

2024-05

Image segmentation deep learning model used to Identify black Saigatoka And Fusarium wilt in banana early

Elinisa, Christina

NM-AIST

<https://dspace.nm-aist.ac.tz/handle/20.500.12479/2913>

Provided with love from The Nelson Mandela African Institution of Science and Technology

**IMAGE SEGMENTATION DEEP LEARNING MODEL USED TO
IDENTIFY BLACK SIGATOKA AND FUSARIUM WILT IN BANANA
EARLY**

Christian Arnold Elinisa

**A Dissertation Submitted in Partial Fulfilment of the Requirements for the Degree of
Master's in Information and Communication Science and Engineering of the Nelson
Mandela African Institution of Science and Technology**

Arusha, Tanzania

May, 2024

ABSTRACT

Bananas are among the most widely produced perennial fruit crops. Farmers largely produce bananas because they are important staple food and cash crops. However, bananas are highly affected by Fusarium Wilt and Black Sigatoka diseases. These diseases cause yield losses ranging from 30% to 100% of all the banana produce. Farmers face challenges in detecting and mitigating the effects of these two banana diseases because of a lack of knowledge of the diseases and the use of traditional eye observation method in detection. This study is inspired by the success of deep learning and computer vision in detecting a wide range of plant diseases. The study proposed the use of deep learning to automate the early detection of Fusarium Wilt and Black Sigatoka banana diseases. Mask R-CNN and U-Net image segmentation deep learning models were assessed for instance and semantic image segmentation. A dataset comprising 27 360 images of banana leaves and stalks that are healthy, Fusarium Wilt infected, and Black Sigatoka infected collected from the farm was used to train the models. An addition of 407 images of other things apart from the banana plant were downloaded from the internet and used to train the CNN model. From the experiments, the Mask R-CNN model achieved a mean Average Precision of 0.045 29 in segmenting the two banana diseases. The U-Net model achieved an Intersection over Union (IoU) of 93.23% and a Dice Coefficient of 96.45%. Similar results were obtained by Loyani *et al.* (2021) when they segmented a tomato plant paste called tuta absoluta using a U-Net model. Their model achieved a Dice Coefficient of 82.86% and an Intersection over Union of 78.60%. Additionally, the Fusarium Wilt and Black Sigatoka infected banana leaves and stalks were segmented using the Mask R-CNN and U-Net models. The CNN model yielded an accuracy of 91.71% in classifying the two banana diseases. Similar results were obtained by Sanga *et al.* (2020) when they deployed an Inceptionv3 model, which achieved an accuracy of 95.41%. The CNN model was deployed in a mobile application to be used by farmers to detect the two banana diseases early. The mobile application could detect banana diseases early and provide research-based mitigation recommendations that smallholder farmers and other agricultural stakeholders can use to avoid yield losses and financial losses.

DECLARATION

I, Christian Arnold Elinisa, declare to the Senate of the Nelson Mandela African Institution of Science and Technology that this dissertation is my original work and that it has neither been submitted nor concurrently submitted for a degree award in any other institution.

Christian Arnold Elinisa

Name of Candidate	Signature	Date
-------------------	-----------	------

The above declaration is confirmed by:

Dr. Neema Mduma

Name of Supervisor 1	Signature	Date
----------------------	-----------	------

Prof. Anthony Vodacek



17/04/2024

Name of Supervisor 2	Signature	Date
----------------------	-----------	------

Prof. Ciira wa Maina



24/04/2024

Name of Supervisor 3	Signature	Date
----------------------	-----------	------

COPYRIGHT

This dissertation is copyright material protected under the Berne Convention, the Copyright Act of 1999 and other international and national enactments, in that behalf, on intellectual property. It must not be reproduced by any means, in full or in part, except for short extracts in fair dealing; for researcher private study, critical scholarly review or discourse with an acknowledgement, without the written permission of the office of Deputy Vice-Chancellor for Academics, Research and Innovations on behalf of both the author and the Nelson Mandela African Institution of Science and Technology.

CERTIFICATION

The undersigned certify that they have read and hereby recommend for acceptance by the Nelson Mandela African Institution of Science and Technology, a dissertation titled **“Image Segmentation Deep Learning Model for Early Identification of Black Sigatoka and Fusarium Wilt in Banana”** submitted in partial fulfilment of the requirements for award of the degree of Master’s in Information and Communication Science and Engineering of the Nelson Mandela African Institution of Science and Technology.

Dr. Neema Mduma

Name of Supervisor 1	Signature	Date
----------------------	-----------	------

Prof. Anthony Vodacek



17/04/2024

Name of Supervisor 2	Signature	Date
----------------------	-----------	------

Prof. Ciira wa Maina



24/04/2024

Name of Supervisor 3	Signature	Date
----------------------	-----------	------

ACKNOWLEDGEMENTS

First and foremost, I would like to thank the Almighty God for his grace during the time of my Master's degree at the Nelson Mandela African Institution of Science and Technology in many aspects including love, guidance, wisdom, health, endurance, strength, and courage to accomplish this Master's program. I would like to extend my deepest gratitude to my family for their prayers, encouragement, and unwavering support.

This research was accomplished by the combined efforts of many people. I would like to express my gratitude to the following for their valuable contribution, assistance, and support during the research period.

I would like to express my deepest gratitude to my supervisor, Dr. Neema Mduma for her mentorship and for tirelessly providing support towards the accomplishment of this research; also, to my supervisors, Prof. Anthony Vodacek from Rochester Institute of Technology in the USA, and Prof. Ciira wa Maina from Dedan Kimathi University of Technology in Kenya, for their guidance and valuable feedback, especially during manuscript writing. Thank you for your great mentorship and supervision.

I extend my sincere gratitude for the financial support from the Data Science Africa (DSA); International Development Research Centre (IDRC) and the Swedish International Development Cooperation Agency (SIDA) through the Lacuna Fund in Agriculture, Artificial Intelligence for Development (AI4D) Africa and the African Center for Technology Studies (ACTS) that funded this research work.

Finally, I would like to thank my lectures, the NM-AIST CoCSE team, colleagues, classmates, and friends for their support. I extend my gratitude to Mr. Kennedy Jomanga, Mr. Loyani Loyani, Mr. Michael Nkotagu, and the farmers from IITA Arusha for their help in different stages of my research.

DEDICATION

I dedicate this work to my mother Ms. Joyce Joseph Ndesamburo.

TABLE OF CONTENTS

ABSTRACT.....	i
DECLARATION	ii
COPYRIGHT.....	iii
CERTIFICATION	iv
ACKNOWLEDGEMENTS.....	v
DEDICATION.....	vi
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF APPENDICES.....	xv
LIST OF ABBREVIATIONS AND SYMBOLS	xvi
CHAPTER ONE.....	1
INTRODUCTION	1
1.1 Background of the Problem	1
1.2 Statement of the Problem.....	3
1.3 Rationale of the Study.....	3
1.4 Objectives	4
1.4.1 General Objective	4
1.4.2 Specific Objectives	4
1.5 Research Questions.....	4
1.6 Significance of the Study	5
1.7 Delineation of the Study	5
CHAPTER TWO	7
LITERATURE REVIEW	7
2.1 Bananas.....	7
2.2 Black Sigatoka	7

2.3	Fusarium Wilt	9
2.4	Deep Learning Models and Feature Extraction	10
2.4.1	Convolutional Neural Network Feature Extraction	10
2.4.2	Mask Region-Based Convolutional Neural Network Model for Instance Segmentation.....	15
2.4.3	The U-Net Model for Semantic Segmentation	26
2.5	Theoretical Literature Review	31
2.6	Empirical Literature Review.....	32
2.6.1	Feature Extraction.....	32
2.6.2	Using Deep Learning and Machine Learning to Detect Banana Diseases	33
2.6.3	Mobile Applications that deploy Deep Learning models for Plant Disease Detection	35
2.7	Research Gap	35
CHAPTER THREE		37
MATERIALS AND METHODS.....		37
3.1	Study Area and Scope of the Research	37
3.2	Data Collection	37
3.3	Data Preprocessing.....	39
3.3.1	Data Cleaning and Cropping.....	39
3.3.2	Data Renaming.....	40
3.3.3	Data Annotation	41
3.4	Research Framework	42
3.5	Classification with Convolutional Neural Network Model	43
3.5.1	Convolutional Neural Network Model Hyper-Parameters Tuning.....	43
3.5.2	The CNN Model Classes and Data Grouping During Training.....	43
3.6	Transfer Learning.....	46

3.7	Mask Region-Based Convolutional Neural Network Model Hyper-Parameter Tuning	46
3.8	The U-Net Model	48
3.8.1	The U-Net Model Hyper-Parameter Tuning	48
3.8.2	The U-Net Model Classes and Data Grouping During Training	48
3.9	Experiment Setting	51
3.10	Evaluation	52
3.10.1	Accuracy	52
3.10.2	Recall	52
3.10.3	Precision	52
3.10.4	The F-Measure	53
3.10.5	Mean Average Precision (mAP)	53
3.10.6	Intersection Over Union	54
3.10.7	Dice Coefficient	55
3.10.8	Loss Function	55
3.11	Model Deployment	56
3.11.1	Requirements Elicitation and Analysis	57
3.11.2	System Design	60
3.11.3	System Development Methodology	63
3.11.4	Technologies Used	63
3.12	Validation of the Performance of the Developed Mobile Application	63
	CHAPTER FOUR	64
	RESULTS AND DISCUSSION	64
4.1	Feature Extraction Results	64
4.2	Model Development Results	66
4.2.1	Convolutional Neural Network Model Results	66

4.2.2	Mask Region-Based Convolutional Neural Network Model Results	68
4.2.3	The U-Net Model Results	68
4.3	Model Deployment Results.....	75
4.4	Validation of the Performance of the Developed Mobile Application Results	79
4.5	Discussion	81
CHAPTER FIVE		84
CONCLUSION AND RECOMMENDATIONS		84
5.1	Conclusion	84
5.2	Recommendations.....	84
REFERENCES		86
APPENDICES		95
RESEARCH OUTPUTS.....		135

LIST OF TABLES

Table 1:	Total number of data collected	39
Table 2:	A summary of removing duplicates.....	39
Table 3:	The CNN model training hyperparameters.....	43
Table 4:	Data distribution for the CNN model	45
Table 5:	Mask R-CNN model training hyperparameters	47
Table 6:	Data distribution for the U-NET model.....	49
Table 7:	The mobile application’s functional requirements and their description	58
Table 8:	Non-functional requirements for the mobile application and their description.....	60
Table 9:	The CNN model performance for detecting banana diseases.....	67
Table 10:	CNN model training time	68
Table 11:	Model loss results	70
Table 12:	Model evaluation metric results.....	72
Table 13:	Model training time	74
Table 14:	Results of the responses given by farmers to the mobile application validation questionnaire.....	80

LIST OF FIGURES

Figure 1:	Different stages in which Black Sigatoka develop and affect the banana leaves...8
Figure 2:	Damage caused by Fusarium Wilt on banana plants (a) How Fusarium Wilt affects the banana stack (b) How Fusarium Wilt affects banana leaves9
Figure 3:	The CNN architecture illustration 10
Figure 4:	Example of different filters applied to the same input image 12
Figure 5:	An example of the convolutional filter being used to transform an input feature map into an output feature map 13
Figure 6:	An example of padding the input matrix with zeros around the border so as to enhance the output feature map size 14
Figure 7:	An illustration of the application ReLU operation 15
Figure 8:	Max Pooling illustration (Moroney, 2021)..... 15
Figure 9:	Assessed Mask R-CNN model architecture 16
Figure 10:	Mask R-CNN ResNet backbone (Zhang, 2021)..... 17
Figure 11:	Region proposal network (Zhang, 2021)..... 17
Figure 12:	RoI Align (Zhang, 2021) 18
Figure 13:	The Object detection head (Zhang, 2021) 18
Figure 14:	The Mask generation head (Zhang, 2021)..... 19
Figure 15:	Extracting a feature map from an input image using VGG16..... 20
Figure 16:	The RoI target with coordinate size 20
Figure 17:	The RoI on the feature map (Erdem, 2020)..... 21
Figure 18:	Pooling layer (Erdem, 2020) 21
Figure 19:	Quantization when mapping and pooling (Erdem, 2020) 22
Figure 20:	The RoI pooling quantization losses (shown by the light and dark blue colors) and data gain (represented by the green color) (Erdem, 2020) 22
Figure 21:	The RoI box size (Erdem, 2020) 23
Figure 22:	The RoI divided into equal boxes (Erdem, 2020) 23
Figure 23:	Sampling points distribution (Erdem, 2020) 23

Figure 24: Bilinear interpolation for the (a) first point, (b) second point, (c) third point, and (d) fourth point (Erdem, 2020)	24
Figure 25: Max Pooling on first box (Erdem, 2020)	25
Figure 26: The RoIAlign pooling output (Erdem, 2020).....	25
Figure 27: The RoIAlign full size (Erdem, 2020).....	26
Figure 28: The U-Net architecture	27
Figure 29: A convolutional operation yielding a many-to-one relationship.....	28
Figure 30: A transpose convolution yielding a one-to-many relationship.....	28
Figure 31: A 4x16 convolution matrix.....	28
Figure 32: How a convolution matrix is formed.....	29
Figure 33: Converting our input matrix from 4x4 to a column vector 16x1	29
Figure 34: A convolution operation	30
Figure 35: A transposed convolution operation.....	31
Figure 36: Technology acceptance model (Lai, 2017)	32
Figure 37: Data collection in the field	38
Figure 38: Examples of images from the dataset.....	38
Figure 39: How VisiPics was used to detect and delete duplicates	40
Figure 40: How Bulk Rename Utility was used to rename images in the dataset.....	40
Figure 41: How LabelMe was used to manually annotate images (a) Annotation of a healthy banana leaf image; (b) Annotation of a banana leaf image affected by Black Sigatoka disease; and (c) Annotation of a banana stalk image affected by Fusarium Wilt disease	41
Figure 42: The Image annotation outputs (a) Original image (b) Drawing a polygon around the area on a leaf image affected by Black Sigatoka (c) Saving the annotation in JSON format (d) Visualizations of the labels (e) Extraction of the mask in PNG format	42
Figure 43: The research framework.....	42
Figure 44: The TensorFlow Lite suite (Moroney, 2021)	57

Figure 45: Use case diagram for the banana diseases detection mobile application	61
Figure 46: Activity diagram for the banana diseases detection mobile application	62
Figure 47: Sequence diagram for the capture/upload image use case	62
Figure 48: Example of backbone feature maps at the (a) input layer (b) res2c_out activation layer, and (c) res3c_out activation layer	64
Figure 49: Examples of RPN anchors (a) Regions of Interest (ROIs), (b) Negative anchors, and (c) Positive anchors	65
Figure 50: Original image, ground truth from the mask and the mask overlaid on the original image	65
Figure 51: Data augmentation applied equally on the image and its mask	66
Figure 52: Performance for the CNN model.....	67
Figure 53: Examples of how the Mask R-CNN model predicts segmentations (a) Image segmentation of a leaf affected by Black Sigatoka disease: and (b) Image segmentation of a leaf affected by Fusarium Wilt disease.....	68
Figure 54: Loss over epoch graph for U-Net group 8 model.....	71
Figure 55: Intersection over Union over epoch graph for U-Net group 8 model	73
Figure 56: Dice coefficient over epoch graph for U-Net group 8 model.....	73
Figure 57: Segmentation predictions from the U-Net model.....	75
Figure 58: Banana disease detection mobile application splash screen and detect page in English and Kiswahili	76
Figure 59: Banana disease detection mobile application detect page with detection results for Fusarium Wilt and mitigation recommendation page for the detected disease	77
Figure 60: Banana disease detection mobile application detect page with detection results for Black Sigatoka and mitigation recommendation page for the detected disease ...	78
Figure 61: Banana disease detection mobile application detect page with results for a healthy banana leaf and an image that is not of a banana leaf or stalk.....	78
Figure 62: Banana disease detection mobile application about banana page and about diseases page.....	79

LIST OF APPENDICES

Appendix 1:	Questions that Were Used to Come Up with the Functional and Non-Functional Requirements	95
Appendix 2:	Mobile Application Validation Questionnaire	96
Appendix 3:	Mask R-CNN Model Source Code	98
Appendix 4:	The U-Net Model Source Code	108
Appendix 5:	The CNN Model Source Code	113
Appendix 6:	Model Deployment Flutter Source Code	117
Appendix 7:	Poster Presentation	136

LIST OF ABBREVIATIONS AND SYMBOLS

AI	Artificial Intelligence
AI4D	Artificial Intelligent for development
API	Application Programming Interface
BLSD	Black Leaf Steak Disease
CNN	Convolutional Neural Network
COCO	Common Objects in Context
DL	Deep learning
DSA	Data Science Africa
FAO	Food and Agriculture Organization
FAOSTAT	Food Agricultural Organization Statistics
Faster R-CNN	Faster Region-based Convolutional Neural Network
GB	Giga Bytes
GDP	Gross Domestic Product
GPU	Graphical Processing Unit
HOG	Histogram of Oriented Gradients
IDE	Integrated Development Environment
IDRC	International Development Research Centre
IoU	Intersection Over Union
ISBI	International Symposium on Biomedical
KNN	K-nearest neighbor
LBP	Local Binary Pattern
mAP	Mean Average Precision
Mask R-CNN	Mask Region-based Convolutional Neural Network
ML	Machine Learning
NM-AIST	Nelson Mandel African Institution of Science and Technology

ODK	Open Data Kit
ReLU	Rectified Linear Unit
ResNet	Residual Network
RGB	Red, Green, and Blue
RoI	Region of Interest
RoIAlign	Region of Interest Alignment
RPN	Region Proposal Network
SDG	Sustainable Development Goal
SVM	Support Vector Machine
TP	True Positive
URL	Uniform Resource Locator
UUV	Unmanned Underwater Vehicles
VGG	Visual Geometry Group
XP	eXtreme Programming

CHAPTER ONE

INTRODUCTION

1.1 Background of the Problem

The United Nations' 2nd Sustainable Development Goal (SDGs) is to “End hunger, achieve food security and improved nutrition, and promote sustainable agriculture” (United Nations, 2021). This goal cannot be achieved without good approaches such as the use of artificial intelligence in crop disease management and proper structured resource utilization. According to reports (Che'Ya *et al.*, 2022), farmers are capable of automatically identifying and detecting infections and diseases early on, which helps to lessen their effects, enhance treatment results, and stop infections from recurring in agriculture.

According to the 2020 economic survey report, agriculture generated 26.9% of Tanzania's Gross Domestic Product (GDP), making it a significant economic sector (National Bureau of Statistics *et al.*, 2021). However, crop diseases have been the greatest challenge affecting major food security crops, including bananas. Nearly 70 million farmers grow bananas in the humid and sub-humid tropics of Africa, making it one of the main staple foods and cash crops that are mainly grown by small-scale farmers (FAO, 2021). Regardless of its importance in household food security and subsistence, this crop is highly attacked by diseases, particularly Fusarium Wilt and Black Sigatoka (Sanga *et al.*, 2020; Ramadhani, 2017). It is reported that yield losses due to Fusarium Wilt and Black Sigatoka diseases in bananas range from 30% to 100% in susceptible cultivars and highly susceptible respectively (Bubici *et al.*, 2019; Vézina & Van-den-Bergh, 2020; FAO, 2017).

Black Sigatoka sometimes called Black Leaf Streak Disease (BLSD) is a leaf spot disease caused by heterothallic and airborne fungus *Pseudocercospora fijiensis*. The fungus that causes this disease was discovered in a Fiji island valley called Sigatoka, where it gets its name (Vézina & Van-den-Bergh, 2020). Nevertheless, it is believed to have been common in the Asia-Pacific region much earlier. The disease has been found in Taiwan, Philippines, Indonesia, China, Vietnam, Malaysia, Singapore, and Thailand but in these countries, the fungus is not found in all locations (Vézina & Van-den-Bergh, 2020). The Black Sigatoka disease was initially confirmed to be in Gabon in Africa in 1980. The countries in West Africa that have the diseases are Cameroon, Togo, Benin, Nigeria, Ghana, the Democratic Republic of Congo, and Côte d'Ivoire. In 1987, Black Sigatoka was confirmed in East Africa on Pemba

Island in Tanzania. It spread to neighboring countries like Uganda, Kenya, Burundi, Madagascar, Rwanda, Malawi, Comoros, Mayotte, and in East Africa. According to reports, Black Sigatoka is most common in mid-altitude regions below 1350 meters above sea level. Until recently, it was not seen above this altitude (Johanson *et al.*, 2000). However, the fungus has gradually adapted to higher and cooler altitudes (Erima *et al.*, 2017).

Fusarium Wilt of bananas also called Panama disease is a destructive banana disease caused by a soil-borne fungus known as *Fusarium oxysporum* f.sp. *cubense* race 1 (Foc). In Australia's banana plantations, fusarium wilt disease was first identified in 1876 (Altendorf, 2019). It was then confirmed in Panama in 1890 where it was an epidemic (Daly & Walduck, 2006). Fusarium wilt, the first strain of the disease highly affected the susceptible Gros Michel banana which dominated the banana trade globally in Central America. In 1950, the disease caused the Cavendish type of banana to replace the Gros Michel banana type because Gros Michel was highly affected by Fusarium wilt disease (Vézina, 2022). There are four strains or races of Foc which are Race 1, Race 2, and Race 4 which infects bananas, and Race 3 which is a pathogen for *Heliconia* spp instead of banana (Ploetz *et al.*, 2015). This study focuses on Fusarium Wilt Foc Race 1 which is the first strain of the disease.

In Tanzania, farmers are facing several challenges, including climate change, a lack of agricultural tools, diseases, pest attacks, and the growth of weeds that cause damage to plants and decrease yields. These challenges affect the prosperity of farmers as well as the nation's economy. It is therefore important to find and implement the appropriate and effective solutions to the challenges to improve productivity. Artificial intelligence (AI) has propelled agricultural solutions that enable early pest and plant disease detection, health assessment, crop monitoring, early warning systems, and evaluation of tree canopy cover, etc. Several studies have been done that assessed deep learning models for detecting crop diseases in plants (Sanga *et al.*, 2020; Ramcharan *et al.*, 2017; Mkonyi *et al.*, 2020; Loyani, 2021). However, to the best of the author's knowledge, the segmentation of Black Sigatoka and Fusarium Wilt banana diseases has not been addressed.

Early disease detection using automatic plant disease detection techniques is beneficial. The automatic detection of plant diseases by capturing symptoms as features on plant leaf images is easier, less time-consuming, and less prone to errors when compared to disease detection through simple naked-eye observation done by farmers and experts. Deep learning can be used to automate the detection of plant diseases. By identifying the colour difference in the diseased

area, deep-learning image processing identifies the region of the leaf image that is affected by the disease. The region of interest (i.e., the area in the image of the banana plant affected by the disease) can be identified using a mask provided by image segmentation. Different properties of a picture, such as colour, boundaries, shapes, and texture, are used to segment the region of interest (Singh, 2017). This study developed an early detection mobile application for banana diseases deploying the CNN deep learning model. The mobile application will enable small-holder farmers to quickly detect banana plant diseases to intervene early. Furthermore, the study generated a banana image dataset that was shared on machine learning open-access repositories to facilitate research and teaching at different initiatives in Africa and globally.

1.2 Statement of the Problem

Despite government efforts to set aside funds each year for the growth and development of the agricultural sector, farmers continue to face issues that result in plant damage and decreasing plant yields. Diseases like Fusarium Wilt and Black Sigatoka affect the banana plant and lead to a decrease in plant yields. Most local banana farmers use simple naked-eye observation done by the farmers themselves and the agricultural experts to detect the presence of diseases on the plants, which is cumbersome, costly, prone to errors, and time-consuming. The current state of plant disease diagnosis is transitioning from disease identification using visible symptoms by the eyes to the use of automatic, data-driven solutions applying deep learning and computer vision techniques, which are easier and cheaper.

Deep learning has been applied in several studies to identify tomato diseases (Mkonyi *et al.*, 2020; Shijie *et al.*, 2017; Loyani, 2021); cassava diseases (Nabenda *et al.*, 2020; Ramcharan *et al.*, 2017); and banana diseases (Sanga *et al.*, 2020; Owomugisha *et al.*, 2014). The goal of this research was to create a deep-learning image segmentation model for the early identification of banana diseases. Deployment of the model was done in a mobile application to enhance its usability by the farmers. The developed mobile application also provided recommendations for curing banana diseases.

