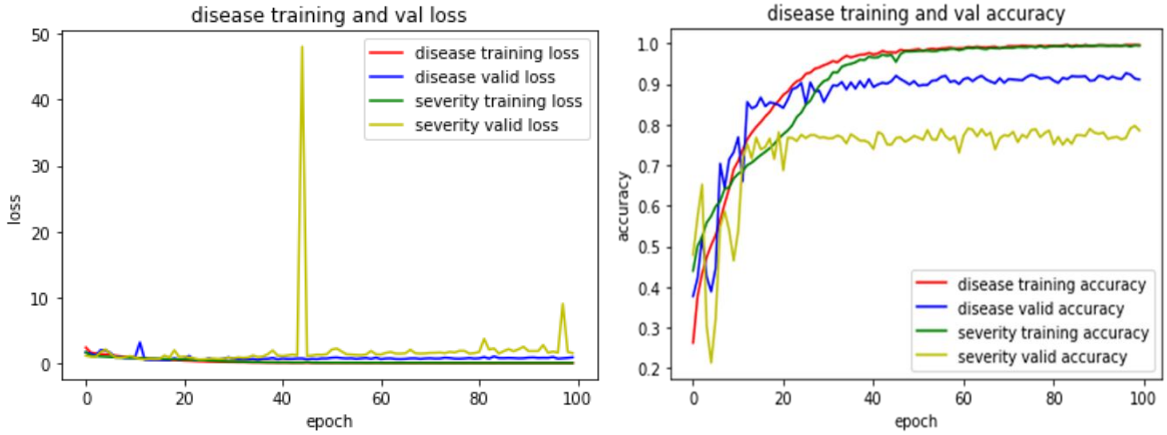


Figure 27 Accuracy and Loss Analysis figure for 5th round

As seen in the above graph and table, the round 5 training hyper parameter combination with learning rate 0.01 and 100 epoch without drop results in greater accuracy and loss. Even though it indicates overfitting, its combination still performed better than the other five hyper parameter clusters. As a result, this combination has been chosen for training the classification model. Regularization techniques have been used to combat overfitting after choosing the best hyper parameter combination.

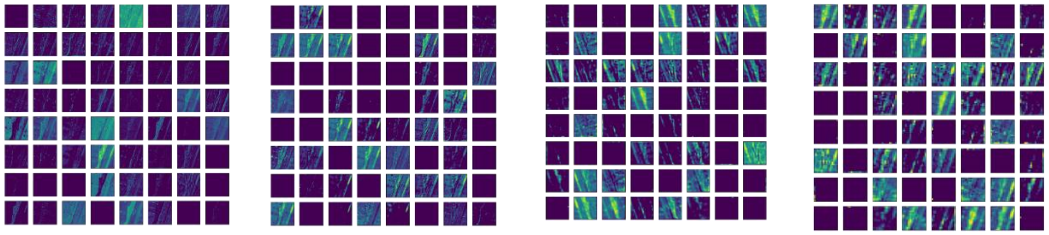
6.3.Regularization Result

Whenever time overfitting happens, several regularization approaches should be taken into account. Increasing the amount of data, stopping early, dropping out, using L1 and L2 regularizers, and eventually training with the k-fold cross-validation method are some of the common techniques. Instead of memorizing the pattern of each image, dropout and K-fold cross-validation procedures have been included to make the model more general. The data acquired using the k-fold cross-validation approach and various regularization techniques are displayed in the next sub-section.

As seen in the following figure, feature maps gathered after each convolutional layer in each distinct module are presented. The original image that was provided to the suggested model to extract the feature maps that are shown after the original images is the following image. The following figure displays the model's post-training output feature maps.

Figure 28 input image for the proposed model

Figure 29 Visualization of feature maps of the model in each stage before training



Feature maps of Feature Sharing Module (A)

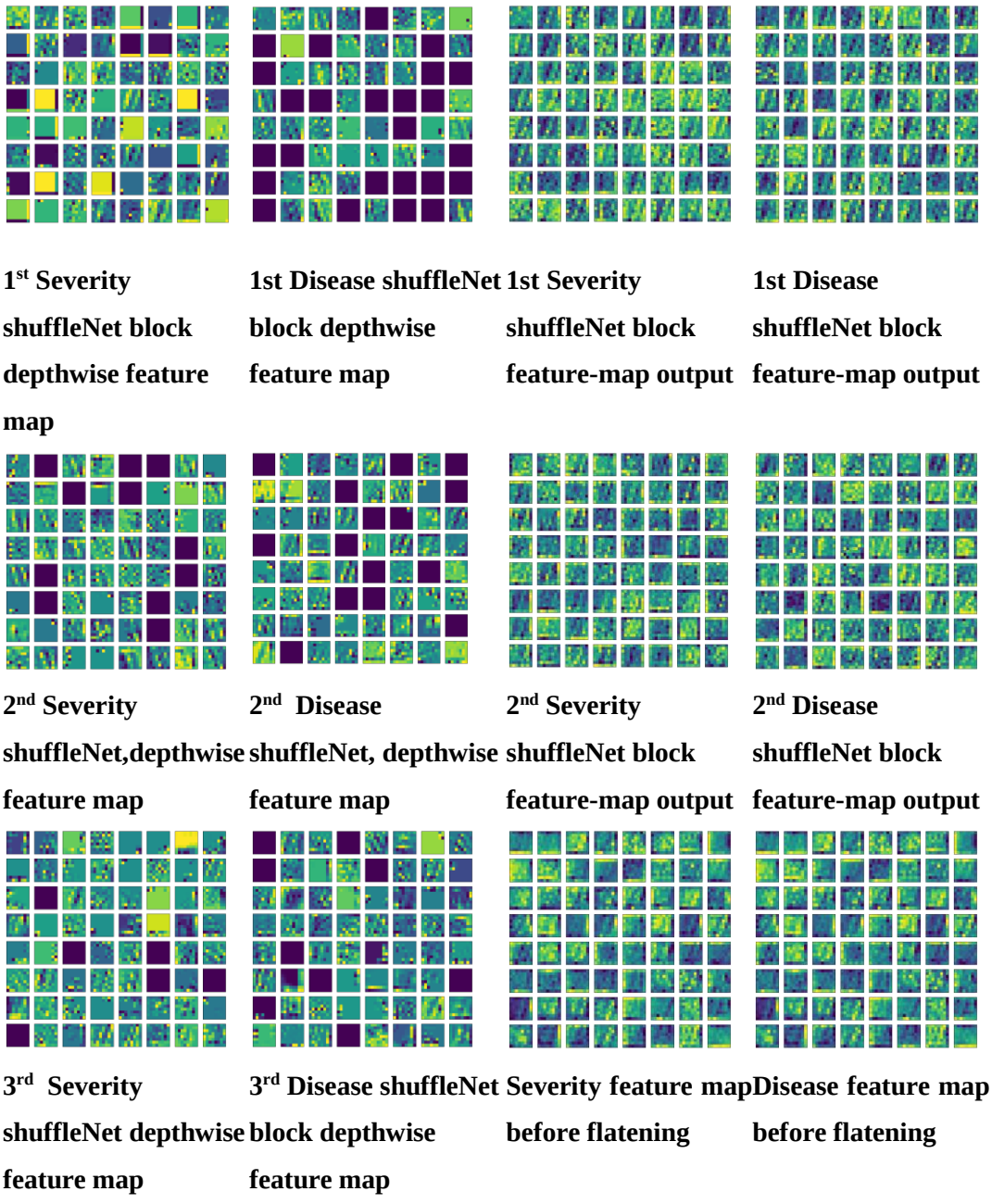


Figure 30 Visualization of feature maps of the model in each stage after training

6.3.1. K-Fold Cross Validation Results with Different K-Values

To combat the overfitting problem and enhance model performance. One of the primary techniques is K-fold cross-validation, as described above and in Chapter 3 section 3.10.1.The K fold cross validation strategy is chosen despite the fact that there are many different cross validation techniques. It allows the model to apply previously learnt or optimized weights in

later training cycles. In order to improve model performance by preventing or lowering overfitting, the second experiment uses the K fold cross-validation procedure with different K values. The minimum number of folds is two, however five-fold and ten-fold cross-validation tests are carried out due of the size of the training data.

Result When Fold =5: The dataset is split into five subclasses and the training is folded into five folds when k is equal to 5. The model performance for each fold is validated using a different subpart. The training epoch was set at 50 in this experiment to implement early stopping before the model is prone to overfitting. A 0.001 L1 value regularizing approach is paired with a dropout of 0.8 for better generalization. The outcomes of the loss and accuracy graphs are displayed below, respectively.

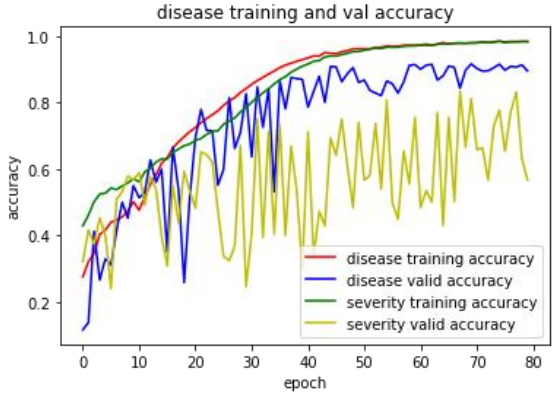


Figure 31 Accuracy evaluation graph of training and validation sets for disease and severity when K=5

The results gathered from each succeeding training iteration are displayed in the accuracy graph evaluation seen above, along with how the model gradually picks up the characteristics from each iteration. According to the diagram's first-fold accuracy graph, the maximum validation accuracy was around 0.7, but the following training iteration increased it to roughly 0.75-0.95. As a result, the two classification validation accuracy eventually ranged from 0.90-0.99, despite some outliers with lower validation accuracy.

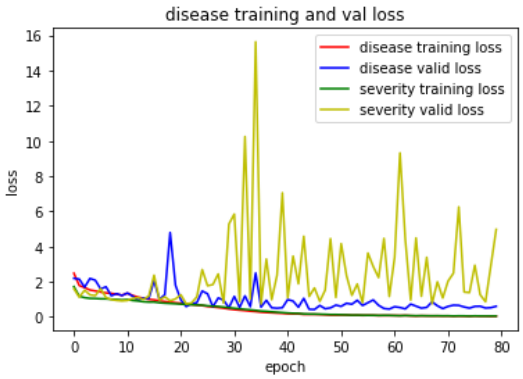


Figure 32Training and validation Loss graph for disease and severity when K=5

Results at k=5 show some data to be outliers and moderate overfitting. This may be seen in the training and validation results' loss and accuracy graphs in the graphs above. When compared to the disease classification graph, particularly severity validation accuracy and loss, there is a moderate overfitting. But in each training iteration, the loss graph similarly steadily declined to almost zero.

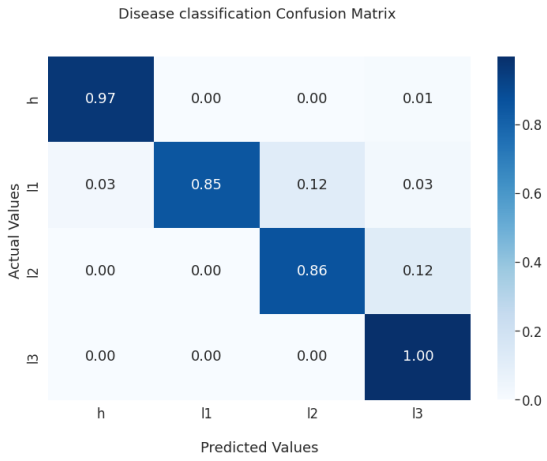


Figure 33 k=5 severity classification confusion matrix

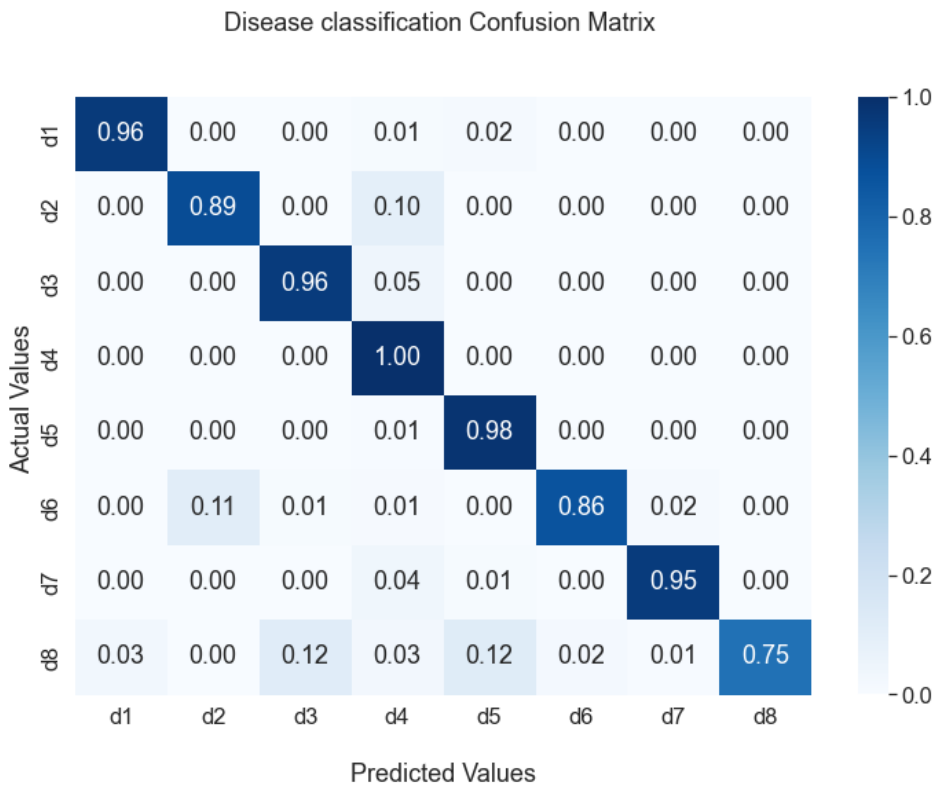


Figure 34 k=5 disease classification confusion matrix

The prediction result from the K=5 fold experiment is presented in the above two confusion matrices. The disease prediction confusion matrix shows that d2[11]11 below, d6[15]and d8[17] misclassification rate is higher than that of the other classes. The severity level classification prediction result on the test dataset is shown in the first confusion matrix, showing that levels 0 and 3 have a greater prediction result than levels 1 and 2.