1.3 Rationale of the Study

The agriculture sector is essential to a nation's economy because it produces food, one of humanity's most fundamental requirements, goods for export to earn foreign currency, and raw materials for manufacturing. Bananas among the staple food and cash crops are highly

produced in many countries, including Tanzania. This crop has great potential to mitigate the presence of food insecurity and alleviate poverty and hunger, especially in developing countries. In Tanzania, a total of 20 735 banana-growing smallholder farmers were reported to have banana disease occurrences in 2011 (Sanga, 2020). The damage being caused by the diseases on banana plants continuously, negatively affects farmers by causing loss of yields and the economy of the country through loss of income. Farmers now need to be able to identify diseases early in the plant and be knowledgeable about the best ways to control the diseases due to the rise in banana yield loss that they are experiencing. This study, therefore, developed an early-detection mobile application for banana diseases. The study trained deep learning models from a dataset of images of banana leaves and stalks using image segmentation and classification methods. A CNN deep learning model was deployed in a mobile application and could advise farmers and extension officers on the most effective ways to reduce the effects of the diseases.

1.4 Objectives

1.4.1 General Objective

To develop an image segmentation deep learning model for early detection of banana diseases.

1.4.2 Specific Objectives

The study aimed to achieve the following specific objectives:

- (i) To identify the requirements for developing the image segmentation deep learning model
- (ii) To develop an image segmentation deep learning model for detecting banana diseases.
- (iii) To deploy the developed model in a mobile-based application.
- (iv) To validate the performance of the developed mobile-based application.

1.5 Research Questions

The study intended to answer the following questions:

- (i) What are the requirements for developing the image segmentation deep learning model?

- (ii) How can an image segmentation deep learning model be developed to detect banana diseases?
- (iii) How can we deploy a deep learning model in a mobile-based application?
- (iv) How can the performance of the developed mobile-based application be validated?

1.6 Significance of the Study

The objective of this study was to solve the challenges faced by farmers and agricultural experts when they try to manually detect diseases in farms using naked-eye observation by developing a mobile-based application for detecting banana diseases. This mobile application deployed a classification deep learning model to accurately identify banana diseases for farmers.

The developed mobile-based application accurately detects the presence of banana diseases early and provides information on recommended steps to take and cures to use for the detected diseases on the plant, allowing the plant to be cured before diseases have affected the entire plant and rendering mitigation efforts ineffective. This will help improve the banana yields for farmers by saving the infected plants from diseases early and causing them to produce banana fruits.

The solution developed by this study will play a role in achieving the United Nations' 2nd SDG, which is to “End hunger, achieve food security and improved nutrition, and promote sustainable agriculture” (United Nations, 2021). By early recommending treatments to banana plants afflicted by diseases, the created application will also contribute to achieving food security by enabling smallholder farmers to save the plants and produce bananas for consumption, sale, and export.

1.7 Delineation of the Study

This study focused on developing a mobile application for identifying Black Sigatoka and Fusarium Wilt fungal banana diseases because they cause high yield losses. The dataset used to develop the mobile application comprised of coloured images, that is, Red-Green-Blue (RGB) colour format, collected from the farms and other images downloaded from the internet. The experiment results show that the developed mobile application could accurately identify the two diseases, healthy banana leaves, and other images not of the banana plant. Further research could improve the performance of the mobile application and increase its ability to

identify more plant diseases and provide mitigation recommendations for the identified diseases.

However, as part of the limitation this study only focused on identifying the two banana diseases, healthy banana leaves, and images not of the banana leaves or stalks without including dry banana leaves and other banana diseases. This factor may affect the accuracy of the prediction done by the model.

CHAPTER TWO

LITERATURE REVIEW

2.1 Bananas

Bananas and plantains regarded as bananas are among the most widely produced perennial fruit crops (Brown *et al.*, 2017). Based on the report by the Food and Agricultural Organization Statistics (FAOSTAT), in 2021 the annual global production of bananas was 124 million tons, and 54.4% of these tones were produced in Asia, with India leading by producing 33 million tons and Africa producing 22 million tons which is equivalent to 22.7% of the global production (FAOSTAT, 2021). Bananas are among the most traded fruits globally (Voora *et al.*, 2020). Green bananas are the primary source of carbohydrates for around 30% of Tanzanians (Suleiman, 2018). They provide a sustainable source of food supply because they produce fruits throughout the year. Bananas can be eaten fresh as fruits, fried, cooked, and processed to make juice, beer, and baby food (Daniel, 2016). Ripe bananas are a cheap source of energy, vitamins, and potassium. Cooking bananas is an essential meal for millions of people (Voora *et al.*, 2020). Despite their many advantages, bananas are highly infected by diseases, including Fusarium Wilt and Black Sigatoka fungal diseases (Sanga *et al.*, 2020).

2.2 Black Sigatoka

The symptoms of Black Sigatoka disease normally start 10 to 14 days after infection by having small and pale-yellow spots on young banana leaves (Soares *et al.*, 2021). Within a few days, the spots enlarge to a few centimetres, turn brown, and have light grey centers, and then the tissue surrounding the lesions deteriorates and turns yellow as these patches spread (Soares *et al.*, 2021). This leads to the entire leaf becoming brown which interferes with photosynthesis and eventually, the leaf dies as lesions combine. The disease causes uneven and premature ripening of the banana fruit. Production of conidia and ascospore at stages 2 to 4 and 5 to 6 occurs respectively. Black Sigatoka disease affects all banana varieties, with a few exceptions that are tolerant to the disease, including the recently released Taliban 1-4 species (Shimwale, 2021). Conidia and ascospores can become windborne in diseased plantations and move up to tens of kilometres from the site of the disease, but ascospores are more crucial to the epidemiology of the disease's transmission by windborne dispersal. However, exposure to sunlight's UV rays seems to limit the long-distance airborne dissemination of viable spores. Dew, rain, and irrigation splashes are ways in which water can spread the disease over short

distances (Churchill, 2010; Muimba-Kankolongo, 2018). The disease can be controlled using weekly fungicide applications (Vézina & Van-den-Bergh, 2020). It is recommended to alternate protectants and systemic fungicides to delay or manage fungicide resistance. Production of resistant strains to fungicides is a common phenomenon that makes the management of this disease to be complicated (Isaza *et al.*, 2016). Again, it is difficult for smallholder farmers to control the disease due to fragmented farms (Isaza *et al.*, 2016). Therefore, early detection of the disease is important for its proper management.

Black Sigatoka affects the banana plant leaf in six different stages. Stage 1 appears as little, yellowish dots that are below 1 mm in size on the underside of the leaf. Stage 2 appears as red or brown streaks first on the leaf's underside, then on its upper side. On the leaf's upper surface, the streak will progressively turn black. The streaks' diameters get longer and larger in stage 3. Stage 4 appears as a brown stain on the leaf's bottom surface and a black spot on the leaf's upper surface. The spot seems circular or elliptical. The first necrotic stage is stage 5. The stain has reached the underside of the leaf blade and is entirely black with a yellow halo surrounding it. In stage 6, the center of the spot becomes light grey, dries out, and is surrounded by a prominent black ring and a light-yellow halo. These marks are still noticeable after the leaf has dried out because the ring remains persistent (Vézina & Van-den-Bergh, 2020). Figure 1 shows the different stages in which Black Sigatoka affects the banana leaves.

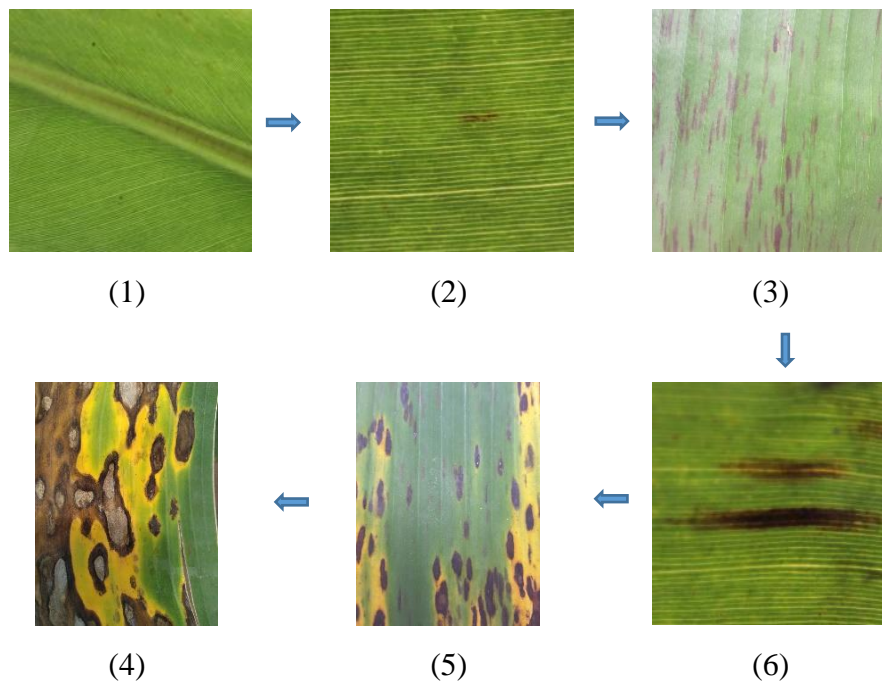


Figure 1: Different stages in which Black Sigatoka develop and affect the banana leaves

2.3 Fusarium Wilt

Fusarium Wilt infection starts in the roots where the fungal spores (Chlamydospores, macroconidia, and microconidia) infect the roots of banana plants and then spread in the corm (Vézina, 2022). Fusarium Wilt symptoms show on the older leaves to wilt and become yellow then the new leaves follow (Viljoen *et al.*, 2016; Jackson, 2014). As the illness advances, the yellowed, wilted leaves eventually collapse, forming a covering of dead leaves surrounding the pseudostem of the banana plant (Altendorf, 2019). This continues until all the leaves fall and dry up, at which point the plant dies. The splitting of the pseudostem's base is another common sign (Viljoen *et al.*, 2016; Jackson, 2014). Figure 2 shows the effects of Fusarium wilt on banana stalks and leaves. The dying bananas release chlamydospores. These spores can survive as endophytes in the soil for more than 30 years, multiplying in various hosts like weeds (Vézina & Rouard, 2021). Chemical pesticides and fungicides cannot be used to control the fungus. The easiest way to ensure that bananas can be grown is to plant resistant cultivars on affected soil or to start plantations on unaffected land (Vézina, 2022). Banana varieties that are highly susceptible to Fusarium Wilt disease are Mchare, Sukari Ndizi, all kinds of Pisang species, and Kayinja (Jomanga *et al.*, 2022; Jomanga & Lucas, 2021). The movement of contaminated planting materials, furrow irrigation, surface runoff water, and diseased soil are common ways that the soil-borne fungus is disseminated (Jackson, 2014). The spread may also be facilitated by contaminated soil on vehicles, tools, and shoes (Daly & Walduck, 2006; Altendorf, 2019).

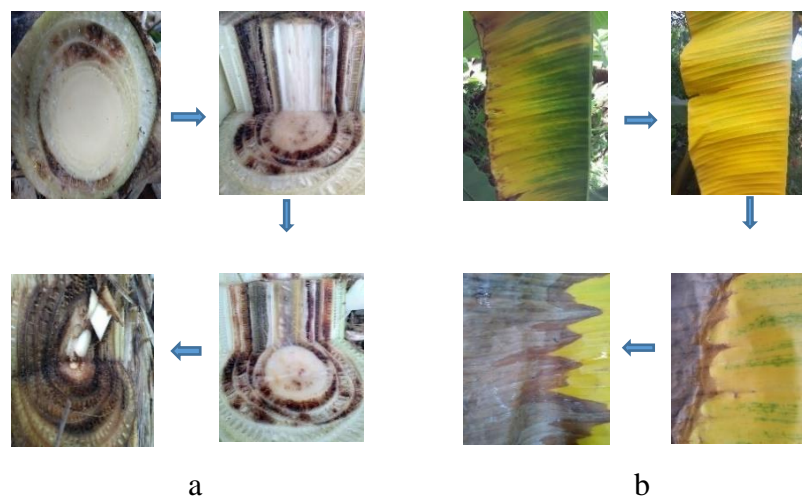


Figure 2: Damage caused by Fusarium Wilt on banana plants (a) How Fusarium Wilt affects the banana stalk (b) How Fusarium Wilt affects banana leaves

2.4 Deep Learning Models and Feature Extraction

Deep learning is a subset of machine learning that trains a computer to make decisions like human beings by learning from examples (Ral, 2020). Deep learning is mainly used with unstructured datasets like images, videos, audio, texts, sensors, and time series data. With these datasets, deep learning can solve problems like image identification and object detection using images and video datasets, solve speech recognition problems using the audio dataset, solve natural language processing problems like question answering, machine translation, sentiment analysis, and text classification using text dataset, as well as analyze sensor data and time series data. With image datasets, deep learning can be used in object localization like image segmentation and object detection, or in image classification. In this study, a convolutional neural network model was trained for classification, and Mask R-CNN and U-Net models were trained for instance and semantic segmentation tasks respectively. Feature extraction is done automatically by deep learning models. This section discusses the architectures and feature extraction done by the models assessed in this study.

2.4.1 Convolutional Neural Network Feature Extraction

The two stages of the CNN model's operation are feature extraction and classification. Figure 3 shows the entire process of a CNN model.

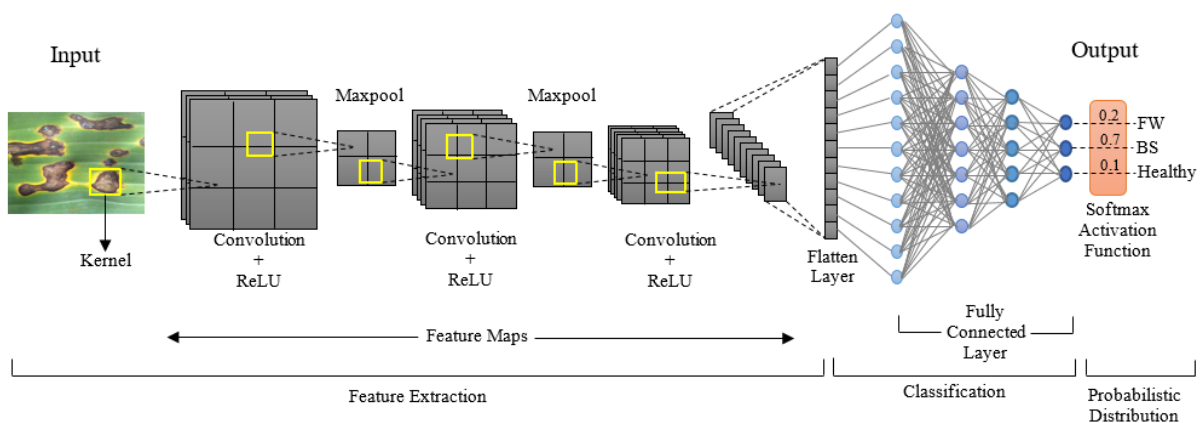


Figure 3: The CNN architecture illustration

After the feature extraction phase, which involved using numerous filters and layers to extract information and characteristics from the images, the images were categorized in the classification stage by their classes.

The feature extraction process includes the following:

- (i) Input layer
- (ii) Convolution layer and activation function
- (iii) Pooling layer

The output feature maps from the feature extraction process are run in a flattened layer before being fed into the fully connected layer or dense layer. The classification phase includes the fully connected layer and activation function.

(i) Input Layer

The input images are coloured (red, green, and blue) (RGB). Each image has pixels that range from zero to 255. Before introducing the images to the model, they were normalized by converting them to a range of zero to one. Input images were resized to 512x512. Therefore, the input shape was 512x512x3, where 3 is the colour channel.

(ii) Convolution Layer

Multiple filters were applied to the input image in a convolution layer to extract its features. Each filter was applied to all parts of the image to extract feature maps that help classify the image. Figure 4 shows examples of different filters applied to the same input image.

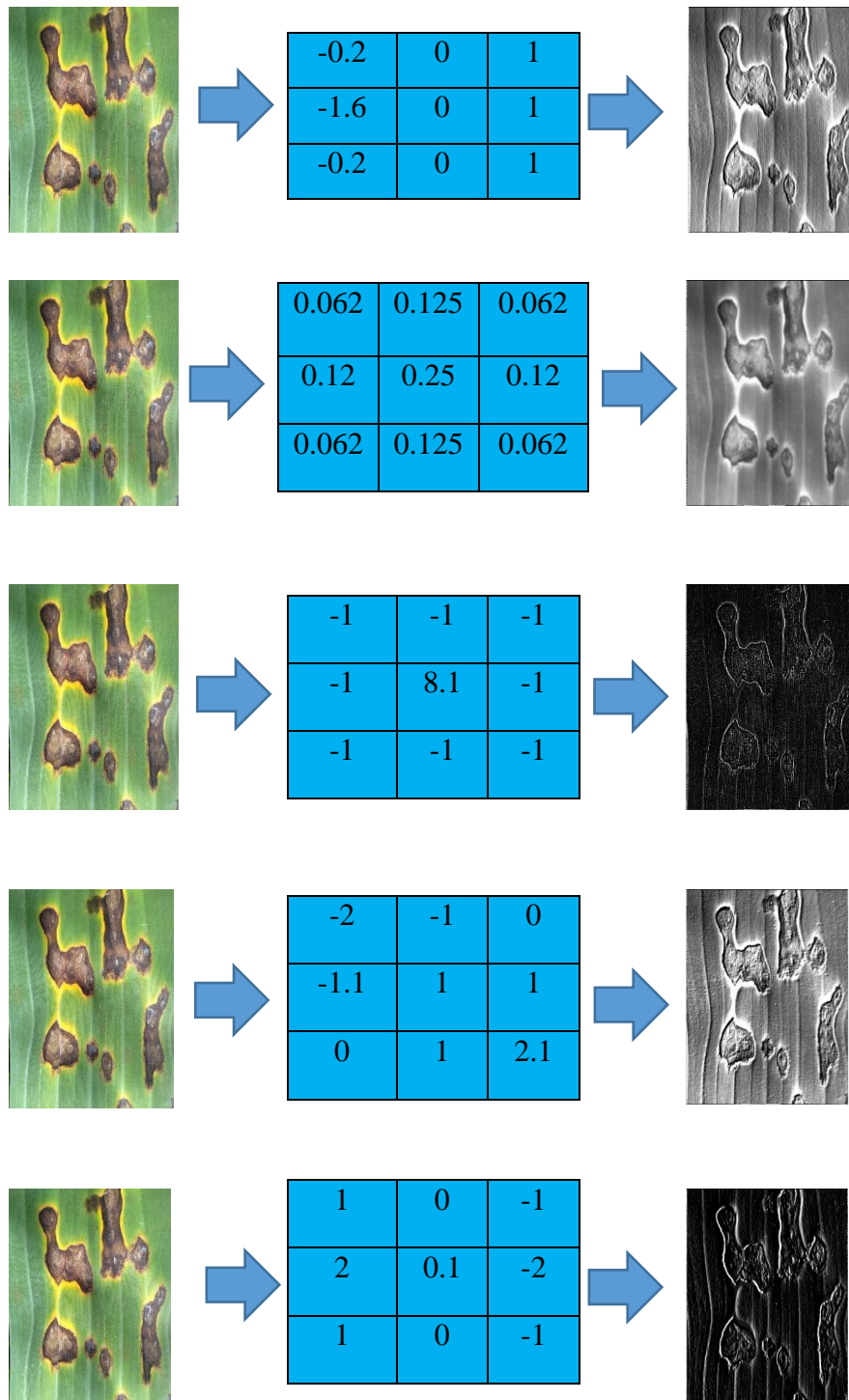


Figure 4: Example of different filters applied to the same input image

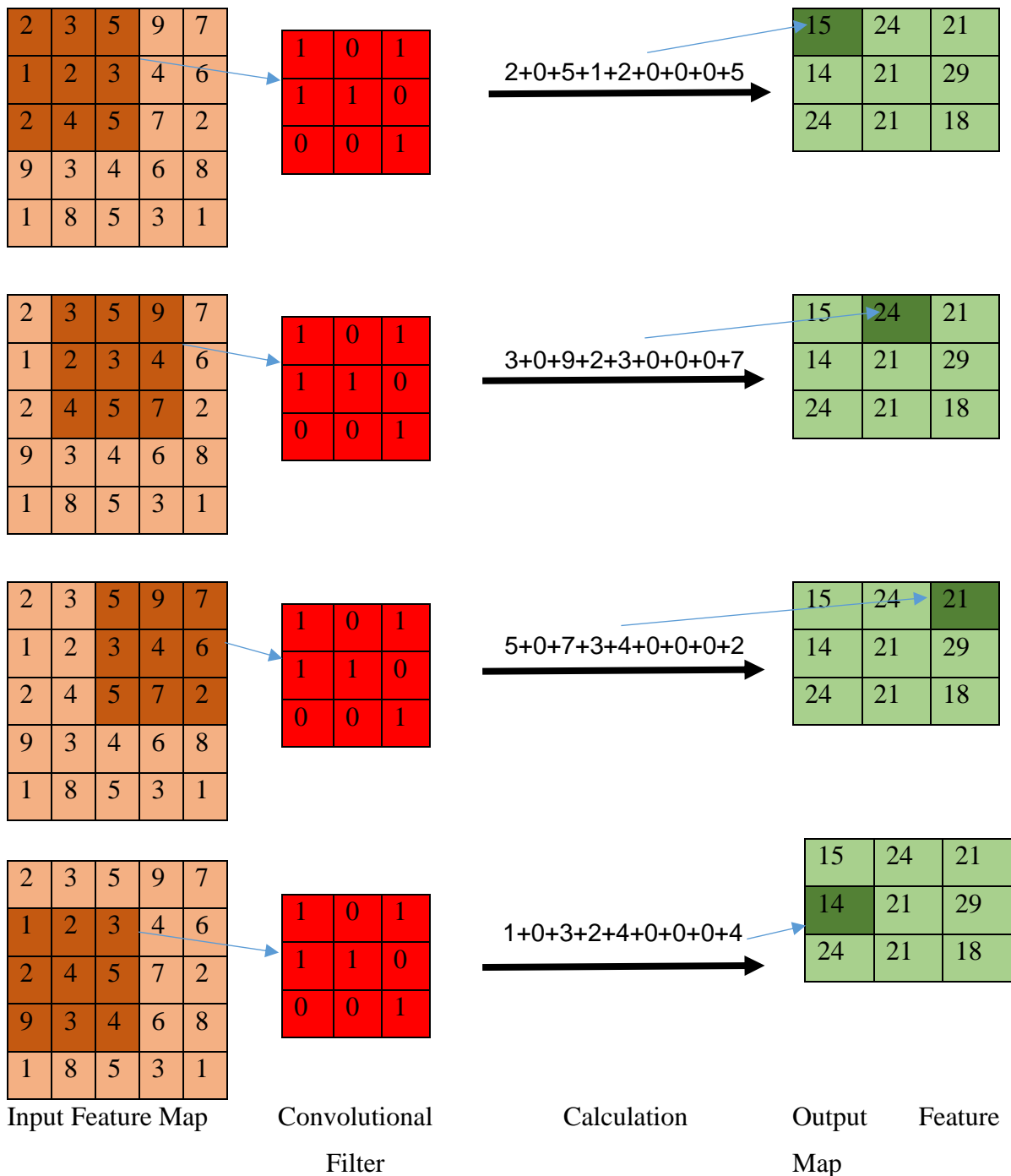


Figure 5: An example of the convolutional filter being used to transform an input feature map into an output feature map

Figure 5 illustrates the process of applying a 3 x 3 convolutional filter to the entire input image to create an output feature map. Depth relates to how many filters were applied during the convolution operation. For example, in Fig. 4, there is an example of five filters being applied to the same input image, resulting in five different output feature maps. The depth in Fig. 4 was five. As the number of filters increased, more accurate results were obtained. The stride refers to how many pixels our filter matrix slides over the input matrix. A stride of one means that

the filter moves over one pixel every time. A stride of two indicates that the filter jumps two pixels at a time as it slides around the input matrix. The feature maps get smaller as the stride gets bigger. In Fig. 5, a stride of one is used. There are occasions when it is desirable to pad the input matrix with zeros around the border to use the filter to the bordering sections of our input image matrix. The benefit of zero padding is that it enables us to control the size of the output feature maps. Not using zero padding is referred to as a "narrow convolution" while zero padding is often referred to as a "wide convolution."