In the following figure the classification report for both prediction is shown. The classification result also indicates lower precision result in disease d1[10], d2[11], and d3[12] whereas other category prediction shows a better precision result.

	precision	recall	f1-score
0	0.97	0.97	0.97
1	1.00	0.85	0.92
2	0.88	0.86	0.87
3	0.86	1.00	0.93
accuracy			0.92
macro avg	0.93	0.92	0.92
weighted avg	0.92	0.92	0.91

Figure 35 Severity classification report at k=5

	precision	recall	f1-score
0	0.97	0.96	0.96
1	0.88	0.89	0.89
2	0.89	0.96	0.92
3	0.80	1.00	0.89
4	0.87	0.98	0.92
5	0.97	0.86	0.91
6	0.98	0.95	0.96
7	1.00	0.75	0.85
accuracy			0.91
macro avg	0.92	0.92	0.91
weighted avg	0.92	0.91	0.91

Figure 36 Disease classification report when k=5

Apart from the fact that the suggested model made considerable progress in the five k-fold experiments, the severity level 2 prediction result showed decreased precision, recall, and f1 score.

6.3.1.1. Result when Fold =10

Similar to this, ten training iterations using a separate subset of validation data are carried out when k = 10; the training dataset is split into ten parts. After testing the 5 and 10 fold cross validation procedures, it was found that the 10 fold result performed better than the 5 fold, hence regularization was tested further. The suggested model is trained using a 10-fold cross-

validation method with several regularizations in order to compare and choose better performance. Lasso regression was utilized in the first trial, and Ridge regression in the second.

- **Experiment 1 with Lasso Regression:** When k = 10, the training dataset is split into ten subgroups, and ten training iterations are carried out using various subsets of the validation data. The lasso regression technique is used to carry out this experiment over a period of 80 epochs. An L1 regularization penalty with a 0.001 L1 value was used to penalize the weight values. The outcome at the conclusion of the tenth training fold is shown in the loss and accuracy graph below. Along with a classification of diseases, the graph also analyzes disease severity.

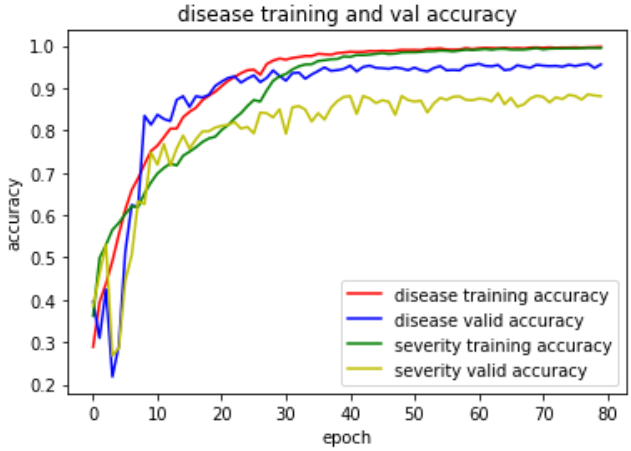


Figure 37 L1 Regularized accuracy evaluation graph of training and validation sets for disease and severity when k=10

Although the graph displays an outlier that produces poorer validation results, lasso regression with a K=10 fold was still able to produce a superior accuracy result. This is brought on by a large diversity of data, which results in a tiny subset of image data having distinct patterns from the rest of the training data. The following loss graph further illustrates the issue of outliers.

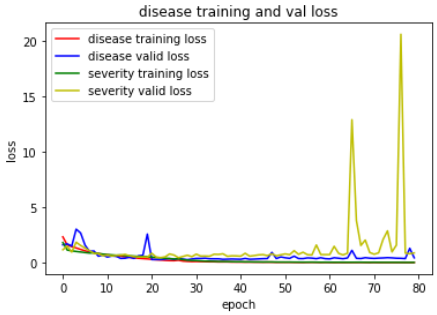


Figure 38 L1 Regularized loss evaluation of training and validation sets for disease and severity when k=10

The model gradually learns complicated features and was able to improve its validation result through continuous iterations, as shown by the accuracy and loss graph of training and validation. To provide a clearer picture of the model's performance, Experiment 2 model evaluation is also used. With the use of a confusion matrix that displays the model's effectiveness, predictions for 10% of the test data set are made. Where the confusion matrix contrasts the actual labels in the test data set with the first model prediction. The following confusion matrix is provided to provide a summary and visual representation of the classification algorithm performance.

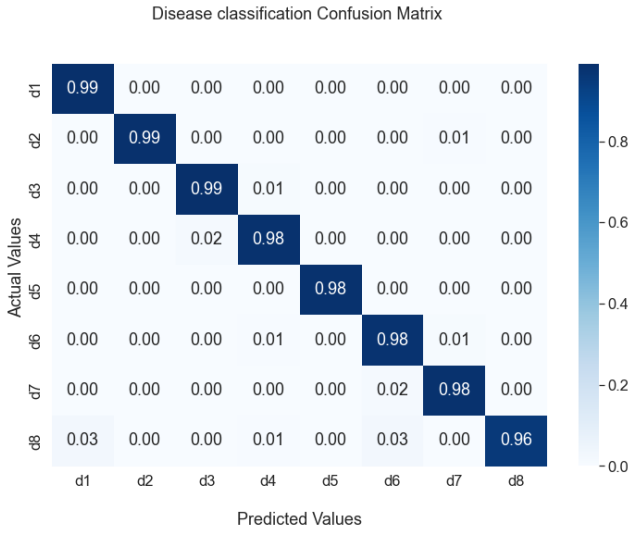


Figure 39 Experiment 1 Disease prediction confusion matrix

Where; d1[10], d2[11], d3[12], d4[13], d5[14], d6[15], d7[16], and d8 [17] are Healthy Maize, Common Rust M aize, Blight Maize, GLS Maize, Healthy Wheat, Leaf Rust Wheat, Steam Rust Wheat, and Y ellow Rust Wheat respectively. Despite category d8 showing an acceptable miss categorizatio n, the prediction result demonstrates that each class was correctly predicted. Other than that a lmost all classifications have been done well.

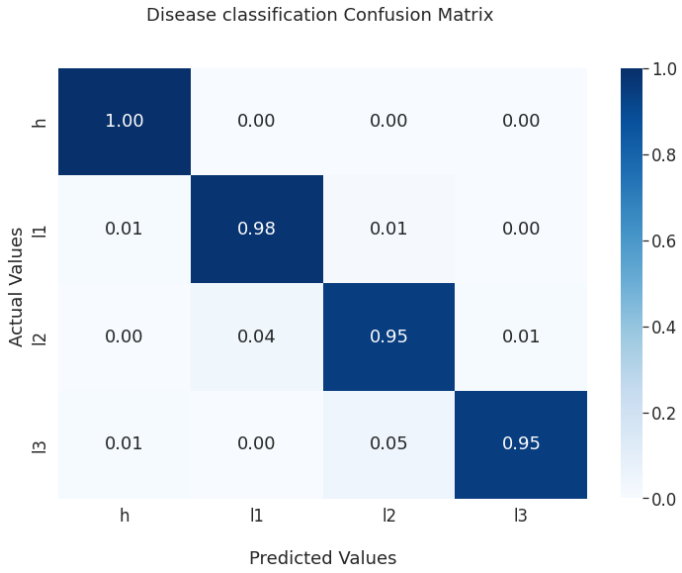


Figure 40 Experiment 1 Severity prediction confusion matrix

Where; H[18], L1[19], L2[20], and L3[21] are Healthy, Severity Level 1, Severity Level 2, and Severity Level 3 respectively. The classification report for L1 regularization for both severity and disease classification is shown in the following images. *Figure 40* shows Experiment 2 severity prediction in percentage, which suggests higher accuracy in the category of healthy

[10]d1= healthy Maize
[11] d2=Common Rust Maize
[12] d3= Blight Maize
[13] d4=GLS Maize
[14] d5= Healthy Wheat
[15] d6= Leaf Rust Wheat
[16] d7= Steam Rust Wheat
[17] d8= Yellow Rust
[18] H=healthy
[19] L1=Severity Level1
[20] L2= Severity Level 2
[21] L3=Severity Level 3

and level 1 classification while a reasonable ability to classify is displayed in L2 and L3, even if it's a good result, more can be done.

	precision	recall	f1-score
0	0.98	1.00	0.99
1	0.96	0.98	0.97
2	0.94	0.95	0.94
3	0.99	0.95	0.97
accuracy			0.96
macro avg	0.97	0.97	0.97
weighted avg	0.96	0.96	0.96

Figure 41 Classification report for L1 regularized Severity classification

The severity classification report for Experiment 1 displays the same outcome, with levels 0 and 1 having the highest recall and level 0 and level 3 having higher precision, while f1-score displays better results in all categories with the exception of level 2. Additionally, the overall accuracy of 0.96 obtained from prediction shows that the suggested model is effective at classifying severity categories.

	precision	recall	f1-score
0	0.97	0.99	0.98
1	1.00	0.99	0.99
2	0.98	0.99	0.98
3	0.98	0.98	0.98
4	1.00	0.98	0.99
5	0.95	0.98	0.96
6	0.98	0.98	0.98
7	0.99	0.96	0.97
accuracy			0.98
macro avg	0.98	0.98	0.98
weighted avg	0.98	0.98	0.98

Figure 42 Classification report for L1 regularized Disease classification

With the exception of disease categories 5 and 7, which both had 0.97 f1 scores with 0.96 recall and 0.95 precision, practically all categories in Figure 42 achieved greater prediction accuracy of 0.98 to 1. Overall, experiment 1's findings indicate a disease classification accuracy of 0.98 on average.

Experiment 2 with Ridge Regression

The following experiment used K-fold cross-validation with a different kernel regularizer and the same K value, which is 10. In this experiment, the overfitting issue is addressed by using an L2 regularization technique, often known as ridge regression. By including the prediction result, it will be simpler to choose the classification model that performs better. In order to achieve this, a combination of the same dropout of 0.8, 60 epochs, and an L2 regularization with 0.001 hyper parameters is used. As a result, the following outcome is noted..

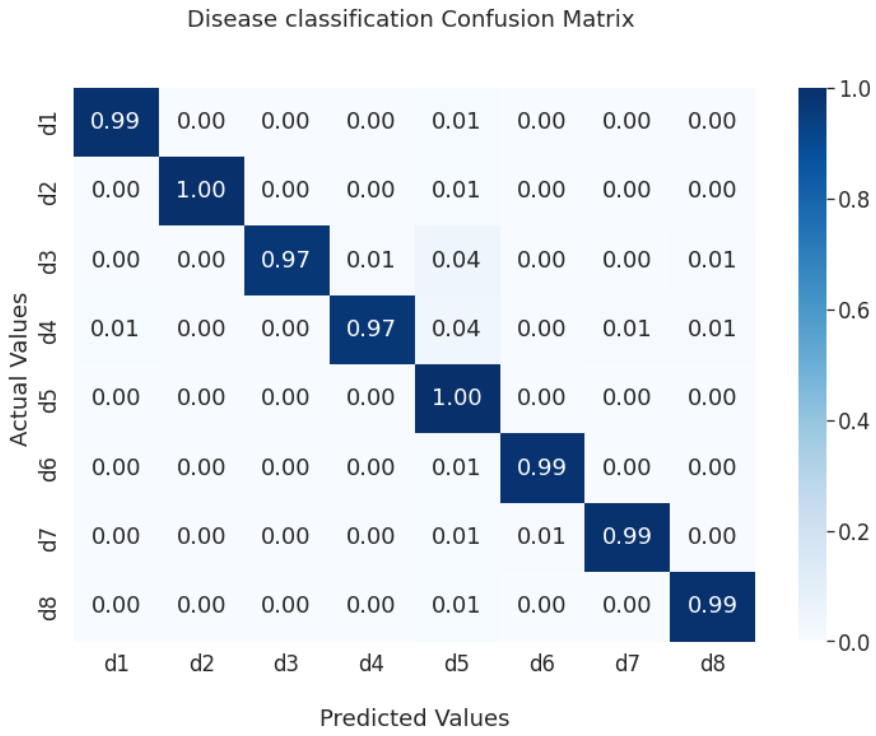


Figure 43 Confusion matrix for L2 regularized Disease classification

The confusion matrix demonstrates that experiment 2 performs well across all classifications, despite the fact that categories d3 (Blight Maize) and d4 show a minor missing prediction (GLS maize). Even said, the model might still perform best for all other classes.