Figure 6 shows an example of adding zero padding to an input image. Adding zero-padding in Fig. 6 results in an output feature map with the same size as the input feature map, which is 5x5. While the same 3x3 filter was used in Fig. 5, the lack of zero-padding on the input image resulted in an output feature map of size 3x3.

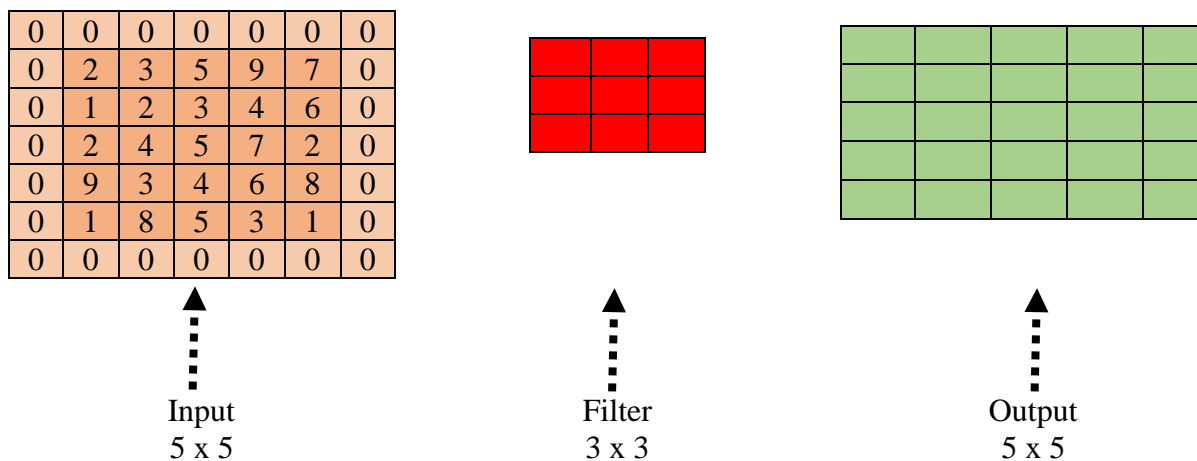


Figure 6: An example of padding the input matrix with zeros around the border so as to enhance the output feature map size

The *Rectified Linear Unit* (ReLU) activation function was used on every convolution operation. It is a function that replaces all negative numbers with zero and returns a number if it is larger than zero. The ReLU helps us prevent the transmission of negative values to the following layer, which can affect the summing function. Figure 7 shows how the ReLU activation function is applied, and replaces all the negative numbers with zero.

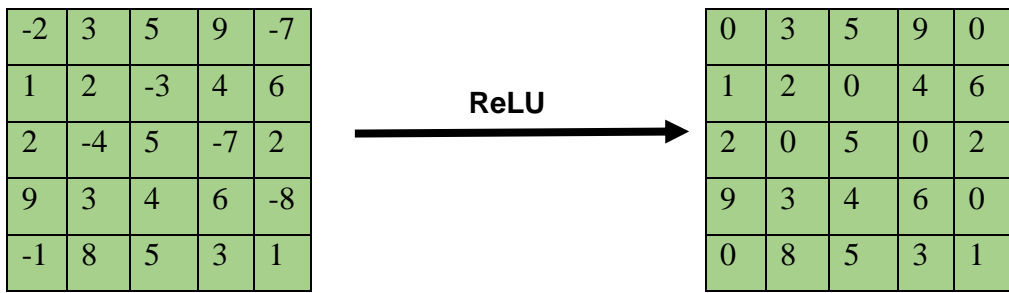


Figure 7: An illustration of the application ReLU operation

(iii) Pooling Layer

The pooling layer, which reduces the dimensionality of each feature map while preserving the most crucial features, is also known as down-sampling or subsampling. Pooling is also described as the method that minimizes the number of pixels in your image while preserving its semantics. The kinds of pooling include Max, Average, and Sum pooling. Figure 8 shows the process of max pooling. The left-hand box in Fig. 8 could represent the pixels in a black-and-white image. The pixels are further grouped into 2x2 arrays, so from 16 pixels, 4 groups of 2x2 arrays are obtained. The groups are called pools. Then, the maximum value (Max Pooling) is chosen in each group, and those values are put back together to create a new image. Sum pooling takes the total of all the values in each group, average pooling takes the average value from the groups. As a result, the pixels on the left image are decreased by 75% (from 16 to 4), and the new image is made up of the maximum value from each pool (Moroney, 2021).

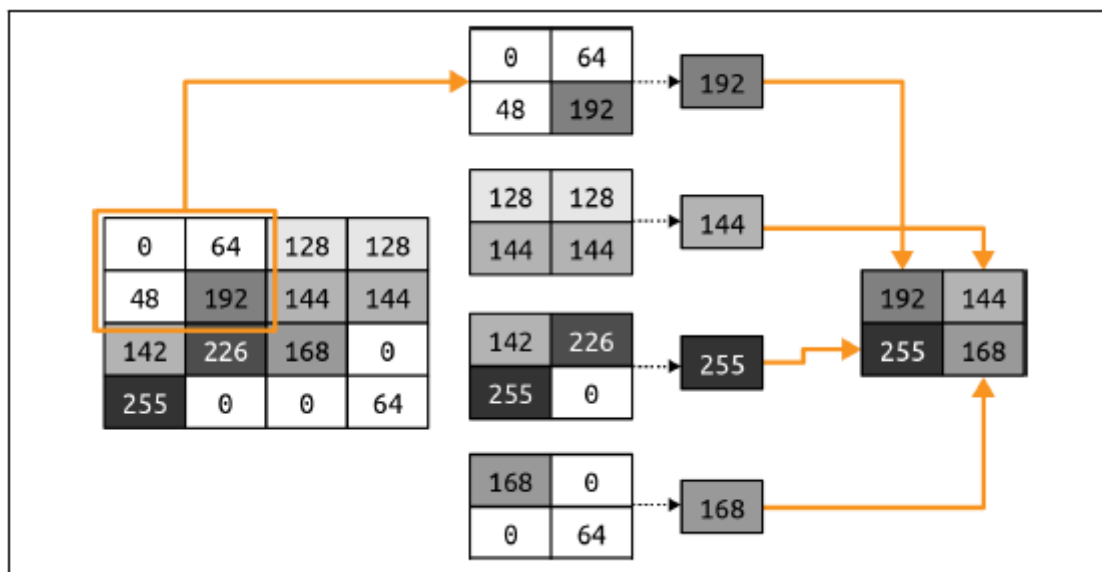


Figure 8: Max Pooling illustration (Moroney, 2021)

2.4.2 Mask Region-Based Convolutional Neural Network Model for Instance Segmentation

Instance segmentation merges the objectives of object detection which aims to classify each object and locate it with a bounding box and semantic segmentation whose goal is to classify each pixel into predefined categories without distinguishing object instances. In addition to the classification and bounding box regression branches, the Mask Region-based Convolutional

Neural Network is a Faster R-CNN extension that includes a segmentation mask predicting branch on every region of interest (He *et al.*, 2018). For each object, Faster R-CNN returns a bounding box and its class label with a confidence score (Ren *et al.*, 2016). Mask R-CNN works more efficiently in instance segmentation because it decouples mask and class prediction (He *et al.*, 2018).

(i) Mask Region-Based Convolutional Neural Network Model Architecture

Figure 9 depicts the architecture of the Mask R-CNN model.

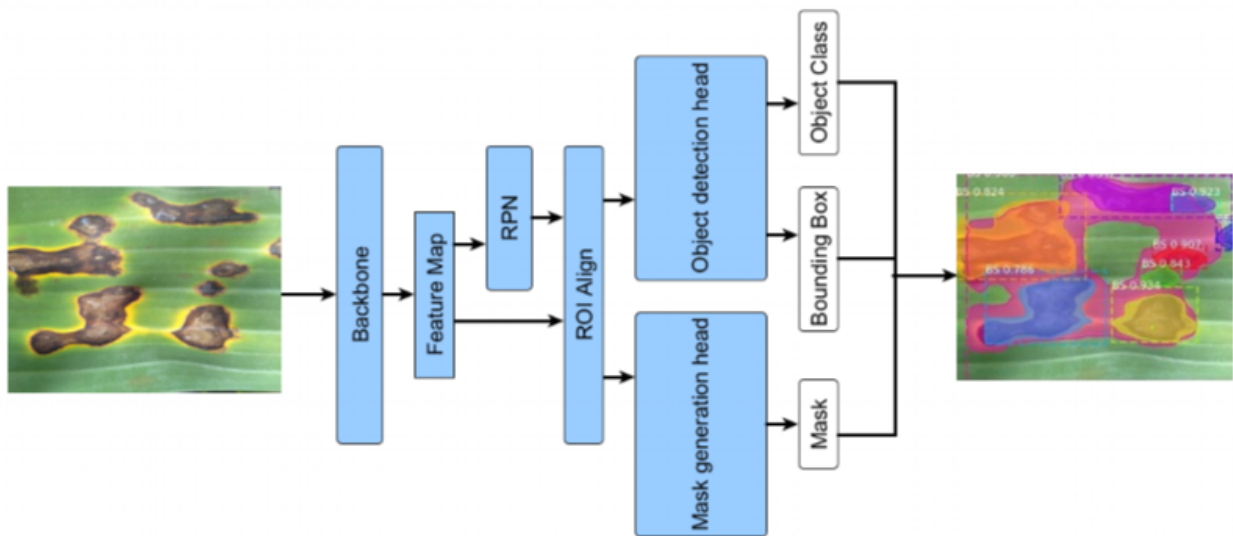


Figure 9: Assessed Mask R-CNN model architecture

Backbone

Backbone is the primary feature extractor of Mask R-CNN. Residual networks (ResNets) with or without feature pyramid network (FPN) are frequently used for this component. When data from an image is introduced into a ResNet backbone, it must first pass through several bottleneck blocks before it can be converted into a feature map. Figure 10 illustrates a ResNet backbone.

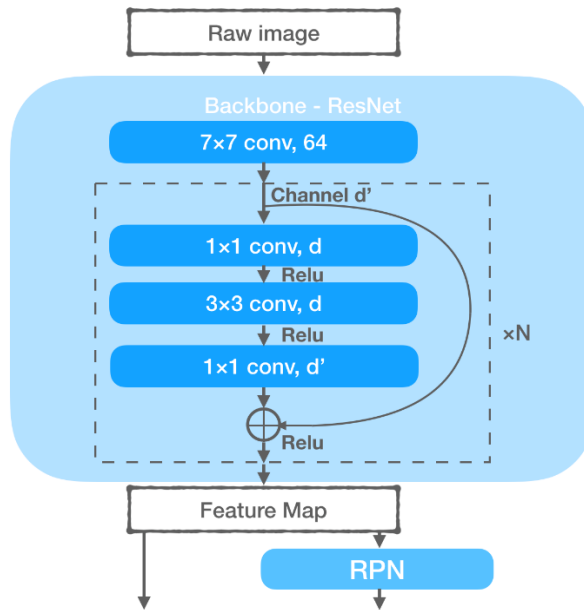


Figure 10: Mask R-CNN ResNet backbone (Zhang, 2021)

Region proposal network

The function known as the Region Proposal Network (RPN) scans the feature map from the backbone and suggests regions that might contain objects of interest, i.e., Regions of Interest (ROI). Figure 11 shows the process of the RPN.

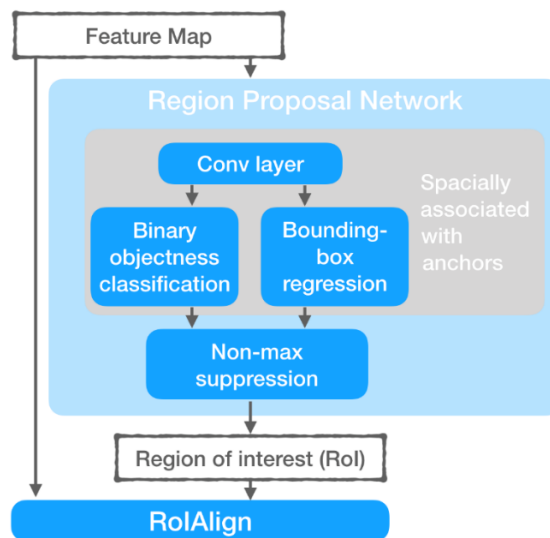


Figure 11: Region proposal network (Zhang, 2021)

Region of interest alignment

Based on the Regions of Interest (RoIs) suggested by the Region Proposal Network (RPN), Region of Interest alignment (RoIAlign) extracts feature vectors from a feature map and

converts them into a fixed-sized tensor for further processing. Scaling is used by RoIAlign to match RoI with their corresponding locations on the feature map. The following parallel branches for object detection and mask generation process the resulting RoI's finer feature map. Figure 12 illustrates RoI Align.

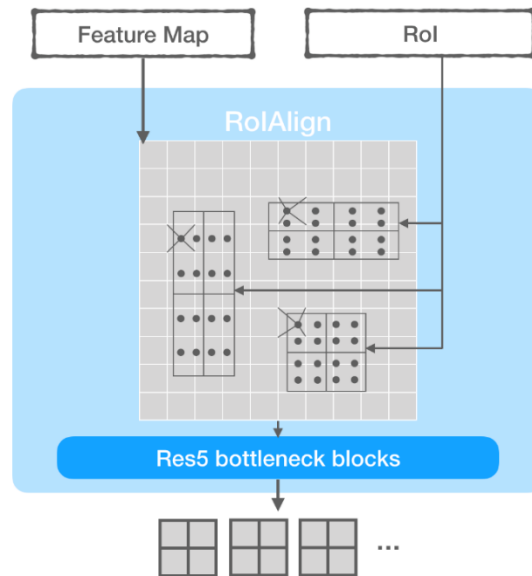


Figure 12: RoI Align (Zhang, 2021)

Object detection branch

An individual RoI object category and a more precise instance bounding box can be predicted based on the individual RoI feature map. Figure 13 shows the object detection head.

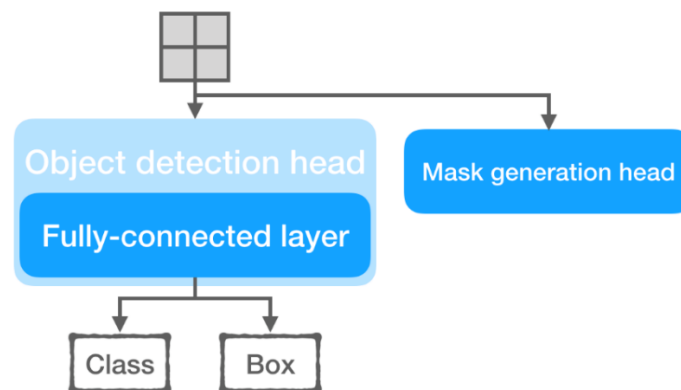


Figure 13: The Object detection head (Zhang, 2021)

Mask generation branch

A transposed convolutional layer and a convolutional layer on the mask-generating branch are progressively fed the RoI feature map. This is a fully convolutional network branch. For one

class, a single binary segmentation mask was generated. The output mask was then selected based on the class prediction made by the object detection branch. This aids per-pixel mask prediction in preventing competition between multiple classes. Figure 14 illustrates the mask generation head.

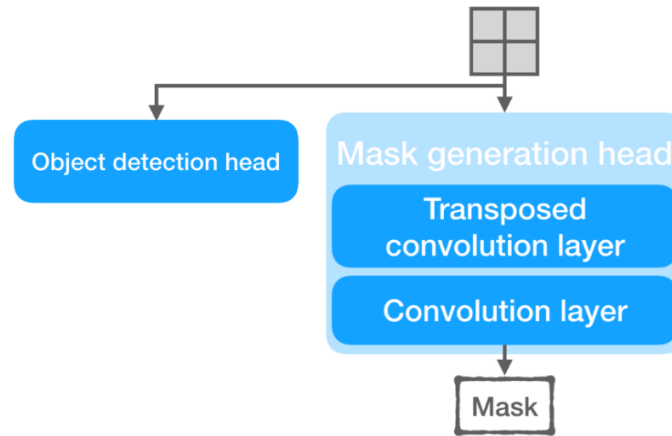


Figure 14: The Mask generation head (Zhang, 2021)

(ii) Mask Region-Based Convolutional Neural Network Feature Extraction

As discussed in the Mask R-CNN architecture section in the Backbone subsection, the Backbone is the main feature extractor of Mask R-CNN. Data is routed through numerous bottleneck blocks as images are supplied into the ResNet backbone before being transformed into a feature map. As illustrated in Fig. 10, to make up a deep residual network, several residual bottleneck blocks were stacked. In a bottleneck block, input passes in two directions: the multiple convolutional layers and the other identical shortcut connection. Their outputs are then added element-wise. The feature map from the final backbone's convolutional layer comprises abstract information about an image, which includes different object instances, their classes, and spatial attributes (Zhang, 2021). Because the residual bottleneck blocks in the ResNet backbone comprise convolutional layers, the feature extraction done in this part will be discussed in the CNN feature extraction.

As illustrated in Fig. 11, in the Region Proposal Network (RPN), the backbone's output feature map was processed by a convolutional layer, which generates a c -channel tensor whose each spatial vector (also having c -channels) is associated with an anchor center. A set of anchor boxes with varied scales and aspect ratios are formed from a single anchor center. The anchor boxes are different regions that are evenly spaced throughout the entire image and fully encircle it. The c -channel tensor is then processed by two siblings 1×1 convolutional layers. One is a

binary classifier that predicts if each anchor box contains an object. Each c-channel vector is mapped to a k-channel vector (represents k anchor boxes with various scales and aspect ratios sharing one anchor center). The second is an object-bounding box regressor. It predicts the offsets between the anchor box and the true object bounding box. Each c-channel vector is mapped into a 4K-channel vector. The bounding boxes with the highest objectness score are selected out of any overlapped bounding boxes that might suggest the same object and discard the others. This is the non-max suppression process (Zhang, 2021).

RoIAlign is the step that finds out exactly where each RoI from RPN is in the feature map. Imagine that our model translates a 512x512x3 (width, height, and RGB) input image into a 16x16x512 feature map with a scale factor of 32 using VGG16 (Fig. 15).

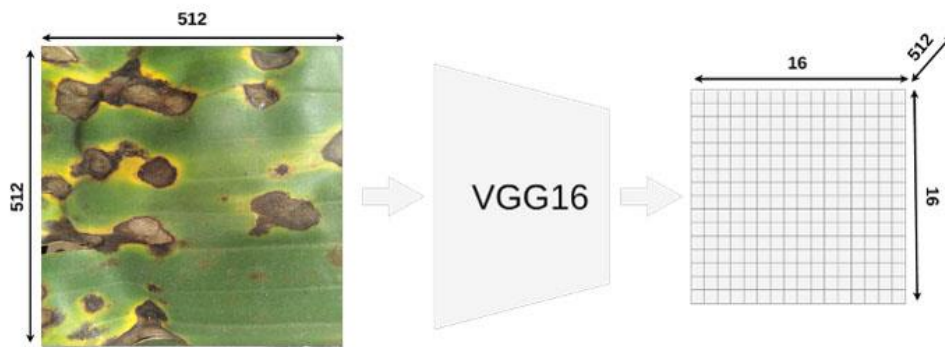


Figure 15: Extracting a feature map from an input image using VGG16

Then a single proposed RoI (145x200 box) was used and mapped onto the feature map (Fig. 16).

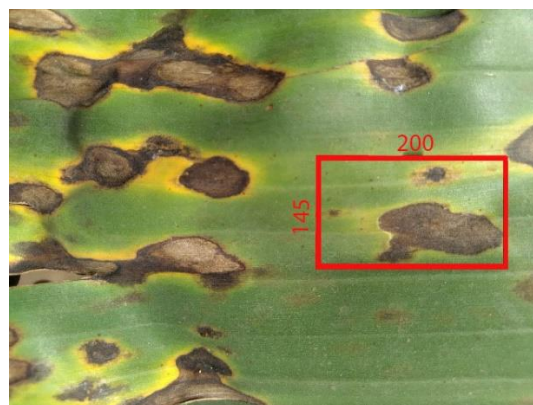


Figure 16: The RoI target with coordinate size

Since certain of the dimensions of objects cannot be divided by 32, RoI (not align) was placed with our grid (Fig. 17).

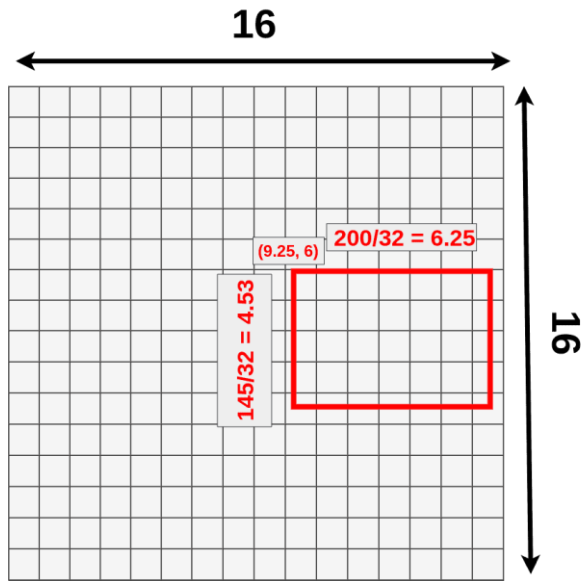


Figure 17: The RoI on the feature map (Erdem, 2020)

As an example, the pooling layer is specified to be of size 3x3, so the output shape is 3x3x512, as seen in Fig. 18.

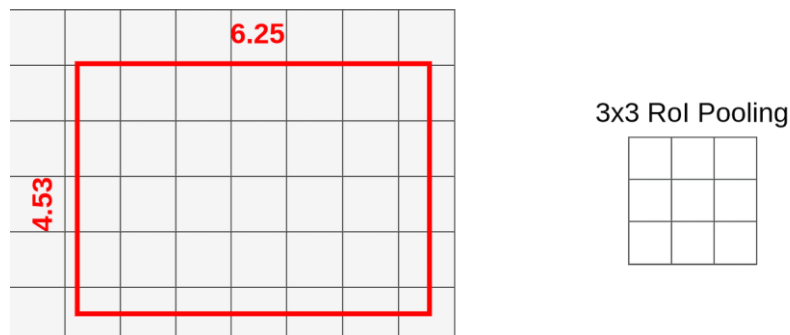


Figure 18: Pooling layer (Erdem, 2020)

Region of Interest Alignment is a quantization-free layer applied by Mask R-CNN to faithfully preserve exact spatial locations. This is an improvement to RoIPool, which was used by Faster R-CNN. The RoIPool does coarse spatial quantization for feature extraction (He *et al.*, 2018). RoIPool's quantization technique restricts input from real numbers to integers (Erdem & Kemal, 2020). Two types of quantization are applied by Fast R-CNN. The mapping process does the first quantization then the pooling process does the second quantization (Fig. 19). The problem with the quantization used in RoI Pooling is that it generates a great loss of data as shown in Fig. 20. Every time quantization is done, a part of the information regarding the object of interest (RoI) goes missing.

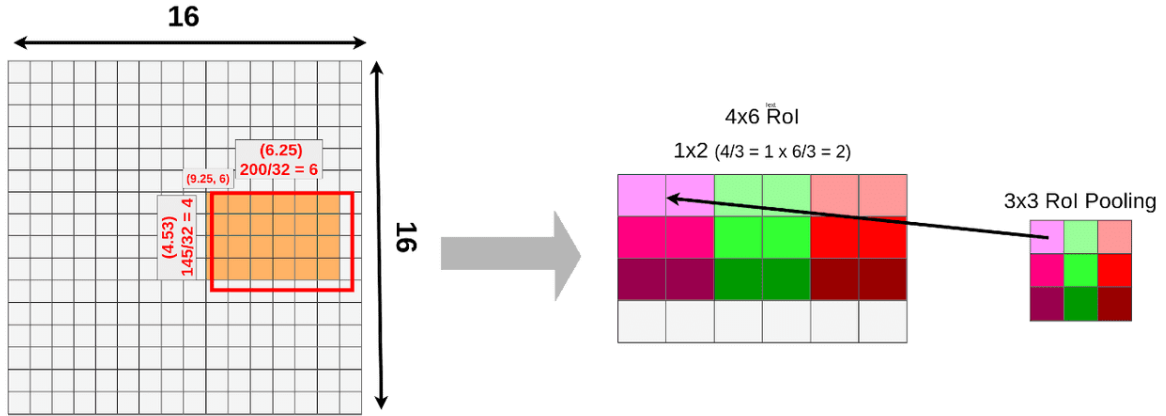


Figure 19: Quantization when mapping and pooling (Erdem, 2020)

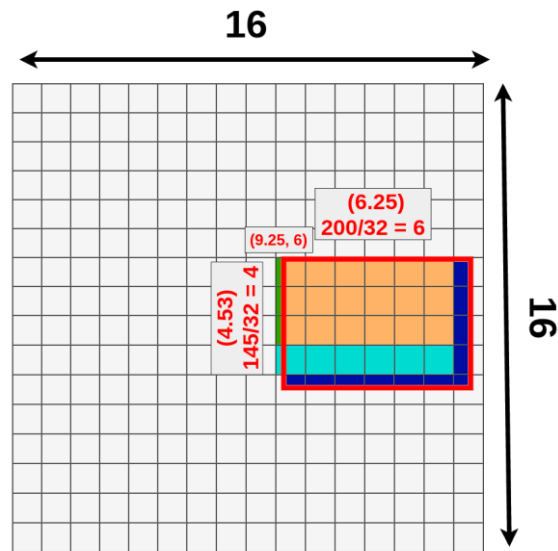


Figure 20: The RoI pooling quantization losses (shown by the light and dark blue colors) and data gain (represented by the green color) (Erdem, 2020)

In RoIAlign, the quantization process was skipped by partitioning the original RoI into nine boxes of the same size and using bilinear interpolation inside each one, as illustrated in Fig. 21. The pooling layer's size and the mapped RoI together define the size of each box. Considering that a 3x3 pooling layer is utilized, the mapped RoI (6.25x4.53) was divided by three. This results in a rectangle with a width of 2.08 and a height of 1.51.



Figure 21: The RoI box size (Erdem, 2020)

Figure 22 shows how our RoI is divided into boxes.



Figure 22: The RoI divided into equal boxes (Erdem, 2020)

In Fig. 22, the top left box covers six different grid cells. Part of the data must be sampled to extract value for the pooling layer. Four sampling points were established inside that box to sample the data, as shown in Fig. 23.

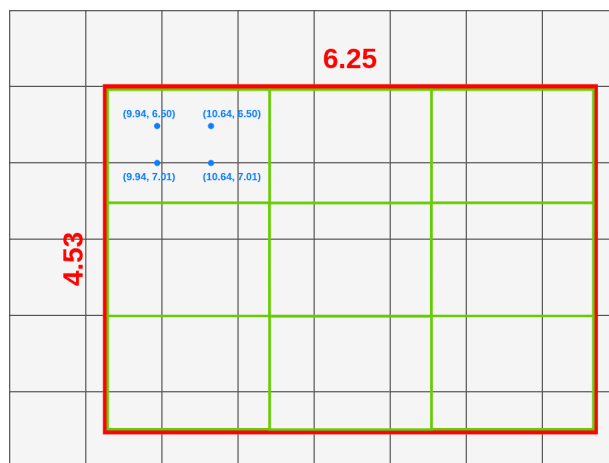


Figure 23: Sampling points distribution (Erdem, 2020)

The points are calculated by dividing the box's height and width by 3. After obtaining all the points, Bilinear interpolation is applied to the sample data from this box. In image processing, bilinear interpolation is frequently employed to sample colors using the equation below:

$$P \approx \frac{y_2 - y}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} Q_{11} + \frac{x - x_1}{x_2 - x_1} Q_{21} \right) + \frac{y - y_1}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} Q_{12} + \frac{x - x_1}{x_2 - x_1} Q_{22} \right)$$

Figure 24 shows the bilinear interpolation for the sampling points.

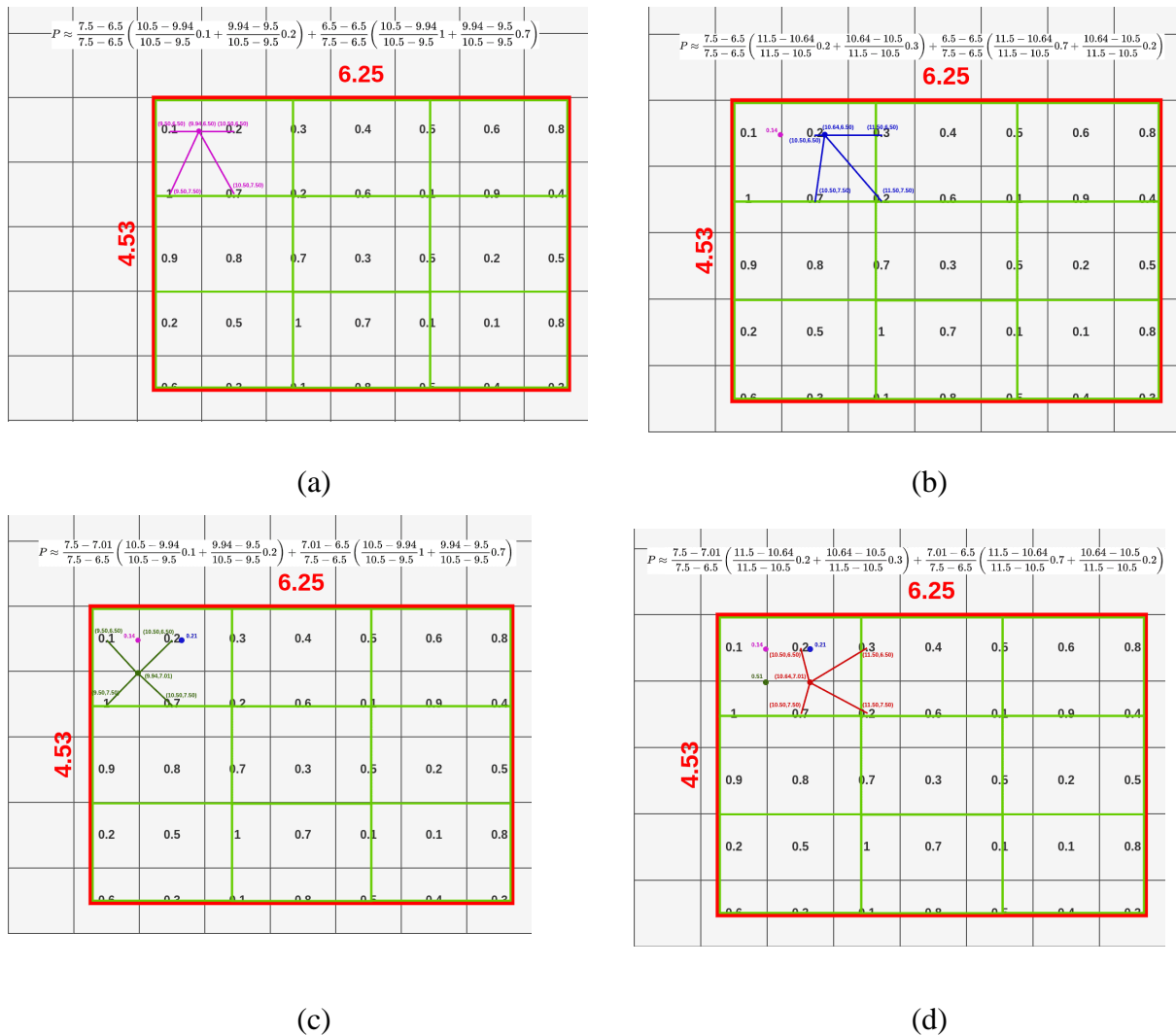


Figure 24: Bilinear interpolation for the (a) first point, (b) second point, (c) third point, and (d) fourth point (Erdem, 2020)

Max Pooling is used once all the points have been determined, as shown in Fig. 25.



Figure 25: Max Pooling on first box (Erdem, 2020)

Data are pooled from the entire RoI using RoIAlign as illustrated in Fig. 26.

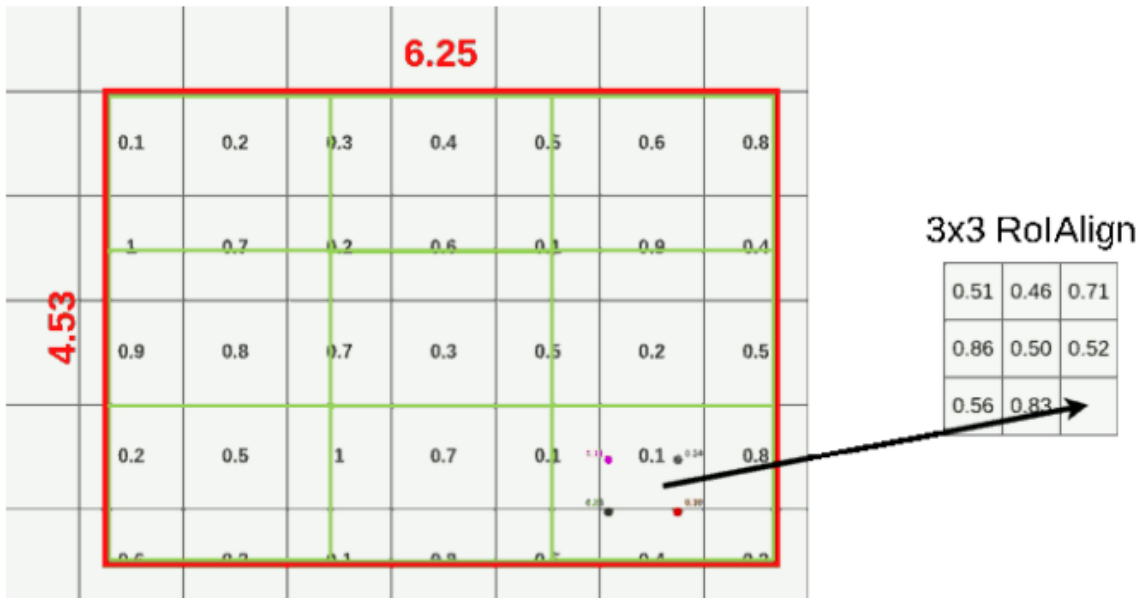


Figure 26: The RoIAlign pooling output (Erdem, 2020)

Region of Interest Alignment pooling is applied for every layer until a result containing 512 layers is obtained as feature map input, as seen in Fig. 27. In RoIAlign, using bilinear interpolation, data is extracted from all cells in the feature map inside the RoI even though sampling points are not placed in all cells (Erdem, 2020).

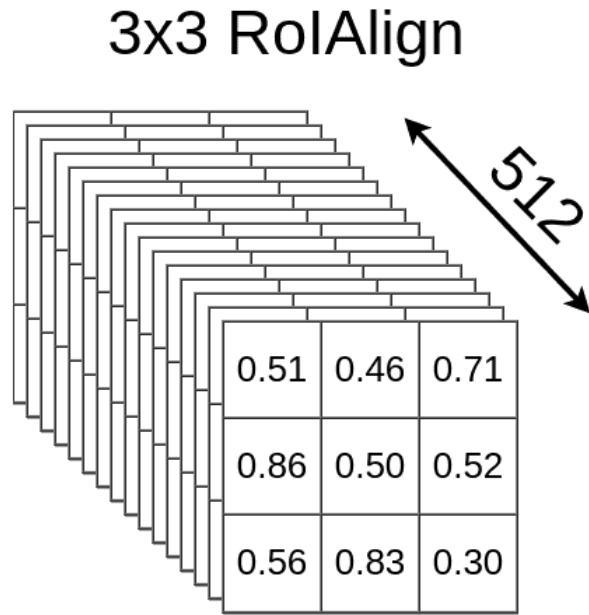


Figure 27: The RoIAlign full size (Erdem, 2020)

2.4.3 The U-Net Model for Semantic Segmentation

Semantic segmentation assigns a class to each pixel in an image. It does not separate different instances of the same class. The U-Net model was used for semantic segmentation. The U-Net gains end-to-end image segmentation skills by receiving a raw image and producing a ready segmentation map (Ronneberger, 2015). The foundation of U-Net is a fully convolutional network (Long *et al.*, 2015). The U-Net extends the fully convolutional network architecture to use a few annotated training images (relying excessively on the use of data augmentation) and still yields precise segmentations (Ronneberger *et al.*, 2015).

(i) The U-Net Model Architecture

The U-Net is comprised of a large number of small operations, shown by small arrows, as illustrated in Fig. 28. Feature maps were represented in blue boxes. The left side of the U-shaped U-Net architecture is known as the contracting path while the right side is the expansive path (Ronneberger *et al.*, 2015).

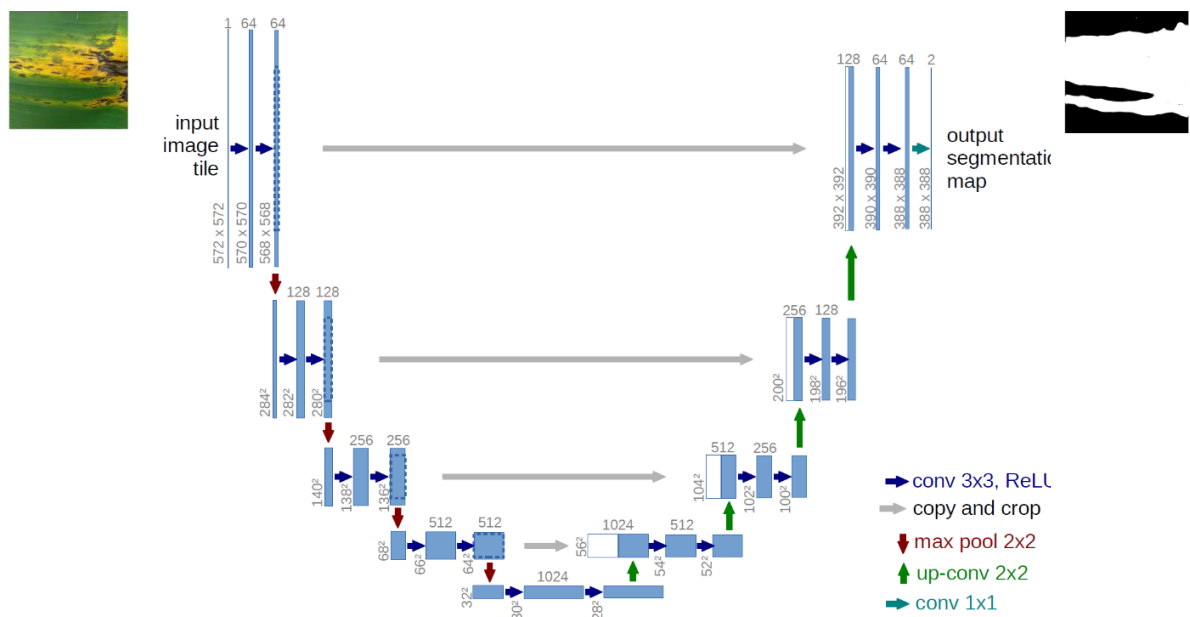


Figure 28: The U-Net architecture

(ii) The U-Net Feature Extraction

The contraction path on the right-hand side of the U-Net architecture follows a standard convolutional network. The feature extraction done in the convolutional network in the contracting path is discussed in the CNN feature extraction. The expansive path is made up of a series of up-convolutions. Up-convolutions are also referred to as transposed convolutions, which are used in up-sampling. The operations in a transposed convolution are similar to the operations in a normal convolution but go backward. This results in up-sampling an image from low resolution to high resolution. A convolutional operation produces a many-to-one relationship, i.e., the nine input matrix values are connected to one output matrix value for a 3x3 convolutional filter. Figure 29 shows the many-to-one relationship produced by a convolution operation. A transposed convolution goes backward from the operation illustrated in Fig. 29. A transposed convolution connects one value in an input matrix to nine values in the output matrix for a 3x3 convolution filter. A transposed convolution gives a one-to-many relationship. Figure 30 shows the one-to-many relationship yielded by a transpose convolution as it does up-sampling.

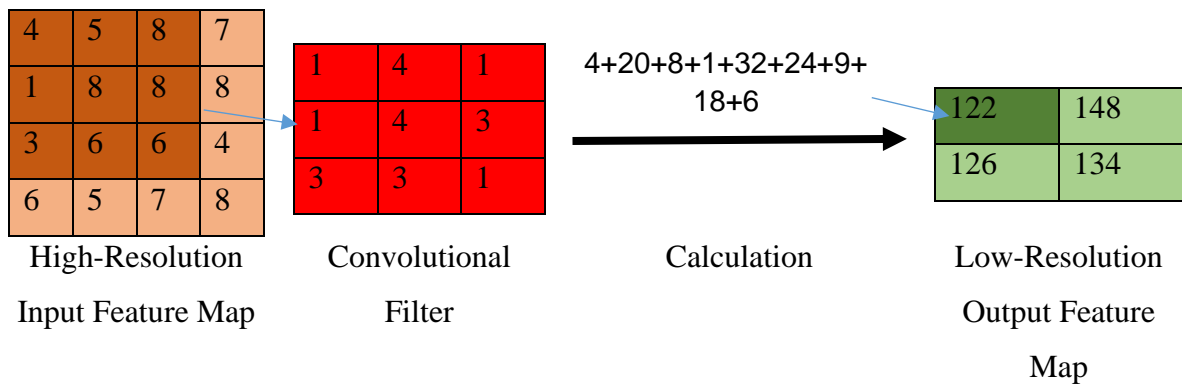


Figure 29: A convolutional operation yielding a many-to-one relationship

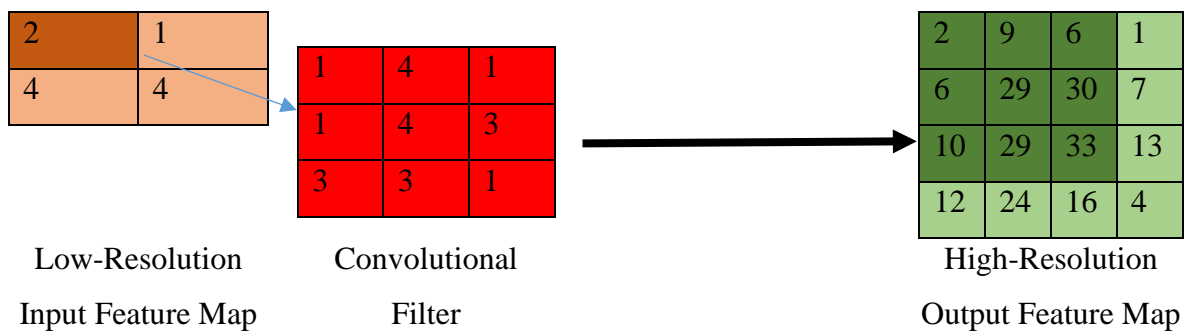


Figure 30: A transpose convolution yielding a one-to-many relationship

In Fig. 30, an attempt is made to up-sample a matrix of size 2x2 matrix to a matrix of size 4x4. The one-to-nine relationship is maintained. The transpose convolution operation done in Fig. 30 can be explained by a convolution matrix and a transposed convolution matrix.

A convolution matrix is a convolution filter that has been reorganized so that convolution operations can be produced through matrix multiplication. The 3x3 convolution filter (in red) in Fig. 29 and 30 is rearranged into a 4x16 convolution matrix by adding zero padding. Figure 31 shows a 4x16 convolution matrix, while Fig. 32 illustrates how the convolution matrix is formed.

1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

Figure 31: A 4x16 convolution matrix

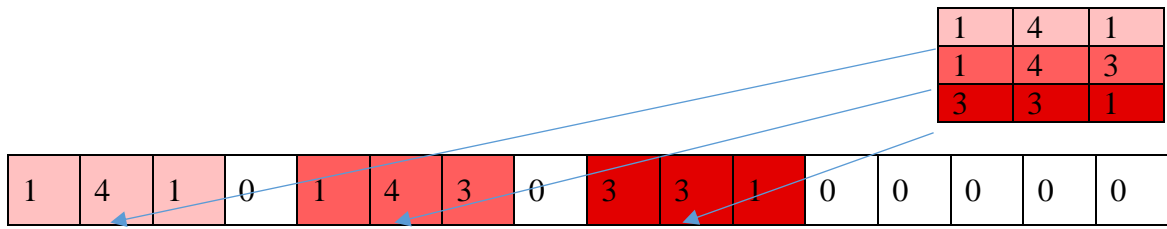


Figure 32: How a convolution matrix is formed

The input matrix is flattened from 4x4 to a column vector of 16x1 to employ the convolution matrix, as seen in Fig. 33.

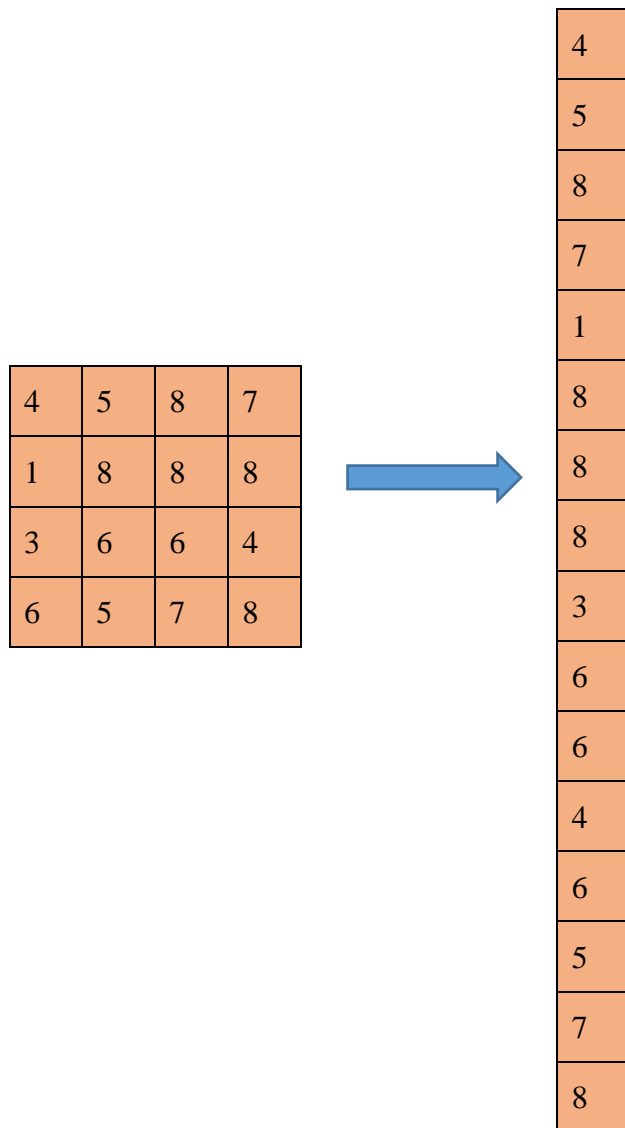


Figure 33: Converting our input matrix from 4x4 to a column vector 16x1

As shown in Fig. 34, a convolutional operation is created by multiplying the 16x1 input matrix by the 4x16 convolutional matrix.