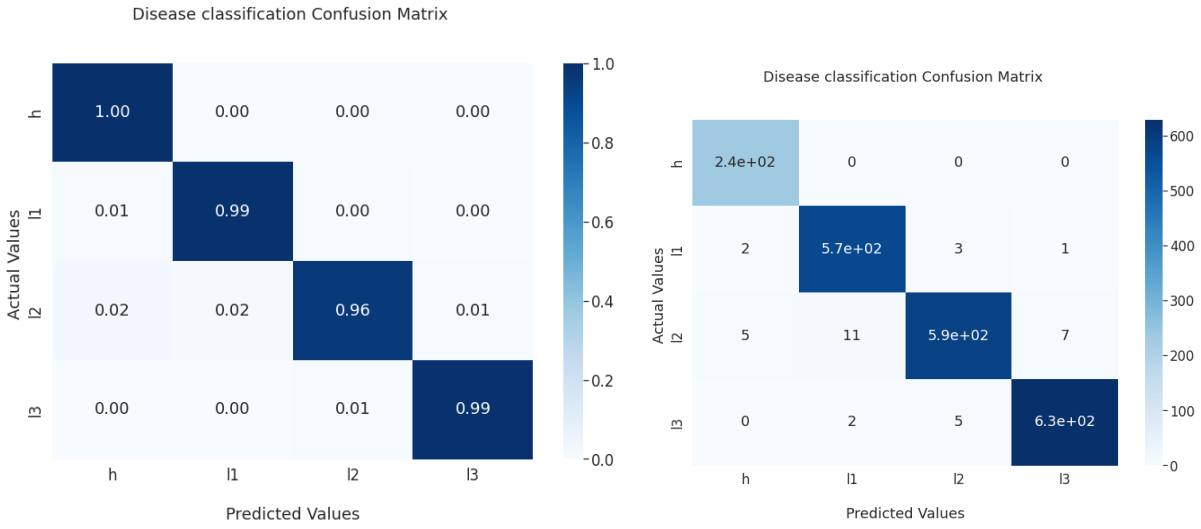


Figure 44 Experiment 2 Severity prediction confusion matrix in percentage and figure

The two images above show the confusion matrix, which is depicted in percentages as well as in the figure that shows the proportion of predictions that were accurately made and missed for each class. The following graphs demonstrate the disease and severity classification of training and validation's loss and accuracy graphs.

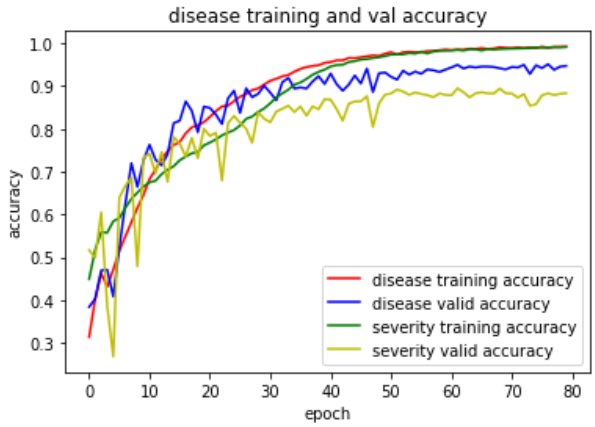


Figure 45 L2 Regularized accuracy evaluation of training and validation sets for disease and severity when k=10 for each fold

The loss diagram shows that the proposed model with L1 regularization have a shown declining gradient as the graph indicates the scale of the loss graph range reduces through the increase of training iterations.

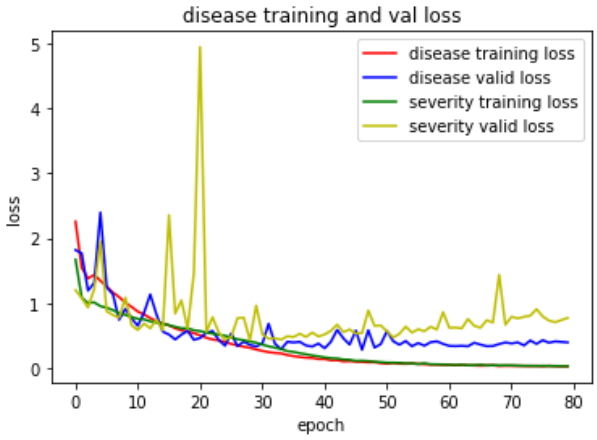


Figure 46 Experiment 2 L2 Regularized loss evaluation of training and validation sets for disease and severity when k=10 for each fold

Similar to this, the accuracy diagram shows that both types of categorization have increased their validation accuracy as the fold number has increased. As a result, the model's performance has increased generalization while decreasing overfitting. Experiment 2 classification report demonstrated significant progress over the previous experiment, with all categories achieving precision, recall, and f1-scores greater than 0.97. According to the above analysis of different k values and regularizing methods, Experiment 2 with k=10 and L2 regularization performs significantly better for all 7 disease classes and 4 severity classes. Additionally, both classes' overall accuracy was raised to 0.98.

	precision	recall	f1-score
0	0.99	0.99	0.99
1	1.00	1.00	1.00
2	1.00	0.97	0.98
3	0.99	0.97	0.98
4	0.89	1.00	0.94
5	0.99	0.99	0.99
6	0.99	0.99	0.99
7	0.98	0.99	0.99
accuracy			0.98
macro avg	0.98	0.99	0.98
weighted avg	0.99	0.98	0.99

Figure 47 Experiment 2 disease classification report

The proposed model performs better than practically all illness categories, as seen in the image above. Although the precision and f 1 score in category number 4, Healthy Wheat Images, are slightly lesser than those in other categories, the prediction outcome is still superior. The severity prediction classification report for a proposed classification model is shown in the following figure. The following series of numbers show a categorization report that includes recorded precision, recall, and f1 score. The severity prediction result from Experiment 2 shows that all severity categories have improved since the previous experiment, but severity level 2 still has a moderate prediction performance.

	precision	recall	f1-score
0	0.97	1.00	0.99
1	0.98	0.99	0.98
2	0.99	0.96	0.97
3	0.99	0.99	0.99
accuracy			0.98
macro avg	0.98	0.99	0.98
weighted avg	0.98	0.98	0.98

Figure 48 Experment1 severity classification report

The best performing model out of the three experiments is chosen based on the experimental results. The proposed model is chosen after being trained using the 10 fold cross validation method and optimized with ridge regression as indicated in the following bar graph.

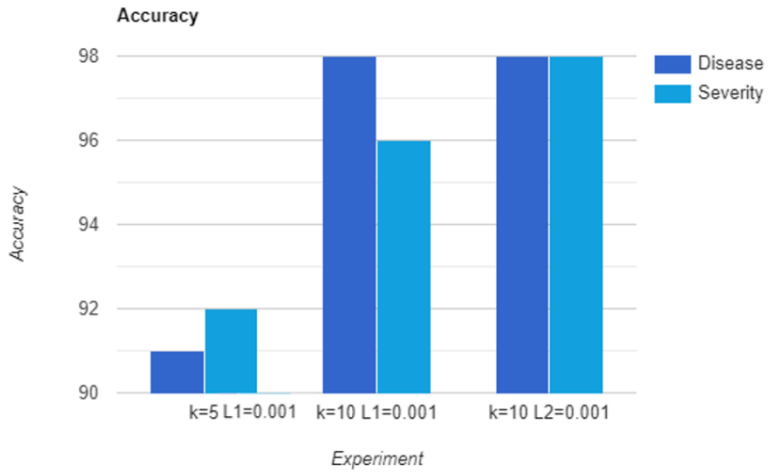


Figure 49 Results for different k-fold cross validation and regularization

11 randomly chosen leaf images are utilized to present the classification result in order to demonstrate the prediction result. The full image samples with their predicted results are presented in Appendix F for these images, which have both disease and severity categorization data gathered.

6.4.Comparison with Previous Model

The VGG model was selected since it is the cornerstone for the suggested model's improvement when compared to other earlier work. By adding a second classifier with three fully connected layers, the VGG mode should be improved to a multi-output model for comparison. The result suggests extremely strong disease classification accuracy and a moderate severity classification in each class, according to the prediction confusion matrix and classification report. The suggested model clearly outperforms Experiment 2, a highly successful experiment, in nearly all categorization findings.

For k=10, the baseline work is trained using 10 folds of cross validation, after which the model weight is saved and the subsequent fold's model is initialized using the saved weights to learn about its flaws, much like transfer learning. As a result, it will learn features ranging from basic to advance over the course of ten repetitions.

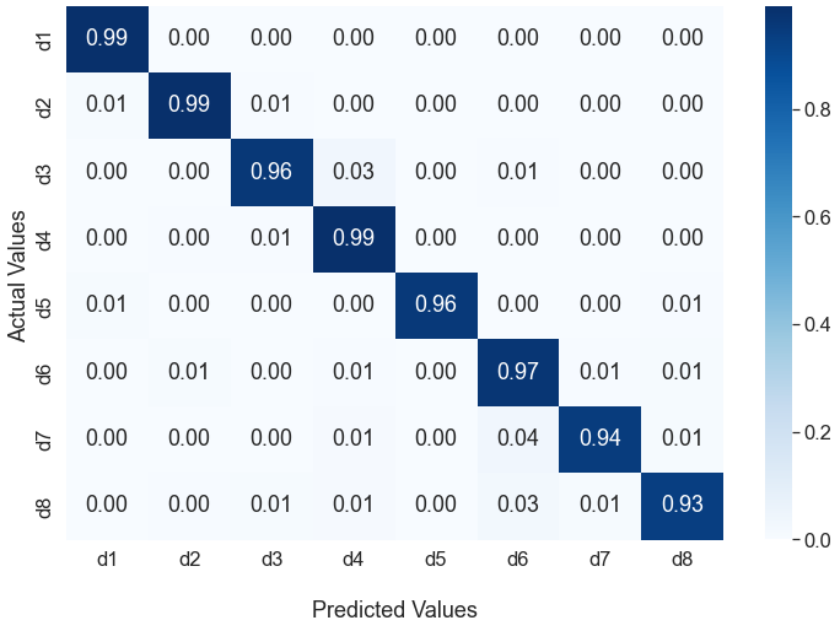


Figure 50 Disease prediction result from Adopted VGG

The prediction result for Adopted VGG demonstrates that the model works well, as seen in the confusion matrix above, with the exception of a moderate miss classified in the d8 and d7 disease categories.

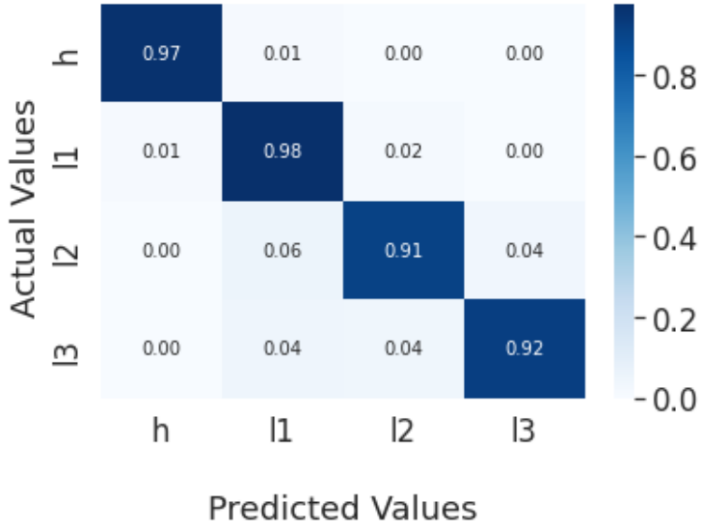


Figure 51 Severity prediction result of adopted VGG

The severity classification confusion matrix demonstrates that the selected model was able to detect healthy and severity level 1 leaves with a better degree of accuracy, but when compared to the other categories, it exhibits moderate mis-prediction in L2 and L3.

	precision	recall	f1-score
0	0.99	0.97	0.98
1	0.90	0.98	0.93
2	0.94	0.91	0.92
3	0.96	0.92	0.94
accuracy			0.94
macro avg	0.95	0.94	0.94
weighted avg	0.94	0.94	0.94

Figure 52 VGG severity prediction classification report

	precision	recall	f1-score
0	0.98	0.99	0.99
1	0.98	0.99	0.99
2	0.97	0.96	0.97
3	0.93	0.99	0.96
4	1.00	0.96	0.98
5	0.93	0.97	0.95
6	0.98	0.94	0.96
7	0.97	0.93	0.95
accuracy			0.96
macro avg	0.97	0.97	0.97
weighted avg	0.97	0.96	0.96

Figure 53 VGG disease prediction classification report

This leads to the model's overall performance for disease categorization and severity being 0.94 and 0.96, respectively. The confusion matrix and classification report are displayed below.