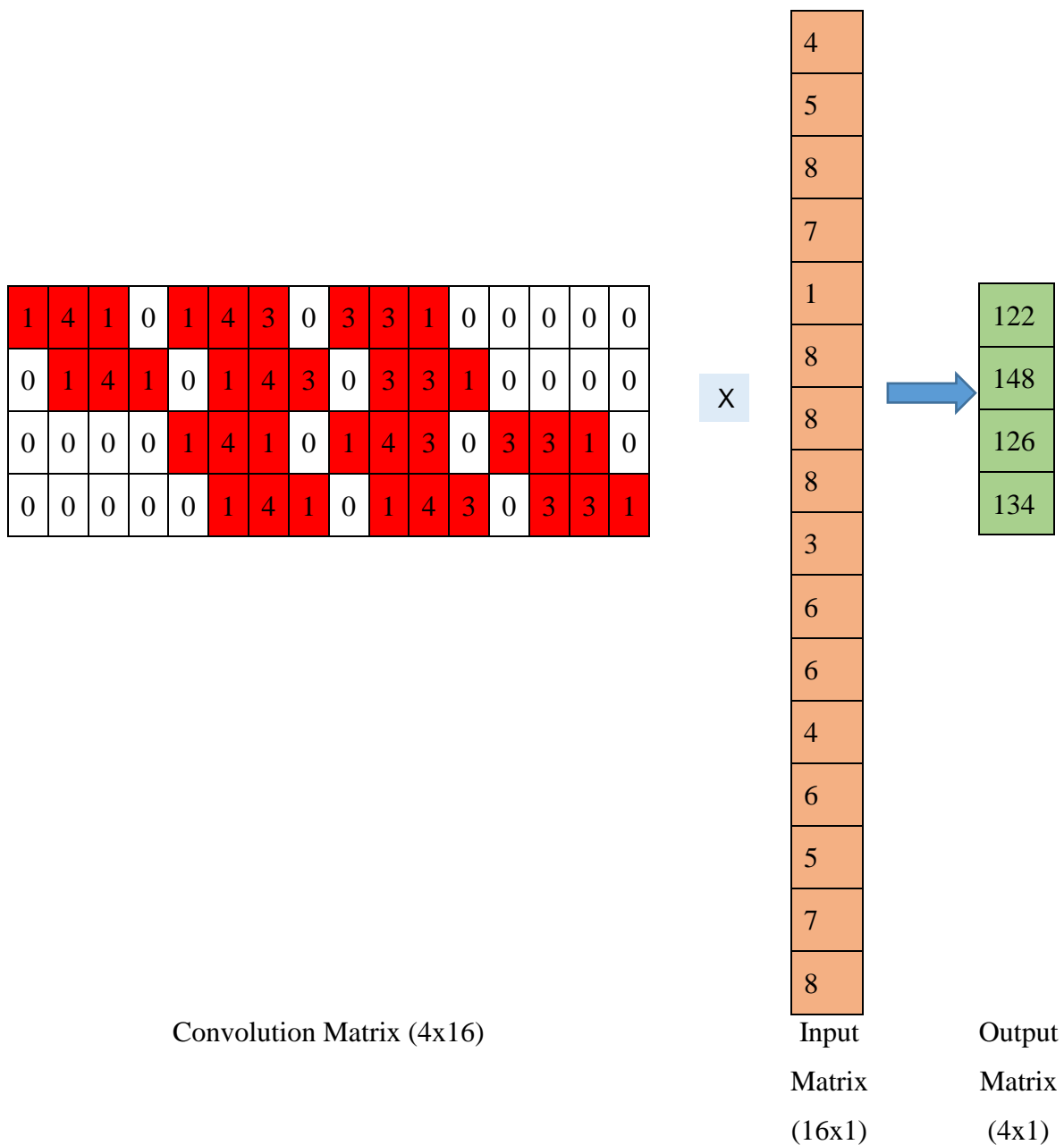


Figure 34: A convolution operation

The output matrix in Fig. 34 can be transformed into a 2x2 matrix, producing the same outcome as in Fig. 29.

When you transpose the convolution matrix (4x16) to create a (16x4) matrix, you acquire a matrix known as a transposed convolution matrix. You can use the convolution matrix to go from 16 (4x4) to 4 (2x2) because it is 4x16. Consequently, if you have a 16x4 transposed convolutional matrix, you can move from 4 (2x2) to 16 (4x4). A transposed convolutional matrix maintains the one-to-nine relationship as discussed before. Figure 35 illustrates the operation of a transposed convolution. Figure 35 also explains the transposed convolution

operation that was given in Fig. 30.

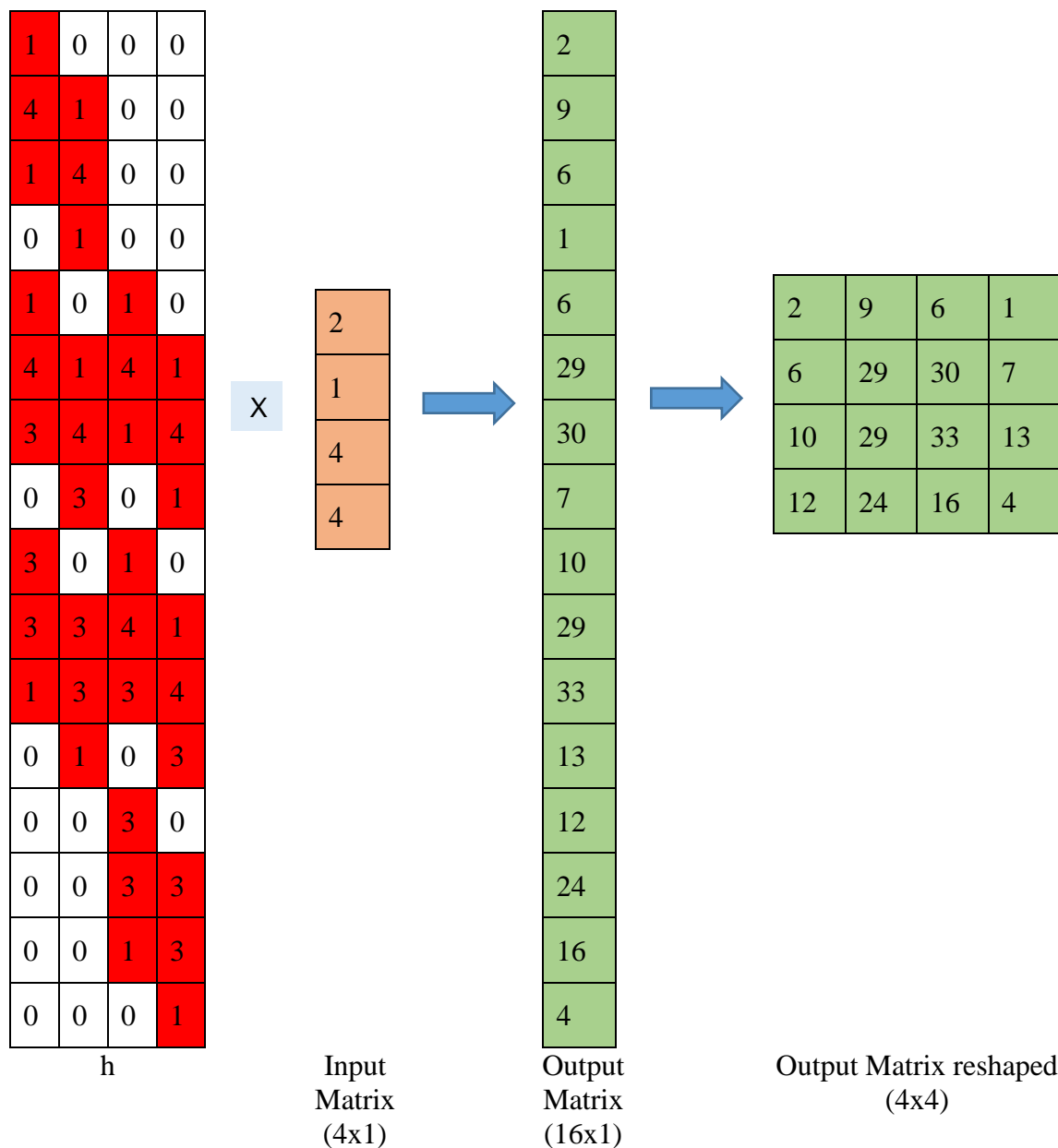


Figure 35: A transposed convolution operation

To perform a transposed convolution, it is not necessary to begin with a normal convolution. The weight layout needs to be transposed from that of the convolution matrix (Naoki, 2017).

2.5 Theoretical Literature Review

This study used a theory called the technology acceptance model. This model's selection was based on its ability to determine the attitude and behavioural intention of farmers to adopt and use our proposed product, for example, the farmers' perceptions of its usefulness and ease of

usage (Venkatesh, 2000). This was covered in the validation of the mobile application in the field with the stakeholder groups. This study also used the theory of diffusion of innovation. The selection of this theory is based on its ability to determine the rate at which consumers will adopt a new product or service (Dearing, 2018). The summary of the adopted model is presented in Fig. 36.

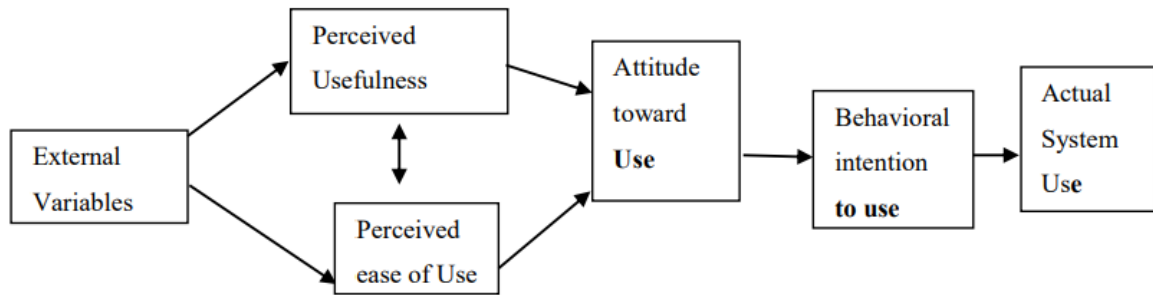


Figure 36: Technology acceptance model (Lai, 2017)

2.6 Empirical Literature Review

This section discusses related works categorized into three categories which are feature extraction, using deep learning and machine learning to detect banana diseases, and mobile applications that deploy deep learning models for plant disease detection.

2.6.1 Feature Extraction

Bodapati and Veeranjanyulu (2019) applied deep convolutional neural networks (DCNN) for image classification and feature extraction. In their study, Bodapati and Veeranjanyulu (2019) performed two tasks: doing feature extraction using DCNN and then performing classification on the extracted features using support vector machines (SVM). The DCNN architecture had three convolutional and pooling layers which were followed by a fully connected output layer for feature extraction, and these features were fed into a two-hidden-layer neural network for classification in the first task. The results of their study showed that using u-SVM for classifying features extracted from DCNN yielded slightly better performance than using s neural network for classifying features extracted from DCNN.

Zhu *et al.* (2017) proposed classification and target recognition for sonar images from unmanned underwater vehicles (UUVs) using deep learning feature extraction. Sonar image feature extraction was done using a convolutional neural network, whereby the features were

classified using an SVM that used manually labeled data for its training. Their study's findings demonstrated that deep learning feature extraction outperformed feature extraction from methods like the histogram of oriented gradients (HOG) and local binary pattern (LBP).

2.6.2 Using Deep Learning and Machine Learning to Detect Banana Diseases

Selvaraj *et al.* (2019) suggested an AI-based system that detects banana pests and diseases. The study used a large dataset of 30 952 annotated images. The images were gathered in southern India and Africa. The diseases incorporated in this study were *Xanthomonas* wilt, Bunchy top disease, Black Sigatoka, Fusarium wilt, Yellow Sigatoka, and Corm weevil. The model classes included each disease, dried or old leaves, and healthy plants. Detection models were built by training Faster R-CNN based on ResNet50 and InceptionV2 and an independent MobileNetV1 convolutional neural network architecture using transfer learning. Six models were built from each architecture to represent diseases according to plant parts, utilizing images from various banana plant parts (including the corm, fruit bunch, leaves, cut fruit, pseudostem, and entire plant). This study showed that Faster R-CNN based on the InceptionV2 and ResNet50 models had better performance compared to the MobileNetV1 model. Faster R-CNN model based on ResNet50 on the pseudostem, leaves, fruit bunch, and entire plant performed better, with mean Average Precision (mAP) of 99%, 70%, 97%, and 73%, respectively. Despite the good performance, the study struggled with unbalanced data and was limited to a small number of images for Bunchy Top disease (902 images), Black Sigatoka disease (980 images), Yellow Sigatoka disease (1066 images), Fusarium Wilt disease (1726 images), and Corm Weevil disease (701 images).

Bhuiyan *et al.* (2023) proposed a lightweight and fast convolutional neural network called the BananaSqueezeNet model for the diagnosis of Pestalotiopsis, Sigatoka, and Cordana banana diseases. This study used a dataset of 937 images collected in Bangladesh that consisted of the three banana diseases. The BananaSqueezeNet model achieved 96.25% accuracy, 96.25% recall, 96.53% precision, 96.17% F1-score, 98.75% specificity, and 95.13% MCC. Despite the good performance, the study used a small number of images.

Narayanan *et al.* (2022) suggested the use of a hybrid convolutional neural network for the classification of banana diseases. The banana diseases addressed by this study are Black Sigatoka, banana bunchy top virus, *Xanthomonas* wilt, and Fusarium wilt. The study used a dataset of 3500 images of infected and healthy banana plants collected from fields in south

India. The study integrated CNN and FSVM, which combine multiclass and binary SVM to classify banana diseases. The proposed method achieved an accuracy of 99%. Despite the good performance, the study lacked a sufficient number of images to train the model.

Amara *et al.* (2017) proposed an approach of deep learning using the LeNet architecture for the classification of banana leaf disease. This study used a dataset of 3700 annotated images belonging to the healthy, black speckle, and black sigatoka, classes. The study had several experiments based on different training to test set ratios which were evaluated using accuracy, F1-score, recall, and precision. For the 80:20 training-to-test ratio, the study achieved 92.88% accuracy, 92.99% precision, 92.88% recall, and a 92.94% F1 score for the colored images. Despite the good performance, this study was limited by a small number of black Sigatoka images which were only 240.

Vidhya and Priya (2023) proposed the use of deep learning and machine learning for the classification of diseases of the banana leaf. The banana diseases addressed by this study are Sigatoka and Leafspot. The models proposed were SVM, KNN, and deep learning using Alexnet. The study used a dataset of colored images of diseased and healthy leaves of banana with or without a background. Data augmentation was performed on the image dataset. The study yielded accuracies for testing of 84.86% for SVM, 76.49% for KNN, and 96.73% for Alexnet.

Similarly, Ramadhani (2017) developed a solution for predicting the presence of banana diseases and informing farmers so that they can prepare and be able to manage these diseases once they occur. The study used weather station data and an intelligent prediction algorithm (prediction based on weather conditions' factors or features) in a mobile application to predict the occurrence of diseases and provide farmers with early warning so that the diseases could be managed. However, the developed approach was only limited to the area whose data were collected for prediction, in this case, the Arumeru District in the Arusha Region. Furthermore, the prediction accuracy was also limited by the accuracy and reliability of the data obtained from the weather stations.

Moreover, Sanga *et al.* (2020) suggested early banana disease detection using a mobile application. The mobile application classifies the diseases using a deep learning model. The CNN architectures Inceptionv3 and Resnet152 were used. Inceptionv3 had an accuracy of 95.41% while Resnet152 had an accuracy of 99.2%. There were 3000 images of banana leaves

of three classes in the dataset: Fusarium Wilt infected leaves healthy leaves, and Black Sigatoka infected leaves. The study had good performance for the developed models but it does not localize the areas in the banana leaf or stalk image that are affected by the diseases.

2.6.3 Mobile Applications that deploy Deep Learning models for Plant Disease Detection

Hui *et al.* (2021) proposed a mobile application that used a deep learning object detection model to detect grape diseases. The mobile application in this study used Faster R-CNN based on the Inception-v2 for efficient detection. The results showed that the application yielded an accuracy of 97.9% when tested on grape disease images while running on the device without a server connection.

Nirmal *et al.* (2022) suggested a smart app that is farmer-friendly for pomegranate disease detection. The study's goal was to use leaf images to automate the disease detection system. The study's dataset was built using Mendeley data and included images of healthy and diseased pomegranate leaves. The study process included image data collection, image pre-processing, classification, and deployment. The deep learning models used for classification were AlexNet and VGG-16. The study showed that AlexNet was more efficient in detecting pomegranate leaf disease, and it was therefore deployed in a mobile application. The mobile application would help farmers detect pomegranate disease without the assistance of specialists.

Loyani and Machuve (2021) suggested a mobile application that employs deep learning to segment *Tuta absoluta*'s effect on tomatoes. This study deployed a segmentation model that was trained on a dataset of images of tomato leaves in a mobile application. The mobile application is used for the early and real-time detection of tuta pests in the early stages of the growth of tomatoes. With 70% minimum confidence and a 5-second time frame, the application was able to identify and segment *tuta absoluta*-infected patches on tomato leaves.

2.7 Research Gap

Some of the related works used a small number of images to train their deep learning models and to the best of the author's knowledge, none of the literature reviewed addressed the segmentation of Black Sigatoka and Fusarium Wilt banana diseases. This study uses an image segmentation technique with a large dataset. The dataset includes banana leaf and stalk images categorized into healthy banana leaves, banana leaves infected by Black Sigatoka disease, and

banana leaf and stalk images infected by Fusarium Wilt disease collected from the field. The dataset was used to develop a deep learning model for the early identification of Fusarium wilt and Black Sigatoka diseases. The advantage of applying deep learning techniques to identify diseases in plants as compared to other approaches, such as the one proposed by Ramadhani (2017), is that a large number of images of plant leaves infected by the diseases are used to train the deep learning models. The model learns from the disease symptoms features shown from the image of the leaf. As long as the disease symptoms are the same or similar (which is the case most of the time), the model detects a disease from a new image with very high accuracy without considering the place in which the image of the plant leaf was taken.

The image segmentation technique has many advantages over other deep learning techniques, such as classification methods. This technique can localize the infected area on the plant leaf image by creating a mask around it. Through this localization, the image segmentation technique shows the precise place on the leaf that is infected. If the image has multiple objects, image segmentation can describe each object in the image, while classification can only describe the whole image as one object of interest.

The best deep learning model was deployed in a mobile application to enhance its use by farmers. The farmers will use the application to identify Fusarium Wilt and Black Sigatoka diseases in the early stages. In addition, the mobile application also provides recommendations for mitigating these diseases recommended by researchers so that farmers can be aware of them and prevent the further spreading of the disease and rescue their yields.

CHAPTER THREE

MATERIALS AND METHODS

3.1 Study Area and Scope of the Research

The study area and scope of this research consist of the areas where the data used in this research was collected. Images of both diseased and healthy banana plants were collected in the Arusha, Kilimanjaro, Kagera, Mbeya, and Dar es Salaam regions of Tanzania. The selection of these areas was based on banana availability and disease prevalence.

3.2 Data Collection

A dataset of 30 640 banana leaf and stalk images was collected from the fields. Data was gathered using the Open Data Kit (ODK) tool called Adsurv, which was installed on a smartphone. Banana leaves and stalks were captured using a smartphone camera. A Samsung SM-A715F/DS phone camera was used to collect the dataset with normal settings. The data collection exercise involved farmers, researchers, agricultural experts, and plant pathologists. The dataset was collected on banana farms with healthy banana plants, some banana plants affected by Black Sigatoka disease, and some banana plants affected by Fusarium Wilt disease. To train the model with images of different qualities, images of various resolutions were obtained. The model was intended to be used in the field, where it is anticipated that smallholder farmers will use inexpensive phones with low quality, the model was trained using both low and high-resolution photos. The images were collected by taking a picture close to the banana leaf as seen in Fig. 37. To support further research in detecting and segmenting Fusarium Wilt and Black Sigatoka banana diseases, the dataset used in this work is freely available in an open-access repository, and more information about the dataset is reported by Mduma and Leo (2023).



Figure 37: Data collection in the field

The dataset had three classes: images of healthy banana leaves, images of Black Sigatoka infected banana leaves, and images of Fusarium Wilt infected banana leaves and stalks as summarized in Table 1. Figure 38 shows sample images of a healthy banana leaf, a Black Sigatoka infected banana leaf, and a Fusarium Wilt infected banana leaf and stalk.



Healthy

Black Sigatoka

Fusarium Wilt

Figure 38: Examples of images from the dataset

Table 1: Total number of data collected

Banana Images	Number of images collected
Healthy Leaves	9779
Black Sigatoka infected leaves	10 137
Fusarium Wilt infected leaves and stalks	10 724
Total	30 640

3.3 Data Preprocessing

Data pre-processing is a crucial step that helps a deep learning model learn and extract features from an image during model training. Data pre-processing in this work included cropping, data cleaning, renaming, and data annotation.

3.3.1 Data Cleaning and Cropping

The image dataset was manually cropped to remove the background and unwanted items and focus on the banana leaf or stalk. Removing duplicates was done to clean the banana images dataset. VisiPics and Duplicate Photo Finder are free software programs that were used to remove 3280 duplicates from the images as summarized in Table 2. This software program was used because it is open-source and easy to use. In VisiPics, strictly similar (or identical) images were removed as seen in Fig. 39. In Duplicate Photo Finder, images were searched against the “same picture” filter.

Table 2: A summary of removing duplicates

Banana Images	Before Removing Duplicates	After Removing Duplicates	Duplicates found and deleted
Healthy Leaves	9779	9120	659
Black Sigatoka infected leaves	10 137	9120	1017
Fusarium Wilt infected leaves and stalks	10 724	9120	1604
Total	30 640	27 360	3280

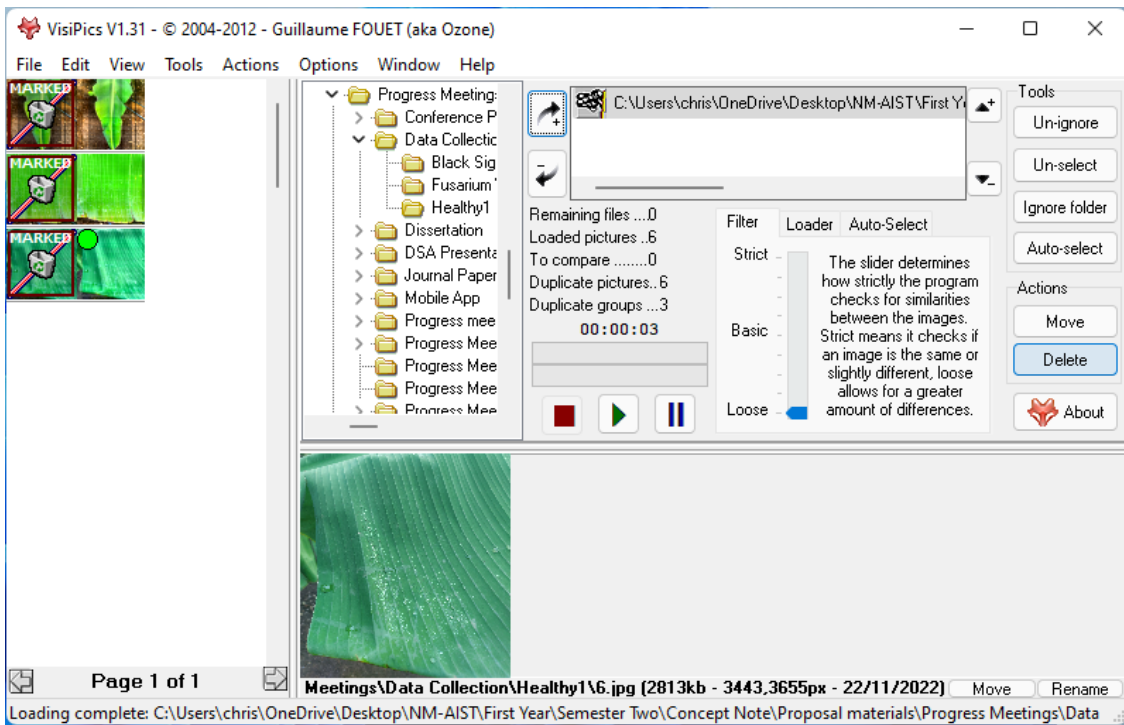


Figure 39: How VisiPics was used to detect and delete duplicates

3.3.2 Data Renaming

For simplicity, the clean images in each class were renamed to comprise image numbers. For example, for healthy images, the first one was HEALTHY_1.jpg, and the numbers kept on increasing to HEALTHY_9120.jpg. Bulk Rename Utility software, was used to rename images for all classes as shown in Fig. 40. This tool was used because it is open-source and simple to use.

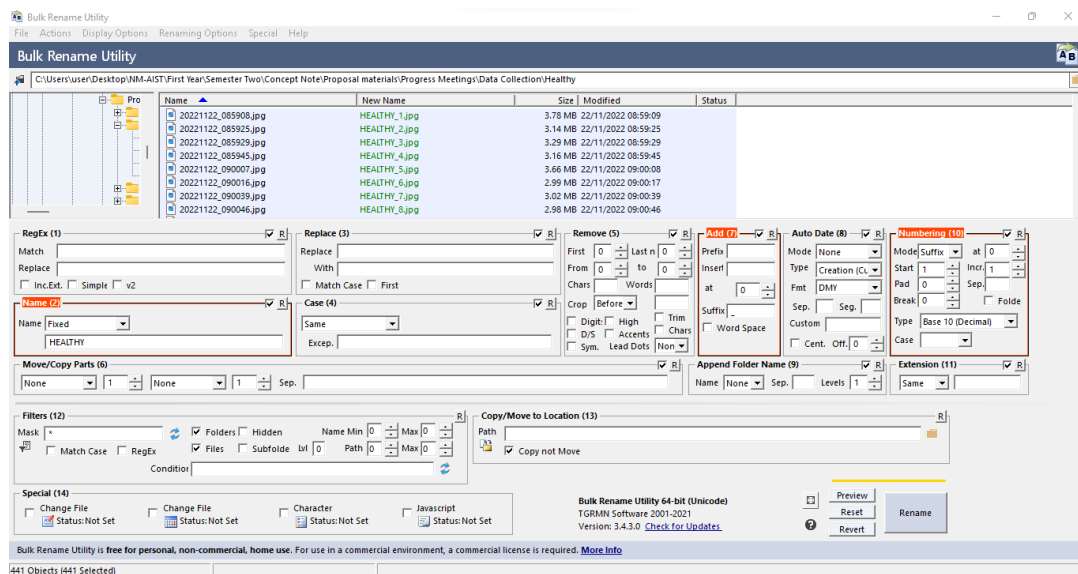


Figure 40: How Bulk Rename Utility was used to rename images in the dataset

3.3.3 Data Annotation

Image segmentation algorithms require images to have masks around the regions of interest with labels in training and validation to obtain accurate predictions of these regions of interest. The 27 360 images for all classes were annotated for the image segmentation task. LabelMe, which is open-source software, was used to annotate the images that were used in instance and semantic segmentation by Mask R-CNN and U-Net models, respectively. The specific procedure was to manually draw a mask around the regions of interest in the banana plant image using irregular polygons and then label them with the class name. For example, for banana leaf images affected by Black Sigatoka disease, an irregular polygon was drawn around each spot showing the damage of the disease on the leaf, and each polygon was given the label "Black Sigatoka". The irregular polygons drawn around the spots that showed the damage of Fusarium Wilt disease on banana leaves and stalks were given the label "fusarium wilt".

For the healthy banana leaves, an irregular polygon was drawn around the whole leaf area in the image, and the polygons were given the label "healthy". During annotation, the dataset had at least one irregular polygon for each image. Each image file had its corresponding annotation file in the same folder with the same name except for the extension. LabelMe saves its annotations in JSON format which were converted into PNG format annotations used in the U-Net model for semantic segmentation. Figure 41 shows how the dataset was annotated using LabelMe software. The outputs of the image annotation process are illustrated in Fig. 42.

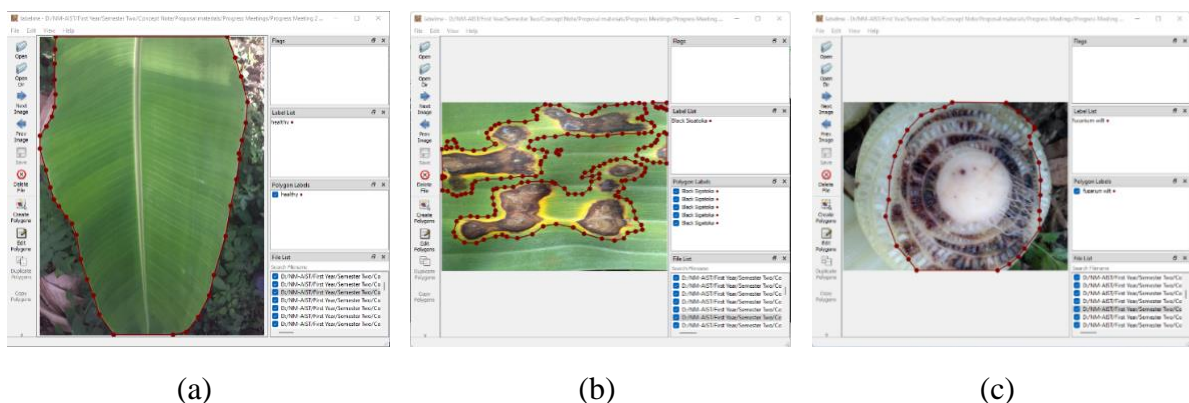


Figure 41: How LabelMe was used to manually annotate images (a) Annotation of a healthy banana leaf image; (b) Annotation of a banana leaf image affected by Black Sigatoka disease; and (c) Annotation of a banana stalk image affected by Fusarium Wilt disease

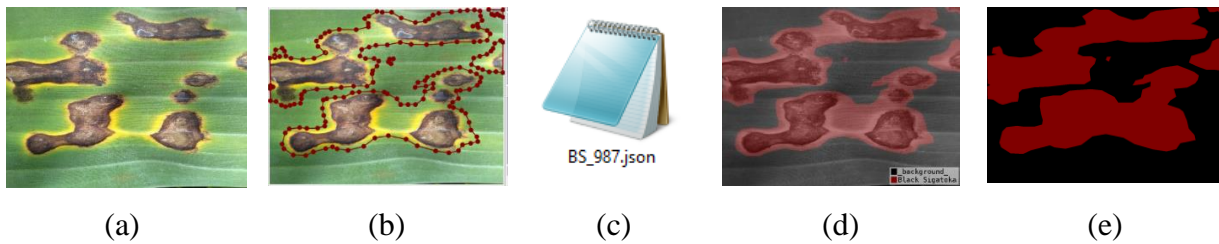


Figure 42: The Image annotation outputs (a) Original image (b) Drawing a polygon around the area on a leaf image affected by Black Sigatoka (c) Saving the annotation in JSON format (d) Visualizations of the labels (e) Extraction of the mask in PNG format

3.4 Research Framework

The research framework in Fig. 43 provides a comprehensive explanation of how this study was conducted. A dataset comprising images of banana leaves and stalks was collected from the field. This was followed by image preprocessing, then model development and validation. For instance, segmentation, the Mask R-CNN model was created, and for semantic segmentation, the U-Net model was. The mobile application deployed the best-optimized model. The mobile application was validated in the field to see how well it worked. The mobile application will be used by farmers and agricultural experts for the early identification of banana diseases. The application also provides farmers with information concerning banana diseases, and after disease detection, the application recommends to farmers initiatives to undertake (like fungicides to use) to rescue their yields.

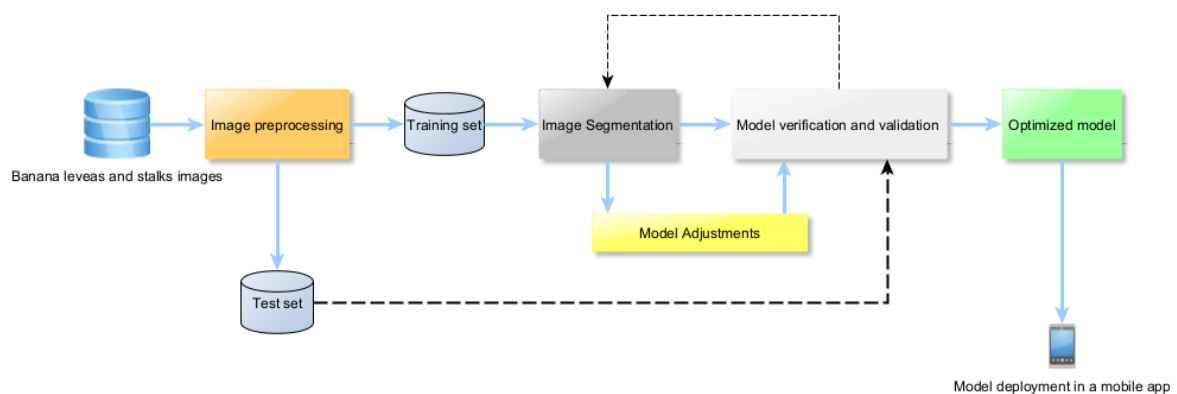


Figure 43: The research framework

3.5 Classification with Convolutional Neural Network Model

A basic CNN model was incorporated to classify banana plants and their diseases because the CNN architecture is used as the foundation of the Mask R-CNN and the U-Net models. The CNN model was built from scratch without using transfer learning.

3.5.1 Convolutional Neural Network Model Hyper-Parameters Tuning

In implementing the CNN model, a sequential model was used. The model contained four convolutional layers, each followed by a max pooling layer, then followed by a dropout layer with a rate of 0.2. The first and second convolutional layers had 16 and 32 filters while the third and fourth each had 64 filters. Each convolutional layer used the ReLu activation function, and all convolutional filters were 3×3 in size. These were followed by a flattened layer, which was followed by a dense layer with 512 neurons with the ReLu activation function. The output dense layer had four neurons, which was the number of our classes, and had a softmax activation function. Images were rescaled to normalize them by 1/255 to a range of zero to one and resized to 512x512 pixels. The CNN model used several hyperparameters as summarized in Table 3.

Table 3: The CNN model training hyperparameters

Parameters	Value(s)
Batch size	32
Epoch	100
Optimizer	Adam (Learning rate = 0.001)
Loss	Categorical Crossentropy
Metric	Accuracy, Precision, Recall, and F-measure

3.5.2 The CNN Model Classes and Data Grouping During Training

The CNN model had four classes, which are "Black Sigatoka", "Fusarium Wilt", "Healthy", and an extra class called "Not Banana". The extra class comprised images of other things apart from a banana leaf or stalk. This extra class of "Not Banana" was included to enable the CNN model to predict images of other things. Without this extra class, if, for example, a picture of the sky was introduced to the CNN model to be predicted, it would be predicted as either Black Sigatoka, Fusarium Wilt, or Healthy. But when the extra class is introduced to the CNN model

during training, the model will predict an image of the sky as "Not Banana". The dataset for this extra class was collected from the internet. The model was trained in groups, where the output weights that were used to train the first group were used as input for training the second group, and so on.

The CNN model was trained in groups because of the large dataset. The entire dataset was divided into five groups, where the first four groups had 2000 images for each of the Black Sigatoka, Fusarium Wilt, and Healthy classes and 407 images for the Not Banana class. The fifth group had 1120 images for each of the Black Sigatoka, Fusarium Wilt, and Healthy classes and 407 images for the Not Banana class. Eighty percent of the images in each group were used for training and 20% for validation. Table 4 shows the data distribution for the CNN model. Appendix 5 shows the CNN model source code.

Table 4: Data distribution for the CNN model

Group	Banana Images	Training Set	Validation Set	Total Images
Group 1	Healthy banana leaves	1600	400	6407
	Black Sigatoka infected banana leaves	1600	400	
	Fusarium Wilt infected banana leaves and stalks	1600	400	
	Not banana leaves or stalk	326	81	
	Group 1 Total	5126	1218	
Group 2	Healthy banana leaves	1600	400	6407
	Black Sigatoka infected banana leaves	1600	400	
	Fusarium Wilt infected banana leaves and stalks	1600	400	
	Not banana leaves or stalk	326	81	
	Group 2 Total	5126	1218	
Group 3	Healthy banana leaves	1600	400	6407
	Black Sigatoka infected banana leaves	1600	400	
	Fusarium Wilt infected banana leaves and stalks	1600	400	
	Not banana leaves or stalk	326	81	
	Group 3 Total	5126	1218	
Group 4	Healthy banana leaves	1600	400	6407
	Black Sigatoka infected banana leaves	1600	400	
	Fusarium Wilt infected banana leaves and stalks	1600	400	
	Not banana leaves or stalk	326	81	
	Group 4 Total	5126	1218	
Group 5	Healthy banana leaves	896	224	3767
	Black Sigatoka infected banana leaves	896	224	
	Fusarium Wilt infected banana leaves and stalks	896	224	
	Not banana leaves or stalk	326	81	
	Group 5 Total	3014	753	
Total		23 518	5877	29 395

3.6 Transfer Learning

Transfer learning is a method where a model created for one job is utilized as a foundation for another model created for a similar but distinct activity. In the Mask R-CNN model, transfer learning is applied through the latest Mask R-CNN trained weights from the COCO dataset. U-Net was trained and yielded the best performance in several International Symposium on Biomedical Imaging (ISBI) challenges (Ronneberger *et al.*, 2015) in semantic segmentation.

3.7 Mask Region-Based Convolutional Neural Network Model Hyper-Parameter Tuning

The Mask R-CNN model used several hyperparameters as summarized in Table 5, for feature extraction and model training. Appendix 3 shows the Mask R-CNN model source code.

Table 5: Mask R-CNN model training hyperparameters

Parameters	Value(s)
Backbone	ResNet50 or ResNet101
Backbone Strides	[4, 8, 16, 32, 64]
Batch Size	1
Detection Maximum Instances	100
Detection Minimum Confidence	0.7
Detection NMS Threshold	0.3
GPU Count	1
Images per GPU	1
Image Maximum Dimension	896
Image Minimum Dimension	896
Image Resize Mode	square
Image Shape	[896 896 3]
Learning Momentum	0.9
Learning Rate	0.001
Loss Weights	{'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
Mask Shape	[28, 28]
Number of Classes	4
RPN Anchor Scales	(8, 16, 64, 128, 256)
RPN Anchor Stride	1
RPN NMS Threshold	0.7
Steps Per Epoch	150
Validation Steps	30
Weight decay	0.0001
Epoch	50

3.8 The U-Net Model

3.8.1 The U-Net Model Hyper-Parameter Tuning

A custom U-Net model was used whereby its initial layer had 32 convolutional filters. Every layer in the contraction path resulted in a doubling of this number of filters. The number of layers in the contraction path was set to 4. The images were scaled to a range of zero to one and resized to 512×512 pixels. The training dataset size was increased by using data augmentation. The data argumentation techniques applied included a rotation of 5.0 degrees, a height and width shift range of 0.05, a shear range of 40, a zoom range of 0.2, vertical and horizontal flipping, and a fill mode of constant. Data augmentation was applied equally to both images and their annotations. The U-Net model experiment used 200 epochs, an SGD activation function, an IoU threshold for a minimum detection probability of 0.5, and a learning rate of 0.01.

3.8.2 The U-Net Model Classes and Data Grouping During Training

In training the U-Net model, two classes were used, which are Black Sigatoka and Fusarium Wilt images. The U-Net model used image masks which were in PNG format. Therefore, each JPG image had its own corresponding PNG mask. The U-Net model was trained in groups because of the large dataset and very high computing time, especially when resizing and converting each image and its mask into NumPy array values. When we tried to train the model with the entire dataset the process of converting the images and their corresponding masks into NumPy arrays (so that they could be used to train the model) was not reaching an end (because of the huge number of images) hence we could not reach the stage of training the model. This caused the dataset in the U-Net model to be divided into 25 groups to handle a small number of images at a time. The first 12 had 500 images together with their corresponding masks for each of Black Sigatoka and Fusarium Wilt. Groups 13 to 24 had 250 images together with their corresponding masks for each Black Sigatoka and Fusarium Wilt. Group 25 had 120 images together with their corresponding masks for each of Black Sigatoka and Fusarium Wilt. Eighty percent of the images in each group were used for training and 20% for validation in a random split.

The output trained weight from the first group was used as the input weight for training the second group and so on. Unfortunately, due to the long time required during the annotation process, an extra class could not be added to be able to segment other plants or things that are

not images of the banana plant. Table 6 shows the data distribution for each group for the U-NET model. Appendix 4 shows the U-Net model source code.

Table 6: Data distribution for the U-NET model

Group	Banana Images	Number of Images	Total Images	Training set	Validation set
Group 1	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 2	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 3	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 4	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 5	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 6	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 7	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 8	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 9	Black Sigatoka infected leaves	500	1000	800	200

	Fusarium Wilt infected leaves and stalks	500			
Group	Banana Images	Number of Images	Total Images	Training set	Validation set
Group 10	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 11	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 12	Black Sigatoka infected leaves	500	1000	800	200
	Fusarium Wilt infected leaves and stalks	500			
Group 13	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 14	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 15	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 16	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 17	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 18	Black Sigatoka infected leaves	250	500	400	100

	Fusarium Wilt infected leaves and stalks	250			
Group	Banana Images	Number of Images	Total Images	Training set	Validation set
Group 19	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 20	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 21	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 22	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 23	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 24	Black Sigatoka infected leaves	250	500	400	100
	Fusarium Wilt infected leaves and stalks	250			
Group 25	Black Sigatoka infected leaves	120	240	192	48
	Fusarium Wilt infected leaves and stalks	120			
Total		18 240	18 240	14 592	3648

3.9 Experiment Setting

The experiments were done on a PC with Windows 11 Pro and one Intel(R) Core (TM) i5-8250U CPU @ 1.60GHz 1.80 GHz, with 8GB of RAM. The notebook was run on Google Colab Pro Plus with a Tesla T4 GPU and 54.8GB of RAM. Python 3 and the TensorFlow backend were used to implement the network.

3.10 Evaluation

Deep learning models are assessed by evaluating how successfully the learned model generalizes to previously unexplored data. A deep learning model's performance is measured using a variety of evaluation metrics. Different deep-learning tasks, including classification, localization, and others, are evaluated using different evaluation metrics. The CNN model's performance was evaluated using accuracy, f-measure, recall, and precision. Mean Average Precision (mAP), Dice Coefficient, and Intersection over Union were used to assess the effectiveness of the instance (Mask R-CNN) and semantic (U-Net) segmentation models. The following definitions apply to these evaluation metrics:

3.10.1 Accuracy

One of the evaluation metrics for classification model evaluation is accuracy. According to Equation 1, accuracy is calculated as the number of accurate classification predictions divided by the total number of predictions:

$$Accuracy = \frac{\text{correct predictions}}{\text{correct predictions} + \text{incorrect predictions}} \quad (1)$$

3.10.2 Recall

The question that recalls answers is: What proportion of actual positives was identified correctly? Recall tells the percentage of predictions the model correctly identified as the positive class when ground truth was the positive class. A model has a recall of 1.0 if it does not have false negatives. Recall can be defined as seen in Equation 2:

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

Where:

TP (True Positives): is the number of positive samples correctly predicted.

FN (false Negatives) is the number of positive samples that are wrongly predicted as negative.

3.10.3 Precision

The question that precision answers is: What proportion of positive identifications were actually correct? Precision tells the percentage of correct predictions when the model predicts

a positive class. A model has a precision of 1.0 if it does not have false positives. Precision can be defined as seen in Equation 3.

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

Where:

TP (True Positives): the number of positive samples correctly predicted.

FP (false Positives): the number of negative samples that are wrongly predicted as positive.

3.10.4 The F-Measure

The computation of the harmonic mean of recall and precision, assigning equal weights to each, is called the F-measure. The F-measure gives the best precision and recall at the same time. This allows for the accounting of both precision and recall in a single score, allowing for the comparison of models and the description of a model's performance. The F-measure can be calculated using the formula in Equation 4.

$$F - Measure = 2 * \frac{precision*recall}{precision+recall} \quad (4)$$

3.10.5 Mean Average Precision (mAP)

The average precisions for every class over all classes are given by the Mean Average Precision (mAP). It is used as the key evaluation metric to assess how well the model segmented the data. Mean Average Precision (mAP) is computed by the formula in Equation 5.

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (5)$$

Where:

mAP is the mean Average Precision of all classes.

AP_i is the Average Precision.

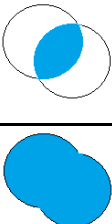
$\sum_{i=1}^N AP_i$ is the sum of Average Precision values.

N is the number of all classes.

Average Precision is the area under the Precision-Recall (PR) curve. The Precision-Recall curve depicts the tradeoff between precision and recall for various thresholds. While high recall is associated with a low false negative rate, high precision is associated with a low false positive rate. High recall and precision are both indicated by a high area under the curve. High scores for both show that the classifier is generating accurate (high precision) and largely positive (high recall) results. In the Precision-Recall (PR) curve, precision is the y-axis and recall are the x-axis. Setting an IoU threshold value yields several precision-recall value pairs, which are then used to plot the graph. Any detection that has an IoU value below the predetermined threshold is classified as a false positive or true positive otherwise. The precision-recall graph is created by computing the precision and recall at each detection, then sorting the results by the threshold and traversing through all precision-recall value pairings.

3.10.6 Intersection Over Union

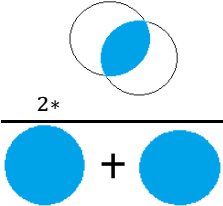
A number called Intersection Over Union (IoU) evaluates how much two boxes or masks overlap. In the context of object detection and image segmentation, IoU evaluates the overlap between the Ground Truth and Prediction region. In image segmentation, IoU is the main metric that evaluates the accuracy of a model. The IoU is the overlapping area (the point where the predicted mask and the ground truth mask meet) over the union area (where the predicted mask and the ground truth mask are joined). When the IoU exceeds a predetermined threshold, the prediction is said to be True Positive (TP), and when it falls short of that threshold, it is said to be False Positive (FP). The Intersection over Union (IoU) is given by the formula in Equation 6.

$$IoU = \frac{\text{Overlapping Area}}{\text{Union Area}} \quad (6)$$


Intersection over Union (IoU) varies in the range of zero to one, with zero signifying that between the masks there is no overlap and one signifying that between the masks there is perfect overlap, i.e., a perfect prediction.

3.10.7 Dice Coefficient

The Dice coefficient is a spatial overlap index and a reproducibility validation metric. The overlap between the predicted mask and the actual mask is measured by the dice coefficient. Its value ranges from zero, which means no spatial overlap between ground truth and predicted mask, to one, which means complete overlap. The dice coefficient is 2 * the area of overlap divided by the total number of pixels in both images. The dice coefficient is defined as seen in Equation 7.

$$Dice\ Coefficient = \frac{2 * Overlapping\ Area}{Total\ number\ of\ pixels\ in\ both\ images} = \frac{2 * \text{Area of Overlap}}{\text{Area of Circle 1} + \text{Area of Circle 2}} \quad (7)$$


3.10.8 Loss Function

Loss is a measure of how poorly the model predicted a single example. Loss is a number given by a loss function. The goal of a model is to minimize the loss function. The loss value is one of the guides to inform us on whether to continue with hyperparameter tuning to improve the model's performance. A model with good performance will have a small loss number.

(i) Mask R-CNN Loss Function

Mask R-CNN combines different losses for each sampled region of interest into one multi-task loss. The loss used by Mask R-CNN sums up the losses from classification, bounding box, and mask prediction. The bounding box and classification losses used by Mask R-CNN originate from Faster R-CNN. The mask branch contains a Km^2 dimensional output for every region of interest that encodes K binary masks of resolution $m*m$, one for every K class. A per-pixel sigmoid is applied to this, and a mask loss is defined as the average binary cross-entropy loss. A per-pixel sigmoid and a binary loss allow the network to generate masks for each class without competition among classes. The Mask R-CNN loss is defined as seen in Equation 8.

$$L = L_{cls} + L_{box} + L_{mask} \quad (8)$$

Where:

L_{cls} is the classification loss.

L_{box} is the bounding-box loss.

L_{mask} is the mask loss.

(ii) The U-Net Loss Function

Ronneberger *et al.* (2015) state that “The energy function is computed by a pixel-wise softmax over the final feature map combined with the cross-entropy loss function.” In other words, the cross-entropy loss function is applied after pixel-by-pixel softmax on the U-Net's output feature map. As a result, the segmentation problem is transformed into a multiclass classification problem and each pixel is given to a certain class. When using a weighted loss, U-Net gives background labels that separate touching items a lot of weight. The loss weighting scheme helps U-Net distinguish touching objects of the same class. Hence, U-Net can separate individual spots of Black Sigatoka within a binary segmentation map.

3.11 Model Deployment

Model deployment is the procedure of putting the deep learning model into use so that it may be used to make predictions or find patterns using data. A deep learning model can be deployed in an embedded Internet of Things system, a web application, or a mobile application. The image segmentation model was deployed in a mobile application so that it could be accessed and used by agricultural extension officers and farmers to detect banana diseases at an early stage. Also, the developed mobile application provides recommendations to users so that they can be aware of what measures to be taken to mitigate the situation.

Deep learning models are resource-intensive. They require a lot of memory and storage capacity to run. To meet the resource limitations in a mobile phone environment, the TensorFlow Lite framework was utilized to transform the model into a mobile readable format. TensorFlow Lite is a collection of tools created to suit all the requirements of embedded and mobile systems for deployment. Some of the constraints addressed by TensorFlow Lite are battery consumption, low latency, lightness, and an efficient model format. TensorFlow Lite is not designed to be a framework for training models; the model is trained in TensorFlow instead. A TensorFlow model was converted into TensorFlow Lite format by a converter that shrinks and optimizes the model, which was then loaded and run using a TensorFlow Lite interpreter. Figure 44 illustrates the TensorFlow Lite suite.