6.5.Discussion

In this study, a proposed model was put into practice using a large dataset, accounting for both best performance and computing costs. The model was trained using a large dataset with plenty of variation. As described in the document, a variety of data sets are incorporated in order to make the model robust to various test datasets. As a result, as the recorded data demonstrate, some of the tests have shown mild or modest overfitting. The model can detect and cover all real-world cases, allowing it to perform better in test datasets where there would be less

fluctuation in the test set. However, constructing a model and applying it with a huge dataset enables the model to do so.

As described in the findings sections, several methods have been utilized to minimize overfitting. A cross validation is used to learn the fundamental features in different training iterations, making the model robust against various variances. The model eventually performed well on the test dataset after learning advanced features by optimizing stored weights in each iteration. Experiment 2 with ridge regression outperforms the other experiments after three experiments using k-fold cross-validation methods with ten iterations. Where the same data were used for each trial. 98 percent classification accuracy for disease and severity is attained in Experiment 2 with L2 = 0.01. It has been chosen for next-level comparison as a result.

A modified VGG was selected in order to compare the basic model with the suggested one, as evidenced by the findings of the prior experiment. The same data and number of classes are used for training and evaluation of both the suggested model and the chosen VGG model for multi-labeling. As demonstrated by the two models' classification, the suggested model performs better than the baseline work. Also, taking the number of parameters used for each model into consideration, the proposed model reduces the number of parameters by half that of the VGG model. A larger number of parameters results in a more complex function, which may allow it to learn more intricate patterns, but it also has a disadvantage. The computing cost rises as the number of parameters does as well. Overfitting is caused by a high number of model trainable parameters, which lengthens the training process.

A comparison of the suggested model's and the VGG model's chosen parameter counts is shown in Table 11. Additionally, multi-classification of a single labeled model, which is a predefined model for large picture training termed VGG16, is included in the comparison of parameters in the table. As shown in the table, the suggested model performs better with the same training dataset while having fewer trainable parameters than both the adopted and pure VGG models by a factor of more than half. Additionally, there has been a 28.5% decrease in training time, which is outstanding progress.

Table 10 Parameter Comparison between Models with 123 square image size

No.	Model	Drop-out	Trainable parameters	Non-trainable parameters	Total parameters	Computation-time
1	VGG	-	119,507,756	0	119,507,756	-
2	Adapted VGG	0.8	115,476,556	33,024	115,509,580	125hr+29min
3	Proposed Model	0.8	**31,323,212**	**26,880**	**31,350,092**	***89hr+45min***

For this study, three primary research questions were chosen that should be addressed throughout the conduct of the study. Following the completion of the research, each question is addressed in light of the experiment's results. Even though the majority of the questions are only partially addressed throughout the document, the following paragraphs provide a brief and precise summary of each question and its corresponding solution.

RQ1: What would be the effect of the variation of real field data on classification model performance?

Attempts to incorporate a wide range of real-world data have an impact on the categorization model, both positively and negatively. The advantage of training the model with more diverse data is that it allows the model to accurately predict unrelated data. As a result of its adaptability, the model will be able to deal with any data. Because of this benefit, the model is resistant and robust to multi-variate and diverse feature data classification. The disadvantage of highly variable training data is that it makes it difficult for the model to fit highly variable patterns. As a result, more training data and time are required. According to the experiment results, certain outliers remained even after the cleaning procedure. As a result, striking a balance between the model's complexity and the variation in the data is critical.

RQ2: How to reduce the computational expense of the classification model while improving disease and severity prediction?

A shuffleNet structure can be used to reduce the computational cost of a CNN model for multi-label classification. For the proposed approach, a feature-sharing module with a shuffleNet structure is developed as a multi-labeling model. While the feature sharing module extracts common features for both disease and severity classification, the following module, which is a different shuffleNet module, shares features extracted from the sharing module, which has an impact on the need for a separate feature extractor for each classification task. As a result, those two shuffling structure modules will perform advanced level feature extraction with light weight CNN performing convolutions and shuffling.

The shuffling structure is composed of various shuffle blocks that use point- and depth-wise convolutional layers as their basic building blocks, reducing the number of parameters needed. The channel shuffle operation allows for extensive information sharing among feature channels. These two fundamental components are critical in lowering a model's computational cost while maintaining superior performance. It is also useful for training with imbalanced data. As a result, it is possible to draw the conclusion that combining shuffleNet CNN with a feature sharing module for multi-label classification has a significant impact on lowering computational costs by reducing the number of parameters required and the depth of the model.

RQ3: What would be the performance of the model on classifying disease and severity occurrence simultaneously?

As shown in the results section above, the proposed model outperforms the experiment results in both disease and severity classification. According to the evaluation, the model's accuracy for disease and severity classification is 0.98. The prediction result for both classifications indicates higher prediction results in almost all classes. As a result, the model prediction is considered highly accurate. In general, the model appears to perform capably on the two tasks.

However, since the model is multi-label classifier it has a high level of complexity, its weight-loss hyper parameter should be carefully tuned. Because the experiment revealed that identifying a suitable combination for both classifications is a kind of dilemma, therefore more experiments with different hyper parameters are required. However, but since multi-label classification has the advantage of lower computation costs and the ability to perform multiple tasks with a single model, it owe more than compensation for its tuning difficulty

CONCLUSION AND FUTURE WORK

Conclusion

A multi-label classification model for disease diagnosis and severity prediction is the goal of this study. With a single model and training data sets, it is possible to complete many jobs. In order to accomplish various classifications, the model extracts features from the leaf pattern and shares those characteristics with the extractors for disease and severity. In the suggested model, feature-sharing modules and shuffle units are combined. The proposed research aims to close the knowledge gap in leaf disease categorization left by similar work, which yields a less accurate model when tested on independent data. To accurately represent the testing data in the actual world, the developed model is trained and tested on vast amounts of maize and wheat data gathered from multiple data sources. To improve the evaluation result on independent data, the training dataset has been built up with a great deal of variety. The local data collection comprised 60% of the images with occluded leaf background representing complicated real-world fields and greenhouse backgrounds.

A deep learning methodology, CNN was used along with several augmentation and preprocessing methods. Different enhancement methods were tested in this study. The intention was to oversample some unbalanced classes and enlarge the dataset. Then, the more realistic positional and color augmentations, like flipping, rotation, shifting, and noise, were selected and implemented. The preparation of the dataset is followed by CNN model training. The suggested model is trained using 128 square-size RGB images divided into 8 disease categories and 4 severity classes using various regularization techniques. Its evaluation outperforms the VGG model, which is tailored for multiple outputs, in a comparison of the findings.

While Experiment 2's Ridge Regression gave the suggested model a superior accuracy score of 0.98 for disease and severity classification, the baseline work, adopted VGG has accuracy values for severity and disease categorization of 0.94 and 0.96, respectively. Additionally, the proposed model cuts the number of parameters by more than half, which helps in the reduction of computation. In general, the proposed model achieves superior performance on training and

testing datasets for the categorization of diseases and their severity while using fewer computational resources.

Future Work

Even though the proposed work has ended up solving the gaps that were indicated in the problem statement the quality of the images makes the classification more difficult. Since most of the training dataset was collected from the secondary data source there was a high variation in the quality of images. Also, it consists of some images that do not represent the figured pattern by the model. Due to this, the accuracy and loss evaluation graph shows some outliers. Therefore primary data preparation by a researcher is recommended for better results. Additionally, some of the diseases may show overlapping in a field therefore symptoms in a single leaf may show more than one disease symptom which makes the classification task more difficult. To reduce such kind of impact and occlusion of different leaf diseases in a single image applying different segmentation techniques to train the model is recommended for a more performed model result.

***

This research work fund was granted by Adama Science and Technology University. **ASTU/SM-R/636/22**

REFERENCES

(n.d.).

Blankson, D., Asare-Bediako, E., Frimpong, K. A., Ampofo, E., Taah, K. J., & Van der Puije, G. C. (2018). Incidence and severity of maize streak disease: The influence of tillage, fertilizer application and maize variety. African Journal of Agricultural Research, 13, 551-560.

EUBFE. (2015). Ethiopia Economic and Trade Report. Addis Ababa: European Business Forum in Ethiopia(EUBFE).

Abate, T., Shiferaw, B., Menkir, A., Wegary, D., Kebede, Y., Tesfaye, K., ... & Keno, T. (2015). Factors that Transformed Maize Productivity in Ethiopia. Food security, 7, 965–981.

Aboneh, T., Rorissa, A., Srinivasagan, R., & Gemechu, A. (2021). Computer Vision Framework for Wheat Disease Identification and Classification Using Jetson GPU Infrastructure. Technologies, 47.

Abuzaher, M., & Jamil, A. A. (2017). JPEG Based Compression Algorithm. International Journal of Engineering and Applied Sciences, 4.

Adams, I. P., Harju, V. A., Hodges, T., Hany, U., Skelton, A., Rai, S., ... & Boonham, N. (2014). First report of maize lethal necrosis disease in Rwanda,New Disease Report. New Disease Reports, 29. doi:10.5197/j.2044- 0588.2014.029.022

Ajit, A., Acharya, K., Samanta, A. . (2020). A review of convolutional neural networks. international conference on emerging trends in information technology and engineering (ic-ETITE) (pp. 1-5). IEEE.

Alehegn, E. (2019). Ethiopian maize diseases recognition and classification using support vector machine. Computational Vision and Robotics, 9.

Aravind, K. R., Raja, P., Mukesh, K. V., Aniirudh, R., Ashiwin, R., & Szczepanski, C. (2018). Disease classification in maize crop using bag of features and multiclass support vector machine. 2nd International Conference on Inventive Systems and Control (ICISC) (pp. 1191-1196). IEEE.

Bayissa Regassa, H. U. (2021). Transmission and Persistence of Maize Lethal Necrosis in Infested Soil and Infected Maize. Plant Pathology,. doi:10.1007/s10658-021-02401-w

Burkov, A. (2019). The Hundred-Page Machine Learning Book. Burkov, Andriy.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. IEEE conference on computer vision and pattern recognition (pp. 1251-1258). IEEE.

Crop - definition of crop by The Free Dictionary. (2022, March 29). Retrieved from The Free Dictionary: https://www.thefreedictionary.com/crop

CSA., C. S. (1990-2001.). Agricultural sample survey reports on area, production for major crops (private peasant holdings meher season). Addis Abbaba: The FDRE Statistical Bulletins.

Darwin, B., Dharmaraj, P., Prince, S., Popescu, D. E., & Hemanth, D. J. . (2021). Recognition of bloom/yield in crop images using deep learning models for smart agriculture. Agronomy, 646.

Das, D., Singh, M., Mohanty, S. S., & Chakravarty, S. (2020). Leaf Disease Detection using Support Vector Machine. International Conference on Communication and Signal Processing (ICCSP) (pp. 1036-1040). IEEE.

Deepika, Jaswal, Sowmya. V, K.P., & Sowman . (2014). Image Classification using Convolutional Neural Networks. International Journal of Advancements in Research & Technology, 3, 1661-1668.

Dhawan, Sachin. (2011). A review of image compression and comparison of its algorithms. International Journal of Electronics \& Communication Technology, IJECT, 2.

Dong, M., Mu, S., Shi, A., Mu, W., & Sun, W. (2020). Novel method for identifying wheat leaf disease images based on differential amplification convolutional neural network. International Journal of Agricultural and Biological Engineering, 13, 205-210.

Dowswell, C. R., Paliwal, R. L., & Cantrell, R. P. (2019). Maize in the third world. USA Inc. Colorado,: CRC Press.

Galanakis, C. M. (Ed.). (2018). Sustainable recovery and reutilization of cereal processing by-products. Elsevier.

Ganatra, N., & Patel, A. (2020). Performance analysis of fine-tuned convolutional neural network models for plant disease classification. International Journal of Control and Automation, 13, 293--305.

Grandini, M., Bagli, E., & Visani, G. . (2020). Metrics for multi-class classification: an overview. arXiv.

Haffangel, H. (1961). Agriculture in Ethiopia. . Rome.: FAO.

Hancock, J. T., & Khoshgoftaar, T. M. (2020). Survey on categorical data for neural networks. Journal of Big Data, 1.

Hossain, E., Hossain, M. F., & Rahaman, M. A. (2019). A color and texture based approach for the detection and classification of plant leaf disease using KNN classifier. International Conference on Electrical, Computer and Communication Engineering (ECCE) (pp. 1-6). IEEE.

Jeffers, D. (2004). Maize diseases: a guide for field identification (4th ed.). Mexico: D.F.: CIMMYT.

Keno, T., Azmach, G., Gissa, D. W., Regasa, M. W., Tadesse, B., Wolde, L., ... & Mahabaleswara, S. (2018). Major biotic maize production stresses in Ethiopia and their management through host resistance. African Journal of Agricultural research.

Khalifa, N. E., Loey, M., & Mirjalili, S. (2021). A comprehensive survey of recent trends in deep learning for digital images augmentation. Artificial Intelligence Review, 1-27.

Koike, S. T., Gaskell, M., Fouche, C., Smith, R., & Mitchell, J. . (2000). Plant disease management for organic crops.

Kumar, R. (2018). Research methodology: A step-by-step guide for beginners. Sage.

Liang, Q., Xiang, S., Hu, Y., Coppola, G., Zhang, D., & Sun, W. (2019). PD2SE-Net: Computer-assisted plant disease diagnosis and severity estimation network. Computers and electronics in agriculture, 157, 518--529.

Ma, N., Zhang, X., Zheng, H. T., & Sun, J. . (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European conference on computer vision (ECCV) , 116-131.

Macauley, H., & Ramadjita, T. (2015). Cereal crops: Rice, maize, millet, sorghum, wheat.

Mehta, Y. R. (2014). Wheat Diseases and Their Management. Switzerland: Springer.

Miceli, P. A., Blair, W. D., & Brown, M. M. (2018). Isolating Random and Bias Covariances in Tracks. International Confference on Information Fusion (FUSION). IEEE.

Mohamed, A. A. . (2017). Food security situation in Ethiopia: a review study. International journal of health economics and policy, 2(3), 86-96.

Mulatu, K., Bogale, G., Tolesa, B., Worku, M., Desalegne, Y., & Afeta, A. . (1993). Maize production trends and research in Ethiopia. The First National Maize Workshop of Ethiopia (pp. 4-12). Addis Ababa, Ethiopia: IAR/CIMMYT.

NIVIDIA. (2022, 7 10). Index. Retrieved from NIVIDIA: https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/

Olana, M. D., Rajesh Sharma, R., Sungheetha, A., & Chung, Y. K. . (2019). Applying deep learning approach for wheat rust disease detection using MosNet classification technique. ASTU (Adama Science and Technology).

Pallagani, V., Khandelwal, V., Chandra, B., Udutalapally, V., Das, D., & Mohanty, S. P. (2019). DCrop: A deep-learning based framework for accurate prediction of diseases of crops in smart agriculture. IEEE International Symposium on Smart Electronic Systems (iSES), 29-33.

Panchal, P., Raman, V. C., & Mantri, S. (2019). Plant diseases detection and classification using machine learning models. 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS). 4, pp. 1-6. IEEE.

Panigrahi, K. P., Das, H., Sahoo, A. K., & Moharana, S. C. (2020). Maize leaf disease detection and classification using machine learning algorithms. In Progress in Computing, Analytics and Networking (pp. 659-669). Singapore.: Springer.

Ramesh, S., Hebbar, R., Niveditha, M., Pooja, R., Shashank, N., & Vinod, P. V. (2018). Plant Disease Detection Using Machine Learning. International conference on design innovations for 3Cs compute communicate control (ICDI3C) (pp. 41-45). ResearchGate.

Randles, B. M., Pasquetto, I. V., Golshan, M. S., & Borgman, C. L. (2017). Using the Jupyter notebook as a tool for open science. IEEE Joint Conference on Digital Libraries (JCDL) (pp. 1-2). Ieee.

Rangarajan, A. K., Purushothaman, R., & Ramesh, A. . (2018). Tomato crop disease classificaation using pre-trained deep learning algorithm. Procedia computer science. Procedia computer science, 133, 1040-1047.

Raschka, S., Patterson, J., & Nolet, C. (2020). Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. Information, 11, 193.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Saleem, M. H., Potgieter, J., & Arif, K. M. (2019). Plant Disease Detection and Classification by Deep Learning. Plants.

Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the . IEEE conference on computer vision and pattern recognition (pp. 1-9). IEEE.

Sharma, P., Berwal, Y. P. S., & Ghai, W. (2020). Performance analysis of deep learning CNN models for disease detection in plants using image segmentation. Information Processing in Agriculture,, 566-574.

Sifre, L., & Mallat, S. (2013). Rotation, scaling and deformation invariant scattering for texture discrimination. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1233-1240). IEEE .

Sood, S., & Singh, H. (2020). An implementation and analysis of deep learning models for the detection of wheat rust disease. International Conference on Intelligent Sustainable Systems (ICISS) . (pp. 341-347). IEEE.

Syarief, M., & Setiawan, W. (2020). Convolutional neural network for maize leaf disease image classification. Telecommunication Computing Electronics and Control, 18, 1376-1381.

Tadele, E. (2021). Developing Ethiopian Paper Currency Recognition Model Using Convolutional Neural Network (CNN). ASTU edu, 11.

Tichkule, S, G. , & Dhanashri. (2015). Plant disease detection using image processing techniques. International Journal of Innovative Research in Science, Engineering and Technology, 4, 295–301.

Twumasi-Afriyie, S., Zelleke, H., Yihun, K., Asefa, B., & Tariku, S. (2002). Development and improvement of highland maize in Ethiopia. Enhancing the Contribution of Maize to Food Security in Ethiopia. EARO and CYMMIT, 31-38.

Types of Crops (6 Major Types) | Informed Farmers. (2022, March 29). Retrieved from informedfarmers.com: https://informedfarmers.com/types-of-crops/

Vaishnnave, M. P., Devi, K. S., Srinivasan, P., & Jothi, G. A. P. (2019). Detection and classification of groundnut leaf diseases using KNN classifier. International Conference on System, Computation, Automation and Networking (ICSCAN) (pp. 1-5). IEEE.

Vijayakumar, T., & Vinothkanna, R. (2020). Retrieval of complex images using visual saliency guided cognitive classification. J Innov Image Process (JIIP), 102-109.

Vijaypal Singh Dhaka, Sangeeta Vaibhav Meena, Geeta Rani, Deepak Sinwar, Kavita. (2021). A Survey of Deep Convolutional Neural Networks Applied for Prediction of Plant Leaf Diseases. Sensors.

Vyas, A., Yu, S., & Paik, J. (2018). Fundamentals of digital image processing. Springer.

Wang, G., Sun, Y., & Wang, J. (2017). Automatic Image-Based Plant Disease Severity Estimation Using Deep Learning. Computational Intelligence and Neuroscience.

Worku, M., Tuna, H., Nigussie, M., Deressa, A., Tanner, D., & Twumasi-Afriyie, S. (2002). Maize production trends and research in Ethiopia. Mandefro, N. Tanner, DG, Twumasi-Afriyie, S.(eds.), 10-14.

Worku, M., Twumasi Afriyie, S., Wolde, L., Tadesse, B., Demisie, G., Bogale, G., ... & Prasanna, B. M. (2012). Meeting the challenges of global climate change and food security through innovative maize research. Proceedings of the National Maize Workshop of Ethiopia. 3rd National Maize Workshop of Ethiopia. (pp. 193-201). A.A: CIMMYT.

Wu, H., Liu, Q., & Liu, X. (2019). A review on deep learning approaches to image classification and object segmentation. Computers, Materials & Continua , 575-597.

Xiang, S., Liang, Q., Sun, W., Zhang, D., & Wang, Y. (2021). L-CSMS: novel lightweight network for plant disease severity recognition. Journal of Plant Diseases and Protection, 557-569.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. Insights into imaging, 9, 611-629.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. Insights into imaging. Insights into imaging, 9, 611-629.

Ying, X. (2019). An overview of overfitting and its solutions. Journal of physics: Conference serie. IOP .

Zhang, X., Qiao, Y., Meng, F., Fan, C., & Zhang, M. (2018). Identification of maize leaf diseases using improved deep convolutional neural networks. IEEE Access, 30370-30377.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 6848-6856). IEEE.

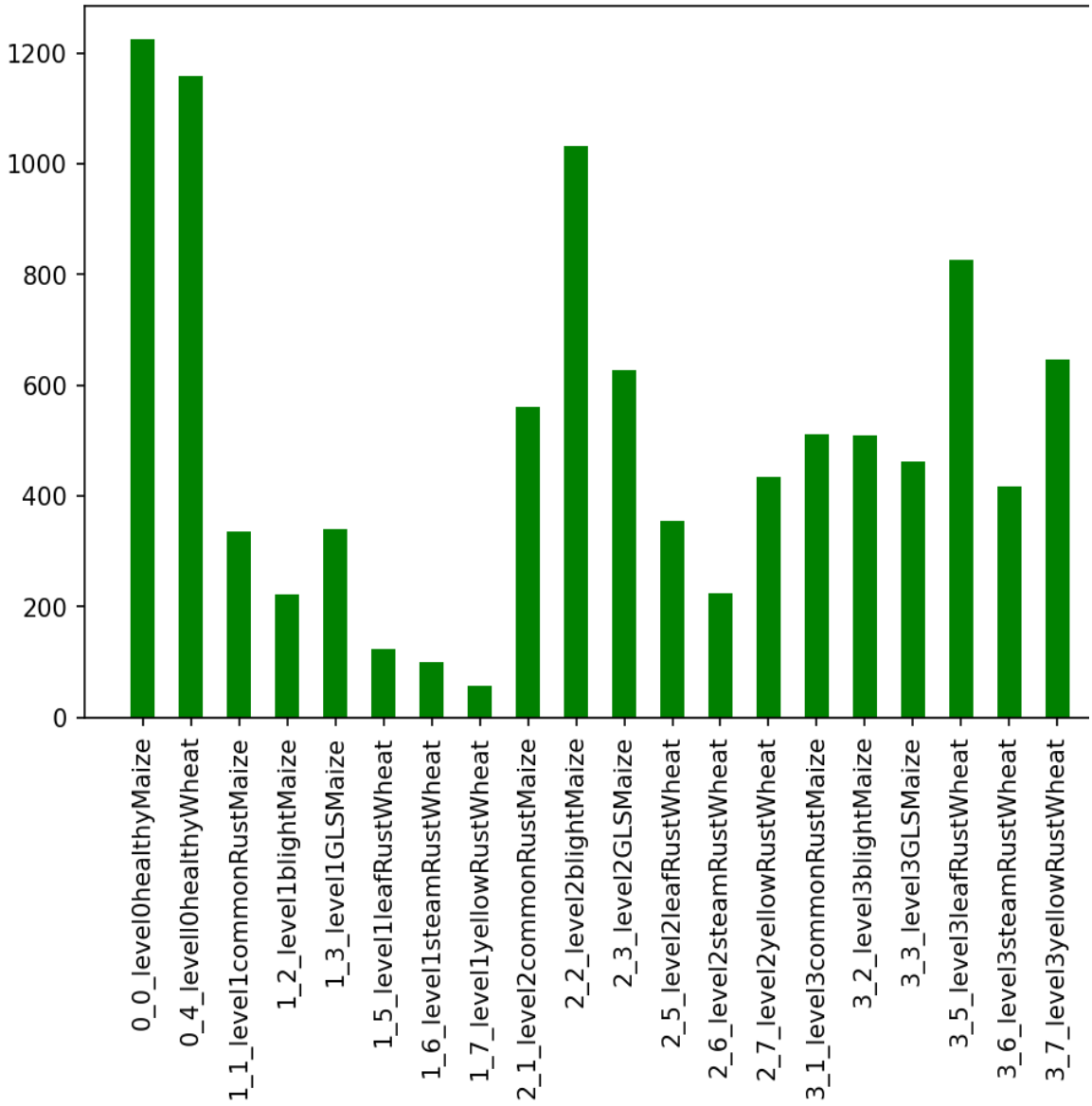
Zhao, J., Xu, C., Xu, J., Huang, L., Zhang, D., & Liang, D. (2018). Forecasting the wheat powdery mildew (Blumeria graminis f. Sp. tritici) using a remote sensing-based decision-tree classification at a provincial scale. Australasian Plant Pathology, 47, 53-61.