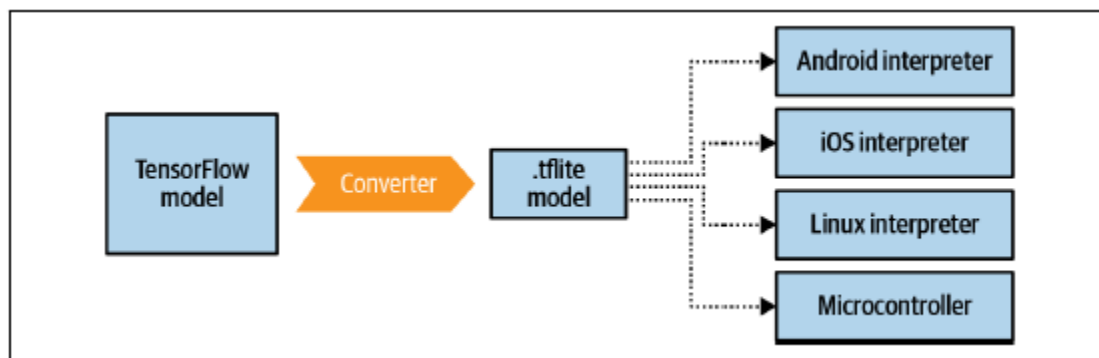


Figure 44: The TensorFlow Lite suite (Moroney, 2021)

The other option was to deploy a normal TensorFlow model on a web server and create an Application Programming Interface (API) to be used to access the model from the mobile application through a Uniform Resource Locator (URL).

The CNN deep learning model was deployed in a mobile application after being translated to TensorFlow Lite format. However, the use of a compressed TensorFlow Lite format model proved to be less efficient because the ability of the model to detect diseased and healthy banana leaves and stalks was reduced tremendously. The original TensorFlow model could detect or predict more accurately than when it was converted to TensorFlow Lite. As a result of this, the study opted to deploy the original TensorFlow model on a web server and create an API to access this model from the mobile application. Appendix 6 shows the Model deployment Flutter Source code.

3.11.1 Requirements Elicitation and Analysis

This section discusses the services that will be provided by the mobile application. To achieve the specific objective of developing an image segmentation deep learning model for the early detection of banana plant diseases, the requirement was to collect images of banana leaves that were healthy, Black Sigatoka infected banana leaves, and Fusarium Wilt infected banana leaves and stalks. The requirements for the development of the mobile application were obtained by interviewing the stakeholders, including prospective users, which are farmers, agricultural extension officers, and other agricultural and technical experts. Appendix 1 shows the questions that were asked to these people. Six people were involved in this process.

(i) Functional Requirements

The services offered by the mobile application are included in the Functional requirements.

They include the relationship between input and output. Table 7 describes the functional requirements that were identified for the developed mobile application.

Table 7: The mobile application’s functional requirements and their description

Functional Requirement ID	Functional Requirement Name	Functional Requirement Description
1	Capture an image	The application should allow farmers and extension officers to take a picture of a banana plant through the mobile phone camera.
2	Upload an image	The application should allow farmers and extension officers to upload an image of a banana plant from the phone’s gallery.
3	Display image	The application should allow farmers and extension officers to display the captured or uploaded image.
4	CNN model runs inference	The application should deploy an image classification deep learning model and allow farmers and extension officers to run inference in the background on the displayed image in the application to detect whether the banana leaf or stalk is healthy or is affected by either Fusarium Wilt or Black Sigatoka banana diseases.
5	View detection results	The application should allow farmers and extension officers to view detection results from the mobile application after detecting a disease.
6	View mitigation recommendations	The application should allow farmers and extension officers to view the mitigation recommendations against the banana diseases detected in the mobile application.
7	View banana information	The application should allow farmers and extension officers to view general information about banana farming, including: <ul style="list-style-type: none"> a) different types of bananas, b) the banana types that provide the highest yields, c) the banana types that have high demand in the market, and d) the best practices in banana farming.

Functional Requirement ID	Functional Requirement Name	Functional Requirement Description
8	View disease information	The application should allow farmers and extension officers to view general information about Fusarium Wilt and Black Sigatoka banana diseases. This includes: <ul style="list-style-type: none"> a) their causes, b) symptoms, c) transmission mechanism, and d) mitigation strategy. <ul style="list-style-type: none"> a. Including how to eradicate sick banana plants without spreading the disease further.
9	Change language	The application should include English and Swahili languages and allow farmers and extension officers to choose which language they prefer for the application.
10	Update/Patch the application	The application should receive its updates and patches from a web-based backend system that will allow the administrator to manage it.

(ii) Non-functional Requirements

Non-functional requirements outline the characteristics or standards used to evaluate the system's performance. How efficiently the system operates can enhance the system's functionality. Table 8 describes the non-functional requirements that were identified for the mobile application.

Table 8: Non-functional requirements for the mobile application and their description

Non-functional Requirement ID	Non-functional Requirement Name	Non-functional Requirement Description
1	Availability	The application and the deep learning model should be available all the time for the farmers to use.
2	Reliability	The application should be able to accurately detect healthy banana plants and banana plants affected by either Fusarium Wilt or Black Sigatoka diseases, given an image of a banana leaf or stalk.
3	Performance	The application should have low latency in performing inference and displaying detection results.
4	Usability	The application should be intuitive and easy to use without any need for guidance.
5	Compatibility	The application should be accessible to mobile phones running on Android operating systems.

3.11.2 System Design

System design defines the system elements, including modules, architecture, system components, and their interfaces, as well as their data. Several design models for the recommended mobile application are included. These are the use case diagram, activity diagram, and sequence diagram.

(i) Use Case Diagram

A use-case diagram shows the activities that are accomplished by the users of the system. It comprises the use cases or discrete tasks, the actors or users of the system, and the relationship between them. Figure 45 shows the use case diagram for the banana disease detection mobile application.

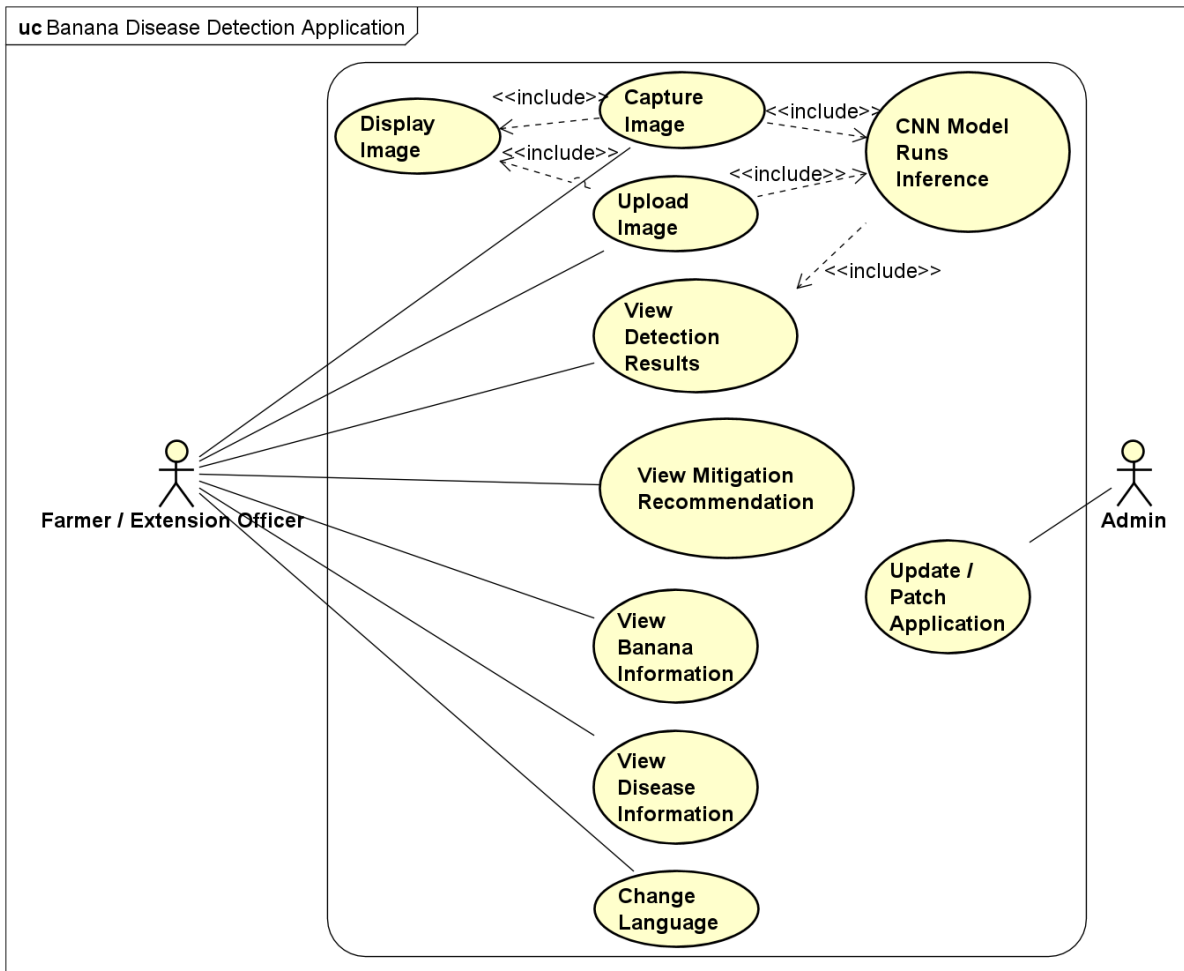


Figure 45: Use case diagram for the banana diseases detection mobile application

(ii) Activity Diagram

An activity diagram is used in business process modelling. It depicts the processes that a user of the mobile application can go through, from the beginning when the user launches the application to when the user achieves their goal and closes their application. Figure 46 illustrates the activity diagram for the mobile application for banana disease detection.

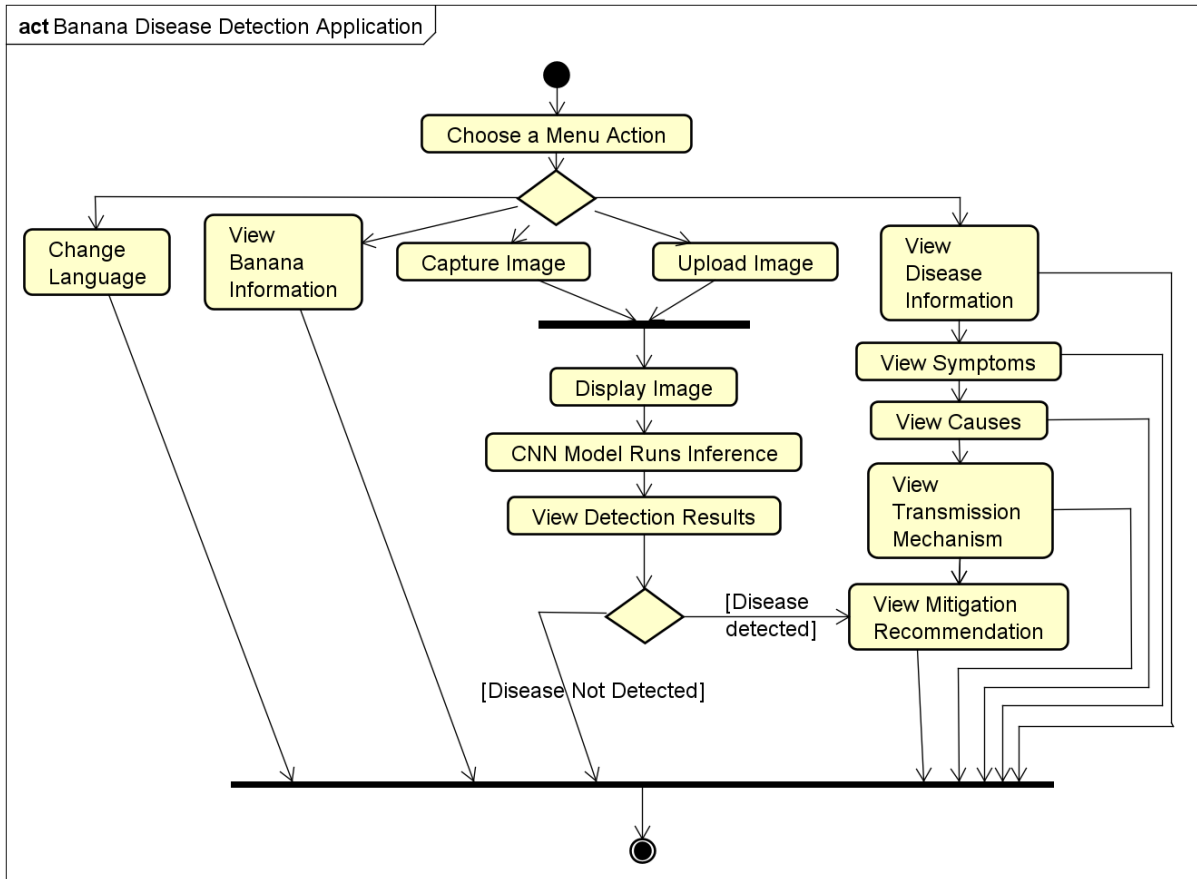


Figure 46: Activity diagram for the banana diseases detection mobile application

(iii) Sequence Diagram

The interaction between the actors and the objects in a system is modelled using a sequence diagram. It depicts the interactions happening in a particular use case or use case instance. Figure 47 illustrates a sequence diagram for the capture/upload image use case.

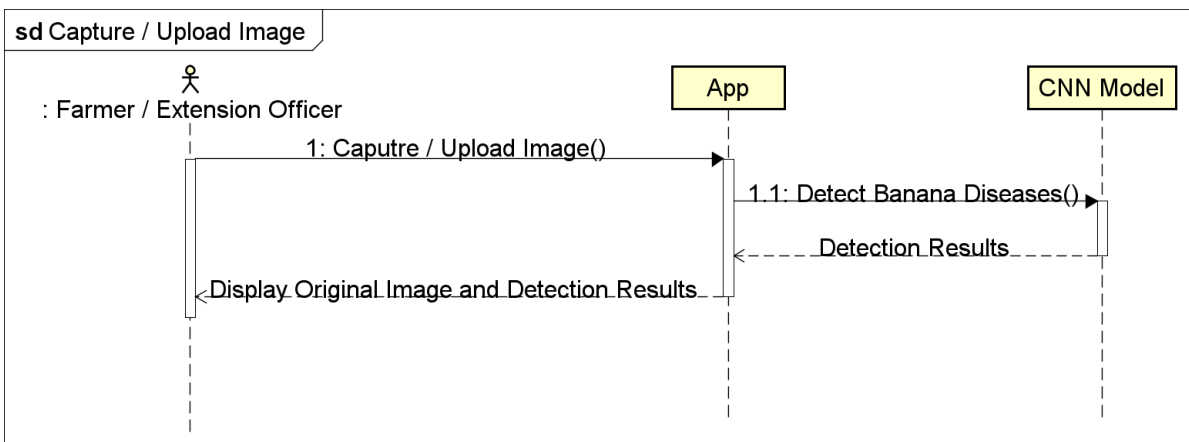


Figure 47: Sequence diagram for the capture/upload image use case

3.11.3 System Development Methodology

This study used the agile method using extreme programming (XP) for developing the mobile application. Agile methodology was adopted because it accelerates software delivery and enhances the ability to manage changing priorities. Extreme programming (XP) allows the development of software in small iterations, whereby each iteration can be tested against customer user stories, and user feedback in acceptance testing helps to determine the readiness of the software for release or needs more iterations.

3.11.4 Technologies Used

The Flutter framework which uses the Dart language was used in the Android Studio IDE to develop the mobile application. The image classification deep learning model was trained using TensorFlow in the Google Colab Pro computing environment. The conversion of the model into a lighter version (mobile phone-compatible) using TensorFlow Lite was done. A Samsung SM-A715F/DS was used in testing the mobile application. The Flask framework was used to develop an API to access the TensorFlow model from the mobile application.

3.12 Validation of the Performance of the Developed Mobile Application

Validation or user acceptance testing of the mobile application was done by using the requirements of the mobile application and testing if they were met. Questions were formulated about the requirements, and the users were required to either answer Yes or No. Appendix 2 presents the questionnaire that was used during system validation.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Feature Extraction Results

The features from the image were extracted by the CNN backbone architecture. Mask R-CNN utilizes ResNet101 and ResNet50 backbone architectures for feature extraction. The subsequent layers receive the extracted features as their input. Figure 48 presents the results of backbone feature maps from the dataset at different layers.

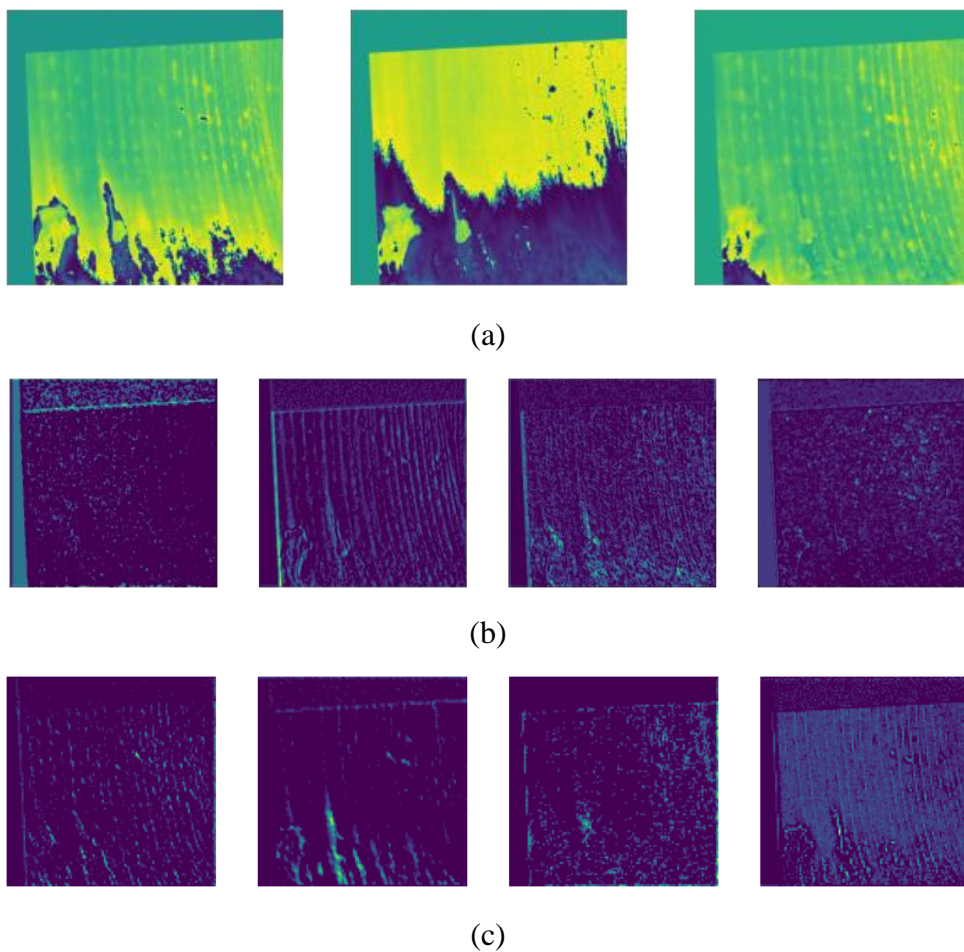


Figure 48: Example of backbone feature maps at the (a) input layer (b) res2c_out activation layer, and (c) res3c_out activation layer

Figure 49 presents the regions of interest (ROIs) and negative and positive anchors from the dataset. The regions of interest highlight the leaf area that is affected by the disease as the area of interest.

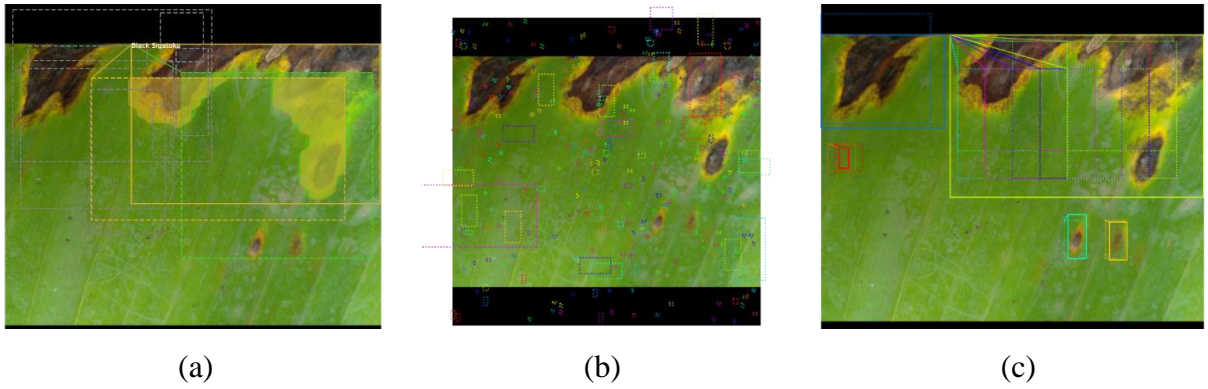


Figure 49: Examples of RPN anchors (a) Regions of Interest (ROIs), (b) Negative anchors, and (c) Positive anchors

From the U-Net model, Fig. 50 shows the original image (bottom row), the ground truth from the mask or annotation (middle row), and the mask overlaid on the original image (top row). The data augmentation is applied equally to the original image and its mask before training, as seen in Fig. 51.

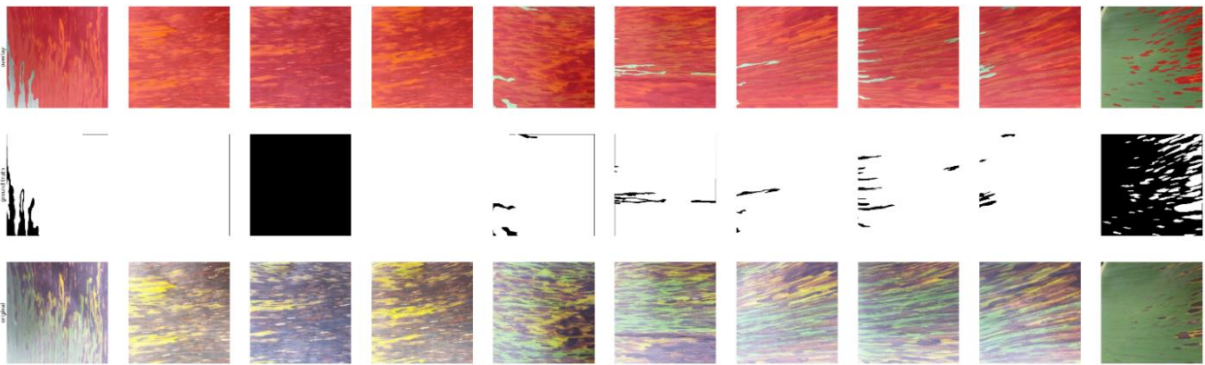


Figure 50: Original image, ground truth from the mask and the mask overlaid on the original image

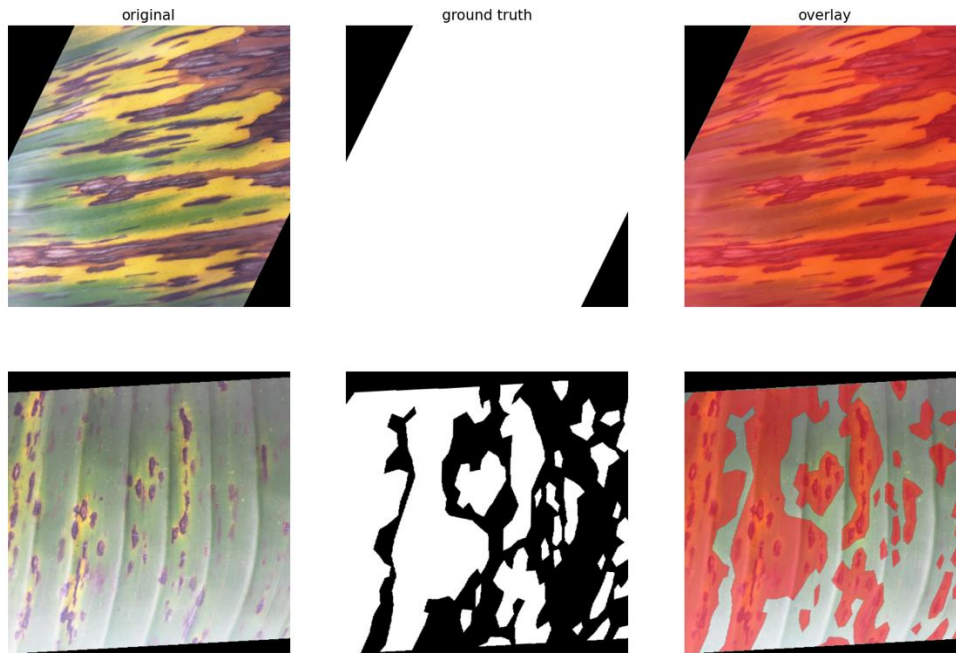


Figure 51: Data augmentation applied equally on the image and its mask

4.2 Model Development Results

4.2.1 Convolutional Neural Network Model Results

(i) General Results

The results of the CNN models trained in groups are given in Table 9. Results showed that the second group had the best overall performance with the highest validation accuracy, F-measure, recall, and precision, and the lowest validation loss. Figure 52 shows the graphs of the CNN model's performance for the second group. On the accuracy over epoch graph on the left of Fig. 52, results show that the validation accuracy rose rapidly to the 6th epoch, then remained steady around 90% with some fluctuations, hitting a maximum of 91.17%, while the training accuracy followed the same pattern without fluctuations, rising higher than the validation accuracy. This shows that the model generalized well. On the loss over epoch graph on the right of Fig. 52, results showed that the training loss decreases rapidly in the early epochs, and at epoch 21, it starts to decrease steadily until the end, while the validation loss decreases rapidly from the beginning, and from epoch 6, it started increasing steadily with fluctuations.