APPENDIX

Appendix A. **Data Organization**

No.	**Category**	**Total**	**Train 80%**	**Valid 10%**	**Test 10%**
1	0_0_level0HealthyMaize	1024	818	103	103
2	0_4_level0HealthyWheat	1059	847	106	106
3	1_1_level1commonRust	335	268	33	34
4	1_2_level1blightMaize	223	178	22	23
5	1_3_level1GLSMaize	341	272	34	35
6	1_5_level1leafRustWheat	224	179	22	23
7	1_6_level1SteamRustWheat	201	160	20	21
8	1_7_level1yellowRustWheat	158	126	16	16
9	2_1_level2CommonRustMaize	561	448	56	57
10	2_2_level2blightMaize	521	416	53	52
11	2_3_level2GLSMaize	628	502	63	63
12	2_5_level2leafRustWheat	355	284	35	36
13	2_6_level2steamRustWheat	225	180	22	23
14	2_7_level2yellowRustWheat	434	347	43	44
15	3_1_level3CommonRustMaize	502	401	50	51
16	3_2_level3blightMaize	495	396	49	50
17	3_3_level3GLSMaize	463	370	46	47
18	3_5_level3leafRustWheat	727	581	73	73
19	3_6_level3steamRustWheat	418	334	42	42
20	3_7_level3yellowRustWheat	647	517	65	65
	Total	***9,518***	***7,624***	***953***	***941***

Appendix B. **Collected Data Distribution Bar Graph**



Appendix C. **Data distribution**

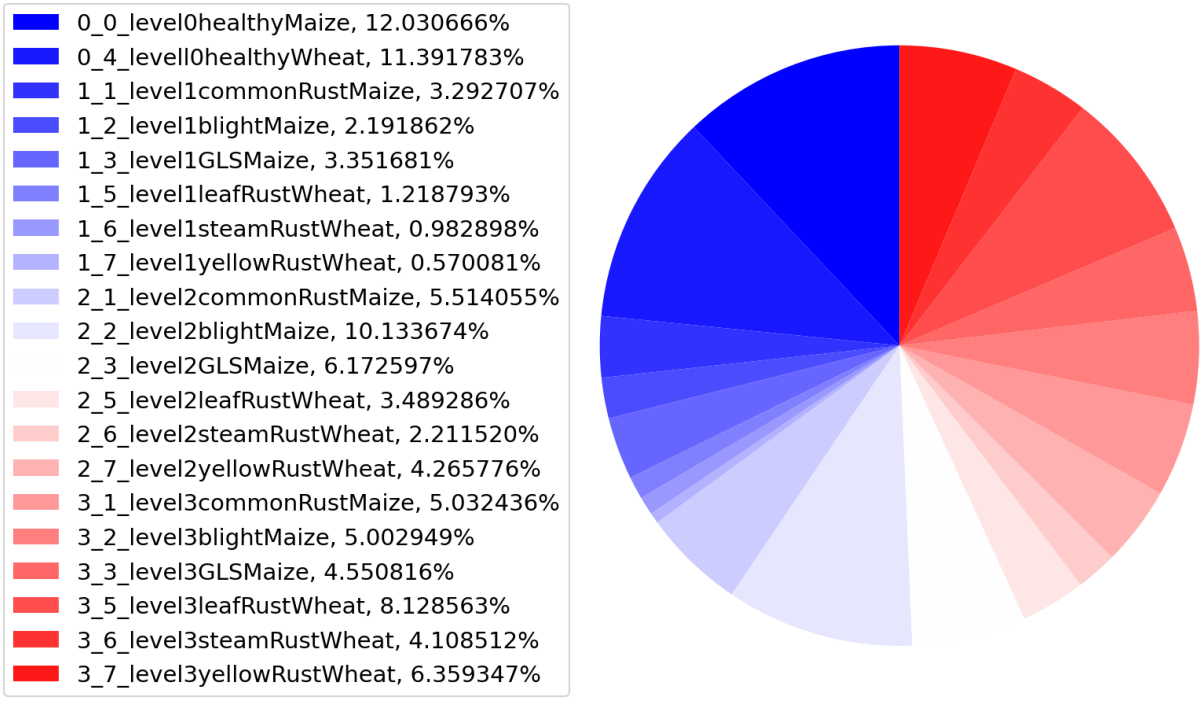


Figure 54 Collected Raw Data Distribution Chart

Appendix D. **Proposed Model Summery**

Table 11 Model Summery

Layer (type)	Output Shape	Param #	Connected to
disease_severity_inputs (InputL	[(None, 128, 128, 3)	0	
conv2d_70 (Conv2D)	(None, 128, 128, 64)	1792	disease_severity_inputs[0][0]
batch_normalization_96 (BatchNo	(None, 128, 128, 64)	256	conv2d_70[0][0]
conv2d_71 (Conv2D)	(None, 128, 128, 64)	36928	batch_normalization_96[0][0]
batch_normalization_97 (BatchNo	(None, 128, 128, 64)	256	conv2d_71[0][0]
max_pooling2d_9 (MaxPooling2D)	(None, 64, 64, 64)	0	batch_normalization_97[0][0]
conv2d_72 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d_9[0][0]
batch_normalization_98 (BatchNo	(None, 64, 64, 128)	512	conv2d_72[0][0]
conv2d_73 (Conv2D)	(None, 64, 64, 128)	147584	batch_normalization_98[0][0]
batch_normalization_99 (BatchNo	(None, 64, 64, 128)	512	conv2d_73[0][0]
max_pooling2d_10 (MaxPooling2D)	(None, 32, 32, 128)	0	batch_normalization_99[0][0]
conv2d_74 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_10[0][0]
batch_normalization_100 (BatchN	(None, 32, 32, 256)	1024	conv2d_74[0][0]

```
conv2d_75 (Conv2D)              (None, 32, 32, 256)  590080      batch_normalization_100[0][0]
batch_normalization_101 (BatchN (None, 32, 32, 256)  1024        conv2d_75[0][0]
max_pooling2d_11 (MaxPooling2D) (None, 16, 16, 256)  0           batch_normalization_101[0][0]
conv2d_76 (Conv2D)              (None, 16, 16, 256)  65792       max_pooling2d_11[0][0]
batch_normalization_102 (BatchN (None, 16, 16, 256)  1024        conv2d_76[0][0]
re_lu_48 (ReLU)                 (None, 16, 16, 256)  0           batch_normalization_102[0][0]
depthwise_conv2d_25 (DepthwiseC (None, 8, 8, 256)    2560        max_pooling2d_11[0][0]
depthwise_conv2d_24 (DepthwiseC (None, 8, 8, 256)    2560        re_lu_48[0][0]
batch_normalization_105 (BatchN (None, 8, 8, 256)    1024        depthwise_conv2d_25[0][0]
batch_normalization_103 (BatchN (None, 8, 8, 256)    1024        depthwise_conv2d_24[0][0]
conv2d_78 (Conv2D)              (None, 8, 8, 256)    65792       batch_normalization_105[0][0]
conv2d_77 (Conv2D)              (None, 8, 8, 256)    65792       batch_normalization_103[0][0]
batch_normalization_106 (BatchN (None, 8, 8, 256)    1024        conv2d_78[0][0]
batch_normalization_104 (BatchN (None, 8, 8, 256)    1024        conv2d_77[0][0]
re_lu_50 (ReLU)                 (None, 8, 8, 256)    0           batch_normalization_106[0][0]
re_lu_49 (ReLU)                 (None, 8, 8, 256)    0           batch_normalization_104[0][0]
tf_op_layer_concat_18 (TensorFl [(None, 8, 8, 512)]  0           re_lu_50[0][0]
                                                                 re_lu_49[0][0]
reshape_36 (Reshape)            (None, 8, 8, 64, 8)  0           tf_op_layer_concat_18[0][0]
permute_18 (Permute)            (None, 8, 8, 8, 64)  0           reshape_36[0][0]
reshape_37 (Reshape)            (None, 8, 8, 512)    0           permute_18[0][0]
tf_op_layer_split_12 (TensorFlo [(None, 8, 8, 256),  0           reshape_37[0][0]
conv2d_79 (Conv2D)              (None, 8, 8, 256)    65792       tf_op_layer_split_12[0][1]
batch_normalization_107 (BatchN (None, 8, 8, 256)    1024        conv2d_79[0][0]
re_lu_51 (ReLU)                 (None, 8, 8, 256)    0           batch_normalization_107[0][0]
depthwise_conv2d_26 (DepthwiseC (None, 8, 8, 256)    2560        re_lu_51[0][0]
batch_normalization_108 (BatchN (None, 8, 8, 256)    1024        depthwise_conv2d_26[0][0]
conv2d_80 (Conv2D)              (None, 8, 8, 256)    65792       batch_normalization_108[0][0]
batch_normalization_109 (BatchN (None, 8, 8, 256)    1024        conv2d_80[0][0]
re_lu_52 (ReLU)                 (None, 8, 8, 256)    0           batch_normalization_109[0][0]
tf_op_layer_concat_19 (TensorFl [(None, 8, 8, 512)]  0           tf_op_layer_split_12[0][0]
                                                                 re_lu_52[0][0]
reshape_38 (Reshape)            (None, 8, 8, 64, 8)  0           tf_op_layer_concat_19[0][0]
permute_19 (Permute)            (None, 8, 8, 8, 64)  0           reshape_38[0][0]
reshape_39 (Reshape)            (None, 8, 8, 512)    0           permute_19[0][0]
tf_op_layer_split_13 (TensorFlo [(None, 8, 8, 256),  0           reshape_39[0][0]
conv2d_81 (Conv2D)              (None, 8, 8, 256)    65792       tf_op_layer_split_13[0][1]
batch_normalization_110 (BatchN (None, 8, 8, 256)    1024        conv2d_81[0][0]
re_lu_53 (ReLU)                 (None, 8, 8, 256)    0           batch_normalization_110[0][0]
depthwise_conv2d_27 (DepthwiseC (None, 8, 8, 256)    2560        re_lu_53[0][0]
batch_normalization_111 (BatchN (None, 8, 8, 256)    1024        depthwise_conv2d_27[0][0]
conv2d_82 (Conv2D)              (None, 8, 8, 256)    65792       batch_normalization_111[0][0]
batch_normalization_112 (BatchN (None, 8, 8, 256)    1024        conv2d_82[0][0]
re_lu_54 (ReLU)                 (None, 8, 8, 256)    0           batch_normalization_112[0][0]
tf_op_layer_concat_20 (TensorFl [(None, 8, 8, 512)]  0           tf_op_layer_split_13[0][0]
                                                                 re_lu_54[0][0]
reshape_40 (Reshape)            (None, 8, 8, 64, 8)  0           tf_op_layer_concat_20[0][0]
permute_20 (Permute)            (None, 8, 8, 8, 64)  0           reshape_40[0][0]
```

```
reshape_41 (Reshape)            (None, 8, 8, 512)    0           permute_20[0][0]
conv2d_84 (Conv2D)              (None, 8, 8, 512)    262656      reshape_41[0][0]
batch_normalization_114 (BatchN (None, 8, 8, 512)    2048        conv2d_84[0][0]
re_lu_56 (ReLU)                 (None, 8, 8, 512)    0           batch_normalization_114[0][0]
depthwise_conv2d_29 (DepthwiseC (None, 4, 4, 512)    5120        reshape_41[0][0]
depthwise_conv2d_28 (DepthwiseC (None, 4, 4, 512)    5120        re_lu_56[0][0]
batch_normalization_117 (BatchN (None, 4, 4, 512)    2048        depthwise_conv2d_29[0][0]
batch_normalization_115 (BatchN (None, 4, 4, 512)    2048        depthwise_conv2d_28[0][0]
conv2d_86 (Conv2D)              (None, 4, 4, 512)    262656      batch_normalization_117[0][0]
conv2d_85 (Conv2D)              (None, 4, 4, 512)    262656      batch_normalization_115[0][0]
batch_normalization_118 (BatchN (None, 4, 4, 512)    2048        conv2d_86[0][0]
batch_normalization_116 (BatchN (None, 4, 4, 512)    2048        conv2d_85[0][0]
re_lu_58 (ReLU)                 (None, 4, 4, 512)    0           batch_normalization_118[0][0]
re_lu_57 (ReLU)                 (None, 4, 4, 512)    0           batch_normalization_116[0][0]
tf_op_layer_concat_21 (TensorFl [(None, 4, 4, 1024)] 0           re_lu_58[0][0]
                                                                 re_lu_57[0][0]
reshape_42 (Reshape)            (None, 4, 4, 128, 8) 0           tf_op_layer_concat_21[0][0]
permute_21 (Permute)            (None, 4, 4, 8, 128) 0           reshape_42[0][0]
reshape_43 (Reshape)            (None, 4, 4, 1024)   0           permute_21[0][0]
tf_op_layer_split_14 (TensorFlo [(None, 4, 4, 512),  0           reshape_43[0][0]
conv2d_87 (Conv2D)              (None, 4, 4, 512)    262656      tf_op_layer_split_14[0][1]
batch_normalization_119 (BatchN (None, 4, 4, 512)    2048        conv2d_87[0][0]
re_lu_59 (ReLU)                 (None, 4, 4, 512)    0           batch_normalization_119[0][0]
depthwise_conv2d_30 (DepthwiseC (None, 4, 4, 512)    5120        re_lu_59[0][0]
batch_normalization_120 (BatchN (None, 4, 4, 512)    2048        depthwise_conv2d_30[0][0]
conv2d_88 (Conv2D)              (None, 4, 4, 512)    262656      batch_normalization_120[0][0]
batch_normalization_121 (BatchN (None, 4, 4, 512)    2048        conv2d_88[0][0]
re_lu_60 (ReLU)                 (None, 4, 4, 512)    0           batch_normalization_121[0][0]
tf_op_layer_concat_22 (TensorFl [(None, 4, 4, 1024)] 0           tf_op_layer_split_14[0][0]
                                                                 re_lu_60[0][0]
reshape_44 (Reshape)            (None, 4, 4, 128, 8) 0           tf_op_layer_concat_22[0][0]
permute_22 (Permute)            (None, 4, 4, 8, 128) 0           reshape_44[0][0]
reshape_45 (Reshape)            (None, 4, 4, 1024)   0           permute_22[0][0]
tf_op_layer_split_15 (TensorFlo [(None, 4, 4, 512),  0           reshape_45[0][0]
conv2d_89 (Conv2D)              (None, 4, 4, 512)    262656      tf_op_layer_split_15[0][1]
batch_normalization_122 (BatchN (None, 4, 4, 512)    2048        conv2d_89[0][0]
re_lu_61 (ReLU)                 (None, 4, 4, 512)    0           batch_normalization_122[0][0]
depthwise_conv2d_31 (DepthwiseC (None, 4, 4, 512)    5120        re_lu_61[0][0]
batch_normalization_123 (BatchN (None, 4, 4, 512)    2048        depthwise_conv2d_31[0][0]
conv2d_90 (Conv2D)              (None, 4, 4, 512)    262656      batch_normalization_123[0][0]
batch_normalization_124 (BatchN (None, 4, 4, 512)    2048        conv2d_90[0][0]
re_lu_62 (ReLU)                 (None, 4, 4, 512)    0           batch_normalization_124[0][0]
tf_op_layer_concat_23 (TensorFl [(None, 4, 4, 1024)] 0           tf_op_layer_split_15[0][0]
                                                                 re_lu_62[0][0]
reshape_46 (Reshape)            (None, 4, 4, 128, 8) 0           tf_op_layer_concat_23[0][0]
permute_23 (Permute)            (None, 4, 4, 8, 128) 0           reshape_46[0][0]
reshape_47 (Reshape)            (None, 4, 4, 1024)   0           permute_23[0][0]
conv2d_91 (Conv2D)              (None, 4, 4, 1024)   1049600     reshape_47[0][0]
```

```
conv2d_83 (Conv2D)              (None, 8, 8, 1024)   525312      reshape_41[0][0]
batch_normalization_125 (BatchN (None, 4, 4, 1024)   4096        conv2d_91[0][0]
batch_normalization_113 (BatchN (None, 8, 8, 1024)   4096        conv2d_83[0][0]
re_lu_63 (ReLU)                 (None, 4, 4, 1024)   0           batch_normalization_125[0][0]
re_lu_55 (ReLU)                 (None, 8, 8, 1024)   0           batch_normalization_113[0][0]
average_pooling2d_7 (AveragePoo (None, 2, 2, 1024)   0           re_lu_63[0][0]
average_pooling2d_6 (AveragePoo (None, 4, 4, 1024)   0           re_lu_55[0][0]
flatten_7 (Flatten)             (None, 4096)         0           average_pooling2d_7[0][0]
flatten_6 (Flatten)             (None, 16384)        0           average_pooling2d_6[0][0]
dense_14 (Dense)                (None, 1024)         4195328     flatten_7[0][0]
dense_12 (Dense)                (None, 1024)         16778240    flatten_6[0][0]
batch_normalization_127 (BatchN (None, 1024)         4096        dense_14[0][0]
batch_normalization_126 (BatchN (None, 1024)         4096        dense_12[0][0]
dropout_7 (Dropout)             (None, 1024)         0           batch_normalization_127[0][0]
dropout_6 (Dropout)             (None, 1024)         0           batch_normalization_126[0][0]
dense_15 (Dense)                (None, 4096)         4198400     dropout_7[0][0]
dense_13 (Dense)                (None, 1024)         1049600     dropout_6[0][0]
severity_catagory (Dense)       (None, 4)            16388       dense_15[0][0]
disease_catagory (Dense)        (None, 8)            8200        dense_13[0][0]
==================================================================================================
Total params: 31,350,092
Trainable params: 31,323,212
Non-trainable params: 26,880
__________________________________________________________________________________________________
```

Appendix E. **Evaluation Demo Images**

0_0_level0healthyMaize260.jpg

0_4_level0healthyWheat25.jpg

1_1_level1commonRustMaizeAug203

1_2_level1blightMaizeAug837.jpg

3_5_level3leafRustWheat192.jpg

3_3_level3GLSMaize302.jpg

2_7_level2yellowRustWheat393.jpg

2_7_level2yellowRustWheat37.jpg

3_6_level3steamRustWheat8.jpg

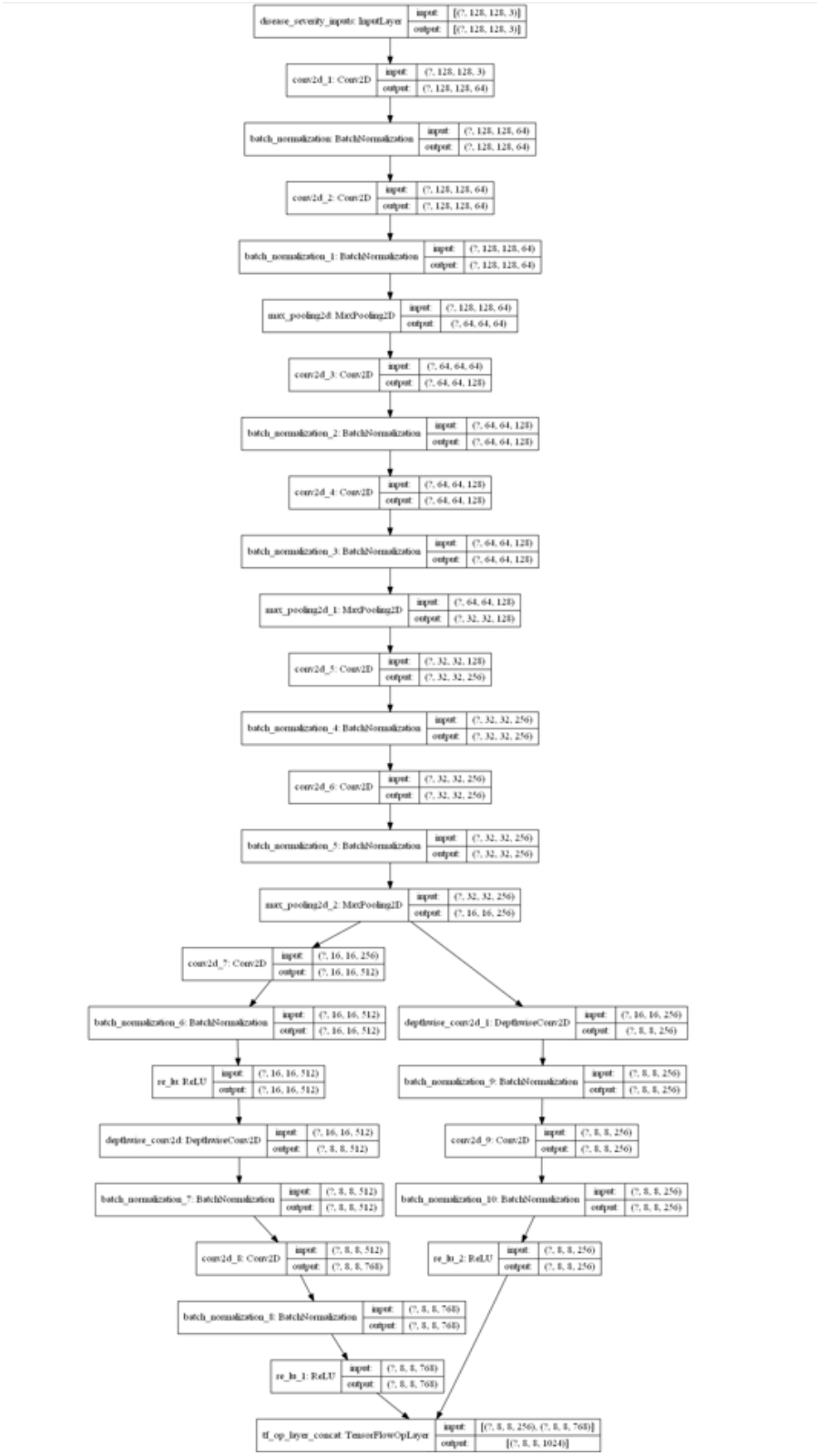
2_6_level2GLSMaizeaug26.jpg

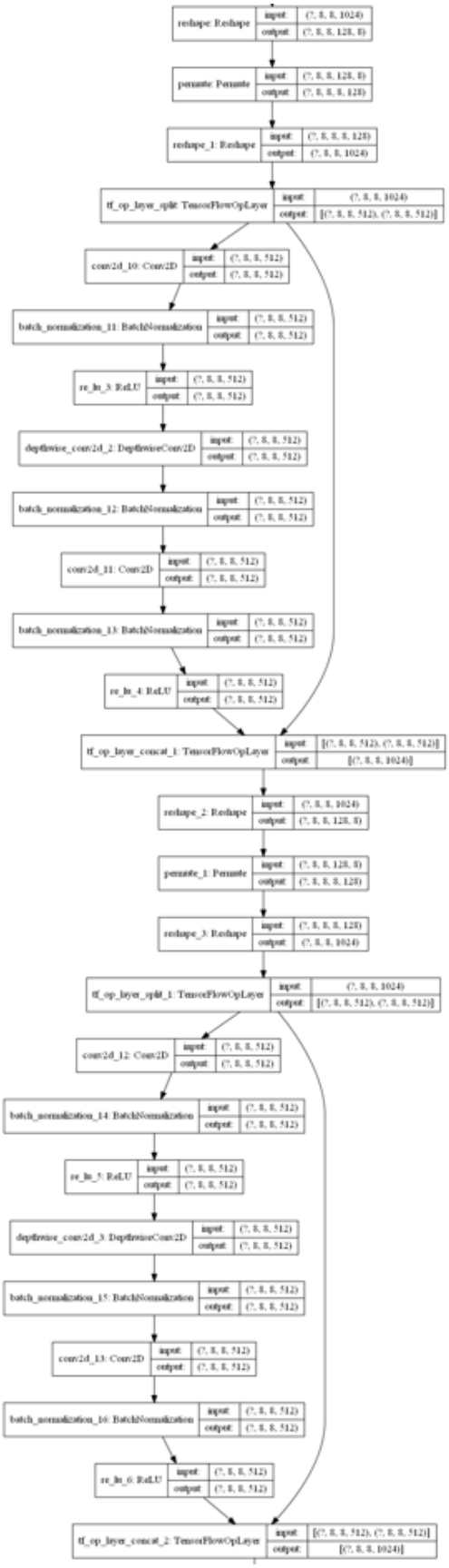
2_1_level2commonRustMaize127.jpg

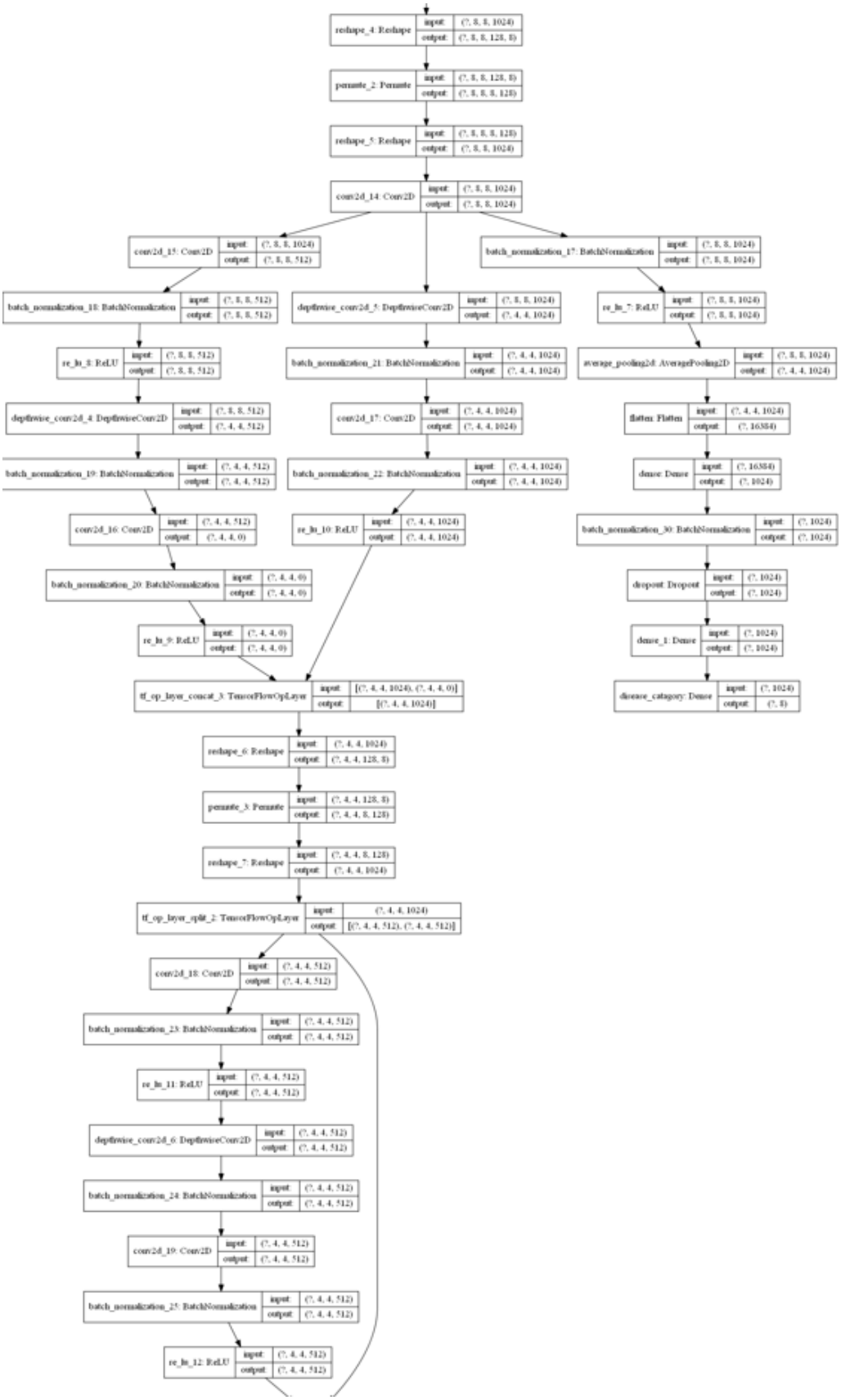
Appendix F. **Prediction Demo**

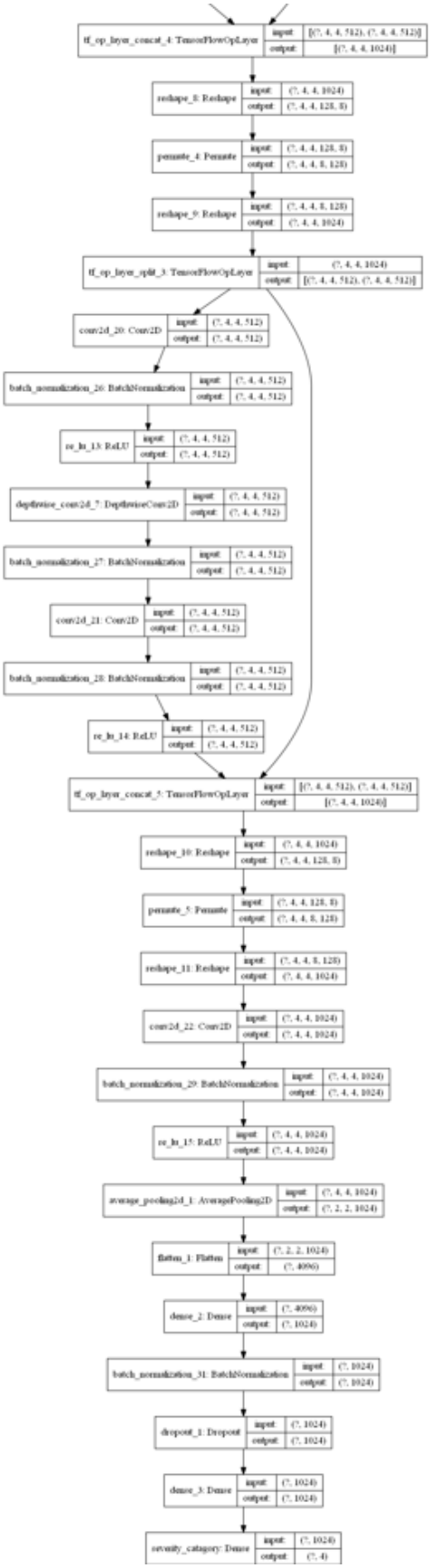
severity	disease	file	Prediction result
0	0	/home/cv-sig/Research Data/h/0_0_level0healthy...	disease pred: [0] severity pred [0]
0	4	/home/cv-sig/Research Data/h/0_4_level0health...	disease pred: [4] severity pred [0]
1	1	/home/cv-sig/Research Data/h/1_1_level1commo	disease pred: [1] severity pred [1]
1	2	/home/cv-sig/Research Data/h/1_2_level1blightM...	disease pred: [2] severity pred [1]
3	5	/home/cv-sig/Research Data/h/3_5_level3leafRus...	disease pred: [5] severity pred [3]
3	3	/home/cv-sig/Research Data/h/3_3_level3GLSMaiz...	disease pred: [3] severity pred [3]
2	7	/home/cv-sig/Research Data/h/2_7_level2yellowR...	disease pred: [7] severity pred [2]
2	7	/home/cv-sig/Research Data/h/2_7_level2yellowR...	disease pred: [7] severity pred [3]
3	6	/home/cv-sig/Research Data/h/3_6_level3steamRu.	disease pred: [6] severity pred [3]
2	6	/home/cv-sig/Research Data/h/2_6_level2GLSMaiz	disease pred: [6] severity pred [2]
2	1	/home/cv-sig/Research Data/h/2_1_level2commonl	disease pred: [3] severity pred [2]

Appendix G. **Proposed Model Full Schematic Diagram**




reshape: Reshape
input: (?, 8, 8, 1024)
output: (?, 8, 8, 128, 8)
permute: Permute
input: (?, 8, 8, 128, 8)
output: (?, 8, 8, 8, 128)
reshape_1: Reshape
input: (?, 8, 8, 8, 128)
output: (?, 8, 8, 1024)
tf_op_layer_split: TensorFlowOpLayer
input: (?, 8, 8, 1024)
output: [(?, 8, 8, 512), (?, 8, 8, 512)]
conv2d_10: Conv2D
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
batch_normalization_11: BatchNormalization
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
re_lu_3: ReLU
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
depthwise_conv2d_2: DepthwiseConv2D
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
batch_normalization_12: BatchNormalization
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
conv2d_11: Conv2D
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
batch_normalization_13: BatchNormalization
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
re_lu_4: ReLU
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
tf_op_layer_concat_1: TensorFlowOpLayer
input: [(?, 8, 8, 512), (?, 8, 8, 512)]
output: [(?, 8, 8, 1024)]
reshape_2: Reshape
input: (?, 8, 8, 1024)
output: (?, 8, 8, 128, 8)
permute_1: Permute
input: (?, 8, 8, 128, 8)
output: (?, 8, 8, 8, 128)
reshape_3: Reshape
input: (?, 8, 8, 8, 128)
output: (?, 8, 8, 1024)
tf_op_layer_split_1: TensorFlowOpLayer
input: (?, 8, 8, 1024)
output: [(?, 8, 8, 512), (?, 8, 8, 512)]
conv2d_12: Conv2D
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
batch_normalization_14: BatchNormalization
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
re_lu_5: ReLU
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
depthwise_conv2d_3: DepthwiseConv2D
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
batch_normalization_15: BatchNormalization
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
conv2d_13: Conv2D
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
batch_normalization_16: BatchNormalization
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
re_lu_6: ReLU
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
tf_op_layer_concat_2: TensorFlowOpLayer
input: [(?, 8, 8, 512), (?, 8, 8, 512)]
output: [(?, 8, 8, 1024)]


reshape_4: Reshape
input: (?, 8, 8, 1024)
output: (?, 8, 8, 128, 8)
permute_2: Permute
input: (?, 8, 8, 128, 8)
output: (?, 8, 8, 8, 128)
reshape_5: Reshape
input: (?, 8, 8, 8, 128)
output: (?, 8, 8, 1024)
conv2d_14: Conv2D
input: (?, 8, 8, 1024)
output: (?, 8, 8, 1024)
conv2d_15: Conv2D
input: (?, 8, 8, 1024)
output: (?, 8, 8, 512)
batch_normalization_17: BatchNormalization
input: (?, 8, 8, 1024)
output: (?, 8, 8, 1024)
batch_normalization_18: BatchNormalization
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
depthwise_conv2d_5: DepthwiseConv2D
input: (?, 8, 8, 1024)
output: (?, 4, 4, 1024)
re_lu_7: ReLU
input: (?, 8, 8, 1024)
output: (?, 8, 8, 1024)
re_lu_8: ReLU
input: (?, 8, 8, 512)
output: (?, 8, 8, 512)
batch_normalization_21: BatchNormalization
input: (?, 4, 4, 1024)
output: (?, 4, 4, 1024)
average_pooling2d: AveragePooling2D
input: (?, 8, 8, 1024)
output: (?, 4, 4, 1024)
depthwise_conv2d_4: DepthwiseConv2D
input: (?, 8, 8, 512)
output: (?, 4, 4, 512)
conv2d_17: Conv2D
input: (?, 4, 4, 1024)
output: (?, 4, 4, 1024)
flatten: Flatten
input: (?, 4, 4, 1024)
output: (?, 16384)
batch_normalization_19: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_22: BatchNormalization
input: (?, 4, 4, 1024)
output: (?, 4, 4, 1024)
dense: Dense
input: (?, 16384)
output: (?, 1024)
conv2d_16: Conv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 0)
re_lu_10: ReLU
input: (?, 4, 4, 1024)
output: (?, 4, 4, 1024)
batch_normalization_30: BatchNormalization
input: (?, 1024)
output: (?, 1024)
batch_normalization_20: BatchNormalization
input: (?, 4, 4, 0)
output: (?, 4, 4, 0)
dropout: Dropout
input: (?, 1024)
output: (?, 1024)
re_lu_9: ReLU
input: (?, 4, 4, 0)
output: (?, 4, 4, 0)
dense_1: Dense
input: (?, 1024)
output: (?, 1024)
tf_op_layer_concat_3: TensorFlowOpLayer
input: [(?, 4, 4, 1024), (?, 4, 4, 0)]
output: [(?, 4, 4, 1024)]
disease_catagory: Dense
input: (?, 1024)
output: (?, 8)
reshape_6: Reshape
input: (?, 4, 4, 1024)
output: (?, 4, 4, 128, 8)
permute_3: Permute
input: (?, 4, 4, 128, 8)
output: (?, 4, 4, 8, 128)
reshape_7: Reshape
input: (?, 4, 4, 8, 128)
output: (?, 4, 4, 1024)
tf_op_layer_split_2: TensorFlowOpLayer
input: (?, 4, 4, 1024)
output: [(?, 4, 4, 512), (?, 4, 4, 512)]
conv2d_18: Conv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_23: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
re_lu_11: ReLU
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
depthwise_conv2d_6: DepthwiseConv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_24: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
conv2d_19: Conv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_25: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
re_lu_12: ReLU
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)


tf_op_layer_concat_4: TensorFlowOpLayer
input: [(?, 4, 4, 512), (?, 4, 4, 512)]
output: [(?, 4, 4, 1024)]
reshape_8: Reshape
input: (?, 4, 4, 1024)
output: (?, 4, 4, 128, 8)
permute_4: Permute
input: (?, 4, 4, 128, 8)
output: (?, 4, 4, 8, 128)
reshape_9: Reshape
input: (?, 4, 4, 8, 128)
output: (?, 4, 4, 1024)
tf_op_layer_split_3: TensorFlowOpLayer
input: (?, 4, 4, 1024)
output: [(?, 4, 4, 512), (?, 4, 4, 512)]
conv2d_20: Conv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_26: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
re_lu_13: ReLU
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
depthwise_conv2d_7: DepthwiseConv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_27: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
conv2d_21: Conv2D
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
batch_normalization_28: BatchNormalization
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
re_lu_14: ReLU
input: (?, 4, 4, 512)
output: (?, 4, 4, 512)
tf_op_layer_concat_5: TensorFlowOpLayer
input: [(?, 4, 4, 512), (?, 4, 4, 512)]
output: [(?, 4, 4, 1024)]
reshape_10: Reshape
input: (?, 4, 4, 1024)
output: (?, 4, 4, 128, 8)
permute_5: Permute
input: (?, 4, 4, 128, 8)
output: (?, 4, 4, 8, 128)
reshape_11: Reshape
input: (?, 4, 4, 8, 128)
output: (?, 4, 4, 1024)
conv2d_22: Conv2D
input: (?, 4, 4, 1024)
output: (?, 4, 4, 1024)
batch_normalization_29: BatchNormalization
input: (?, 4, 4, 1024)
output: (?, 4, 4, 1024)
re_lu_15: ReLU
input: (?, 4, 4, 1024)
output: (?, 4, 4, 1024)
average_pooling2d_1: AveragePooling2D
input: (?, 4, 4, 1024)
output: (?, 2, 2, 1024)
flatten_1: Flatten
input: (?, 2, 2, 1024)
output: (?, 4096)
dense_2: Dense
input: (?, 4096)
output: (?, 1024)
batch_normalization_31: BatchNormalization
input: (?, 1024)
output: (?, 1024)
dropout_1: Dropout
input: (?, 1024)
output: (?, 1024)
dense_3: Dense
input: (?, 1024)
output: (?, 1024)
severity_category: Dense
input: (?, 1024)
output: (?, 4)