Table 9: The CNN model performance for detecting banana diseases

Model Group	Epoch	Validation loss	Precision (%)	Recall (%)	F-measure (%)	Validation Accuracy (%)
CNN Group 1	100	0.7621	65.75	65.81	65.70	65.86
CNN Group 2	100	0.2683	91.08	91.62	90.55	91.17
CNN Group 3	100	0.7367	78.52	78.12	78.12	78.59
CNN Group 4	100	0.6775	84.74	84.05	84.45	84.77
CNN Group 5	100	0.5418	83.28	84.00	82.57	83.49

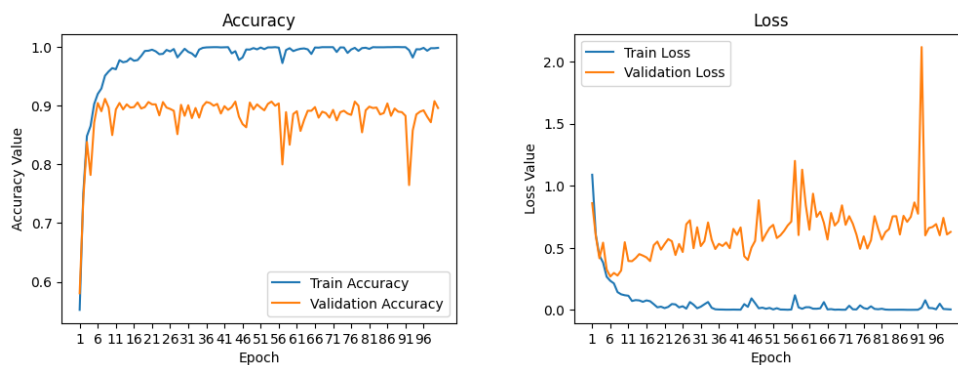


Figure 52: Performance for the CNN model

(ii) Training Time Results

Training time is a crucial measure of a model’s performance. Table 10 shows the training times for the CNN model experiments. The results show that from CNN group 2 to group 5, there is a significant time decrease compared to group 1. This is because CNN group 1 was trained with a larger image size, with a maximum of 700 KB per image, but in CNN groups 2 to 5, the images had a maximum of 100 KB per image. CNN group 5 had the least training time because it had fewer images compared to other groups, as seen in the data distribution for the CNN model in Table 4.

Table 10: CNN model training time

CNN Group	Training time (Seconds)
CNN Group 1	20 558.496
CNN Group 2	13 586.031
CNN Group 3	12 200.882
CNN Group 4	14 189.828
CNN Group 5	6983.726

4.2.2 Mask Region-Based Convolutional Neural Network Model Results

The Mask R-CNN model yielded a mean Average Precision (mAP) of 0.045 29. The Mask R-CNN model with ResNet101 had a training time of 1166.52 minutes (69 991.2 seconds). The complexity of the Mask R-CNN model's structure leads to a much longer training time when compared to the CNN model. The model was able to accurately segment Fusarium Wilt infected areas and Black Sigatoka infected areas with high confidence scores. Figure 53 illustrates examples of how the Mask R-CNN model predicted the segmentations of the Black Sigatoka infected banana leaf and the Fusarium Wilt infected banana leaf respectively.

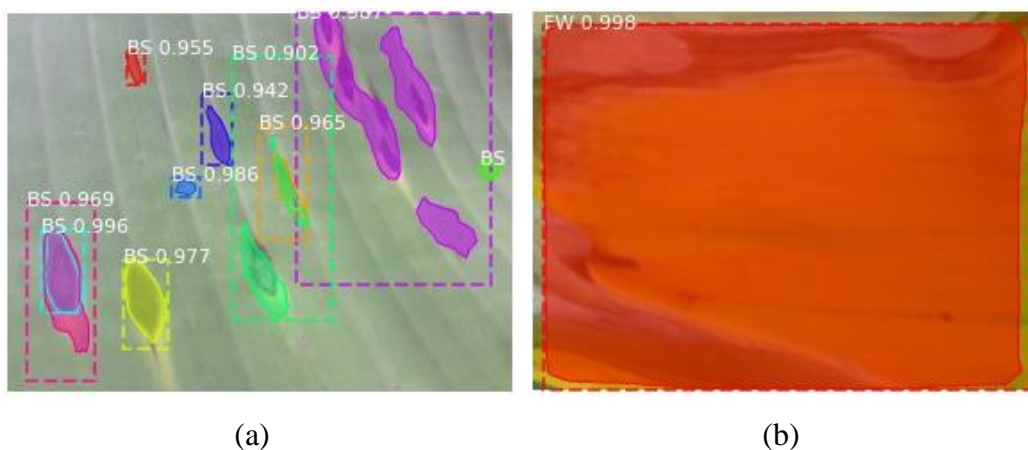


Figure 53: Examples of how the Mask R-CNN model predicts segmentations (a) Image segmentation of a leaf affected by Black Sigatoka disease: and (b) Image segmentation of a leaf affected by Fusarium Wilt disease

4.2.3 The U-Net Model Results

(i) The U-Net Model Loss Results

The loss results for all the model experiments done are shown in Table 11. The results show that U-Net group 8 has the least validation loss and training loss, while the other groups vary

slightly. Following every training epoch, estimates of the training and validation losses were made. Figure 54 shows the loss over epoch graph for the U-Net group 8 model with 100 epochs. The graph shows that the training loss has a decreasing trend during training with small fluctuations, while the validation loss also decreases but with milder fluctuations, hitting a minimum of 0.0583. This suggests that both early in the training process and later on, the U-Net model fits well on the features of our dataset. The U-Net group 8 model had the best performance because it obtained the lowest loss value when compared to other groups.

Table 11: Model loss results

Model Group	Training loss	Validation loss
U-Net group 1	0.0976	0.1151
U-Net group 2	0.1041	0.1025
U-Net group 3	0.1162	0.1161
U-Net group 4	0.0755	0.0744
U-Net group 5	0.0773	0.0743
U-Net group 6	0.0450	0.0729
U-Net group 7	0.0539	0.0735
U-Net group 8	0.0474	0.0583
U-Net group 9	0.1174	0.1331
U-Net group 10	0.1136	0.1410
U-Net group 11	0.1297	0.1893
U-Net group 12	0.0863	0.1336
U-Net group 13	0.1650	0.2297
U-Net group 14	0.1773	0.2304
U-Net group 15	0.0595	0.0886
U-Net group 16	0.1468	0.2372
U-Net group 17	0.1893	0.2574
U-Net group 18	0.1582	0.1956
U-Net group 19	0.1442	0.1793
U-Net group 20	0.1340	0.1696
U-Net group 21	0.1633	0.1814
U-Net group 22	0.1468	0.1771
U-Net group 23	0.1202	0.1496
U-Net group 24	0.1386	0.2028
U-Net group 25	0.0858	0.1658

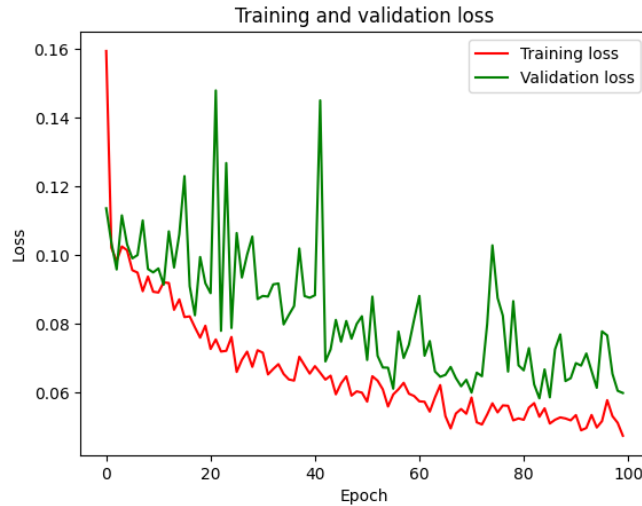


Figure 54: Loss over epoch graph for U-Net group 8 model

(ii) The U-Net Model Evaluation Metrics Results

The quality of the U-Net semantic segmentation model results was evaluated using the Dice coefficient and Intersection over Union. The evaluation metrics results for all the model experiments done are shown in Table 12. It can be seen that the best-performing group overall is U-Net group 8 which achieved a Dice coefficient of 96.45% and an Intersection over Union of 93.23%, with slight variations in other groups. Figure 55 shows the Intersection over Union over epoch graph for the U-Net group 8 model. In the graph, the training IoU begins near 0.84 while the validation IoU begins near 0.88 and they both increase steadily with some fluctuations which are milder for the validation IoU. At the 100th epoch the training IoU is near 0.94 while the validation IoU is near 0.92. This trend shows that the model learned well the dataset features and could segment the diseased areas well. Figure 56 shows the Dice Coefficient over epoch graph for the U-Net group 8 model. The graph shows that the training Dice Coefficient starts near 0.89 while the validation Dice Coefficient starts near 0.94 and they both rise steadily with some fluctuations. The validation Dice Coefficient had milder fluctuations. At the last epoch, the training Dice Coefficient was slightly over 0.96 while the validation Dice Coefficient was near 0.96. This trend shows that the model fits well on the data and could segment the diseased areas well.

Table 12: Model evaluation metric results

Model Group	IoU (%)	Dice Coefficient (%)	Validation IoU (%)	Validation Dice Coefficient (%)
U-Net group 1	86.71	92.21	87.31	93.08
U-Net group 2	86.98	92.47	87.99	93.57
U-Net group 3	83.97	90.67	87.82	93.42
U-Net group 4	87.51	92.73	91.41	95.49
U-Net group 5	88.01	92.98	91.38	95.47
U-Net group 6	92.86	95.97	91.62	95.58
U-Net group 7	92.77	96.01	91.71	95.65
U-Net group 8	93.28	96.25	93.23	96.45
U-Net group 9	82.90	90.05	83.49	90.90
U-Net group 10	85.09	91.38	84.06	91.29
U-Net group 11	84.42	90.51	80.34	89.05
U-Net group 12	89.72	93.72	87.48	93.20
U-Net group 13	70.26	80.77	78.96	87.83
U-Net group 14	75.41	84.83	75.01	85.55
U-Net group 15	91.77	95.33	92.46	95.94
U-Net group 16	81.51	88.35	77.13	86.97
U-Net group 17	74.07	84.18	72.68	84.10
U-Net group 18	73.29	83.43	73.71	84.76
U-Net group 19	77.96	86.77	78.07	87.66
U-Net group 20	81.45	88.61	79.22	88.11
U-Net group 21	76.03	85.53	76.64	86.70
U-Net group 22	79.41	87.42	84.30	91.26
U-Net group 23	84.92	90.80	86.52	92.73
U-Net group 24	83.08	89.97	80.72	89.22
U-Net group 25	89.51	93.99	83.39	90.93



Figure 55: Intersection over Union over epoch graph for U-Net group 8 model

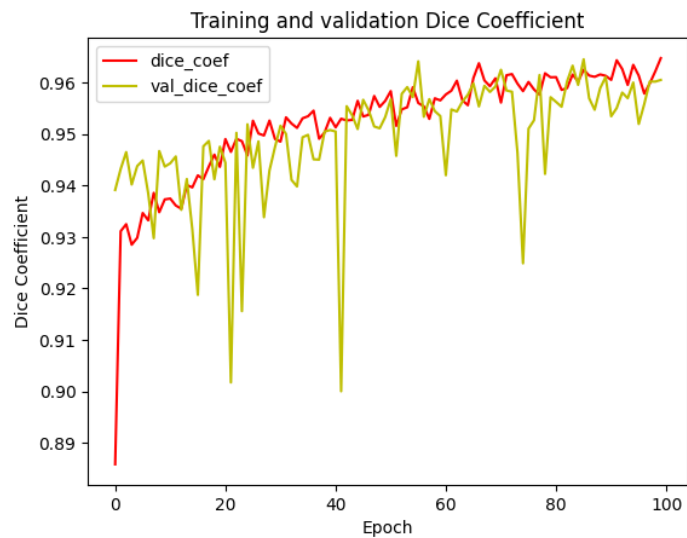


Figure 56: Dice coefficient over epoch graph for U-Net group 8 model

(iii) The U-Net Model Training Time Results

How efficient a model is in training is an important measure of its performance. The training times in minutes for all the model group experiments done are given in Table 13. The training times in minutes for the first 12 model group experiments ranged from 162.47 minutes to 175.63 minutes. The training times from group 13 to group 24 ranged from 89.16 minutes to 93.57 minutes because they had less data and the training time for group 25 model experiment was 87.77 minutes because it had the least amount of data.

Table 13: Model training time

Model Group	Training time (Mins)
U-Net group 1	166.82
U-Net group 2	172.12
U-Net group 3	168.62
U-Net group 4	175.63
U-Net group 5	164.7
U-Net group 6	165.66
U-Net group 7	166.32
U-Net group 8	170.3
U-Net group 9	167.1
U-Net group 10	162.47
U-Net group 11	163.42
U-Net group 12	168.41
U-Net group 13	92.44
U-Net group 14	92.45
U-Net group 15	91.91
U-Net group 16	92.43
U-Net group 17	91.44
U-Net group 18	92.43
U-Net group 19	92.22
U-Net group 20	89.16
U-Net group 21	93.57
U-Net group 22	91.43
U-Net group 23	92.55
U-Net group 24	93.49
U-Net group 25	87.77

The U-Net model could segment the two banana diseases well. Figure 57 shows the segmentation predictions for banana leaves affected by Fusarium Wilt and Black Sigatoka diseases. In Fig. 57, the bottom row shows the original images and the second row from the bottom displays the ground truths from the annotations. The third row from the bottom is the predictions from the U-Net model while the top row is the predictions overlaid on top of the original image. From Fig. 57, the model was able to segment the areas where the leaves were not green signifying the presence of disease, and leave out healthy green areas. From the study all images had either Black Sigatoka disease or Fusarium Wilt disease, so every image had only one class.

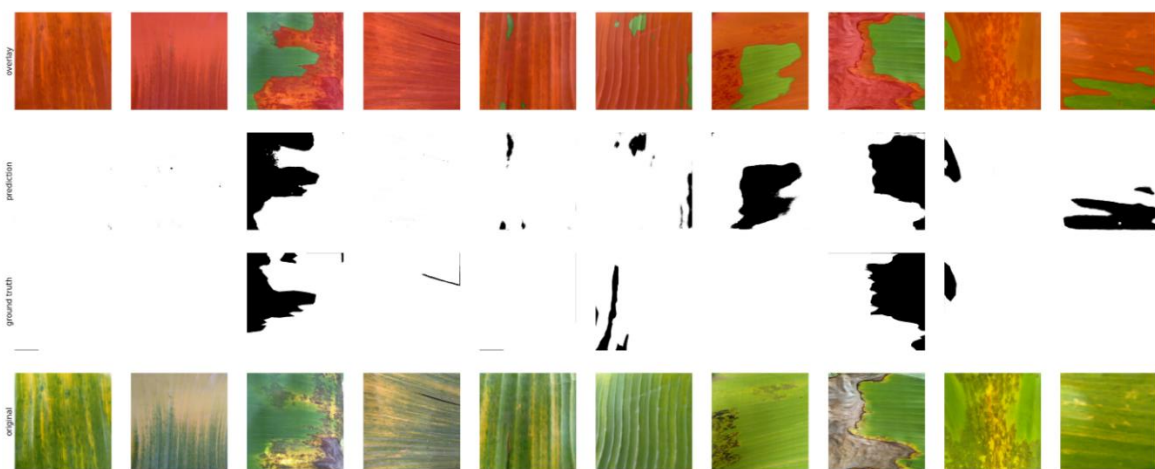


Figure 57: Segmentation predictions from the U-Net model

4.3 Model Deployment Results

An interactive and intuitive mobile application was developed to deploy the CNN deep learning model.

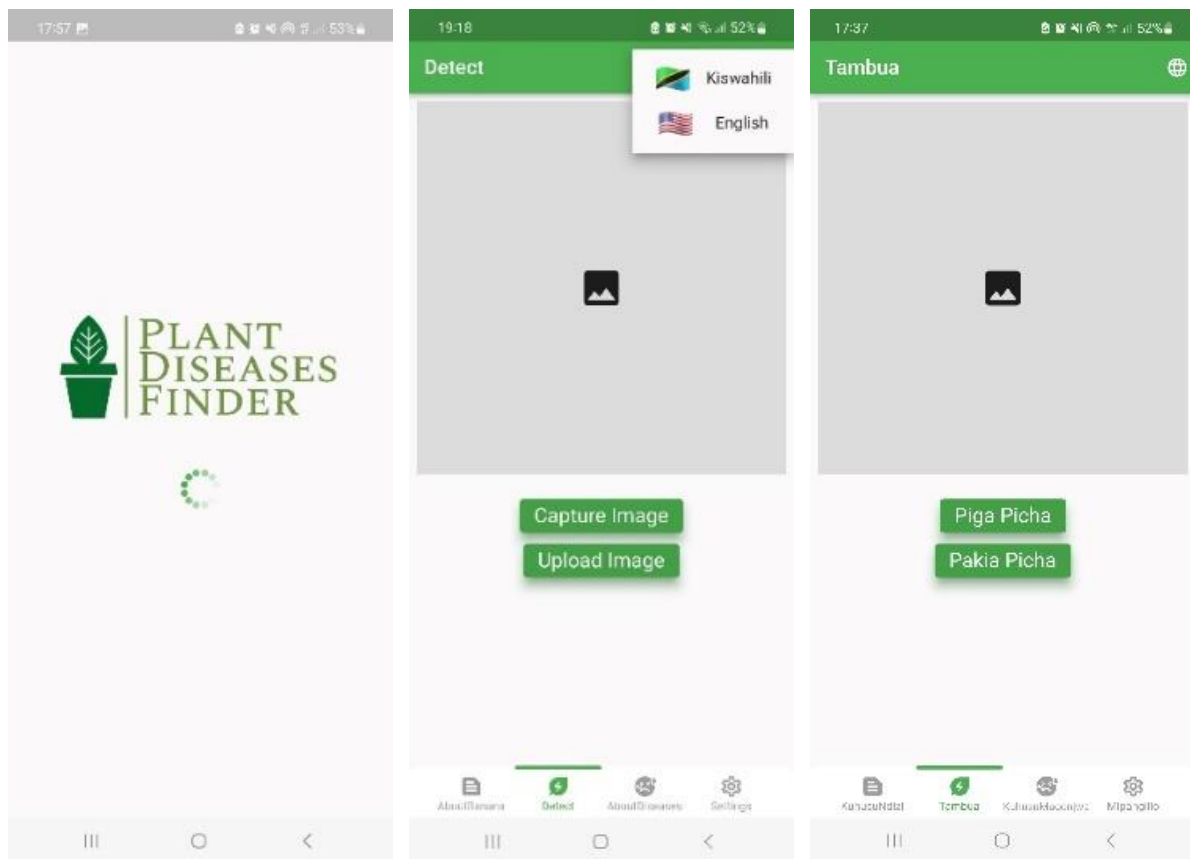
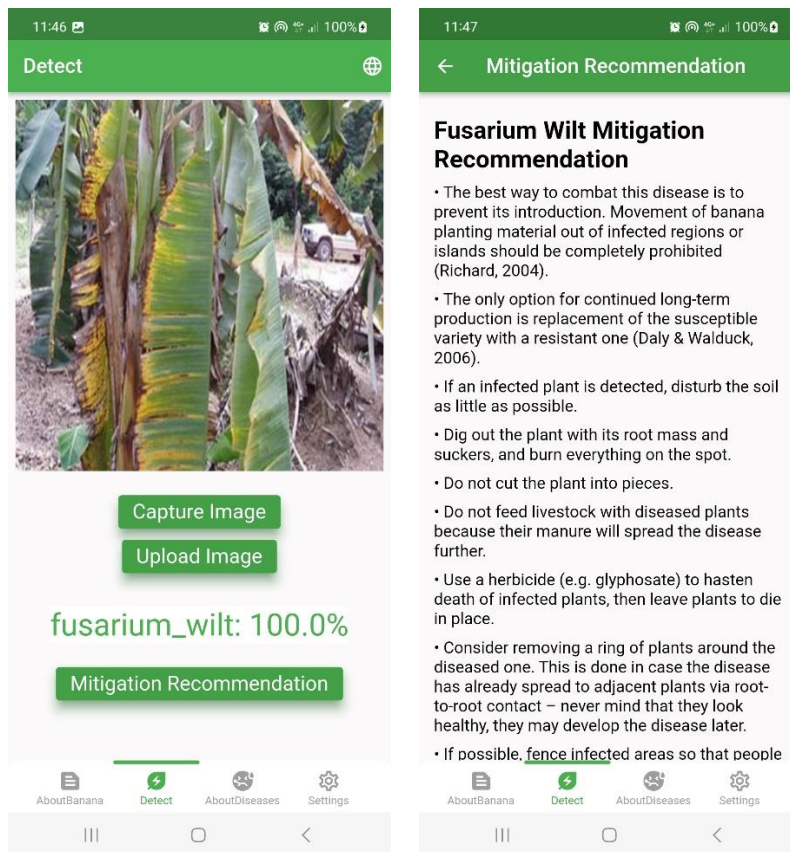


Figure 58: Banana disease detection mobile application splash screen and detect page in English and Kiswahili

An intuitive and easy-to-use mobile application was developed that helps farmers and extension officers detect Fusarium Wilt and Black Sigatoka banana diseases early. When the application is opened, a splash screen is displayed for a few seconds, and then the detect page is opened (Fig. 58). Capturing images from the mobile phone camera or uploading images from the phone's gallery can be done by the user in the "Detect" page. When a banana leaf or stalk image is captured or uploaded, the application automatically runs inference on the image in the background and displays the detection results. If either Black Sigatoka or Fusarium Wilt diseases are detected, the disease name and confidence score will be displayed together with a mitigation recommendation button, as seen in Fig. 59 and 60. When this mitigation recommendation button is pressed, it takes the user to a page that contains research-based mitigation recommendations for the specific detected diseases. This button will not appear when a healthy banana leaf or stalk is detected (Fig. 61). The detect page, about banana page, and about diseases page also have a change language feature on the top right where the user can select either of the two languages supported by this application, which are English and Kiswahili (Fig. 58). This feature is also included in the settings page, and it converts the entire application into either English or Kiswahili, as selected. This feature will help the local farmers

who do not understand English access the application in Kiswahili.



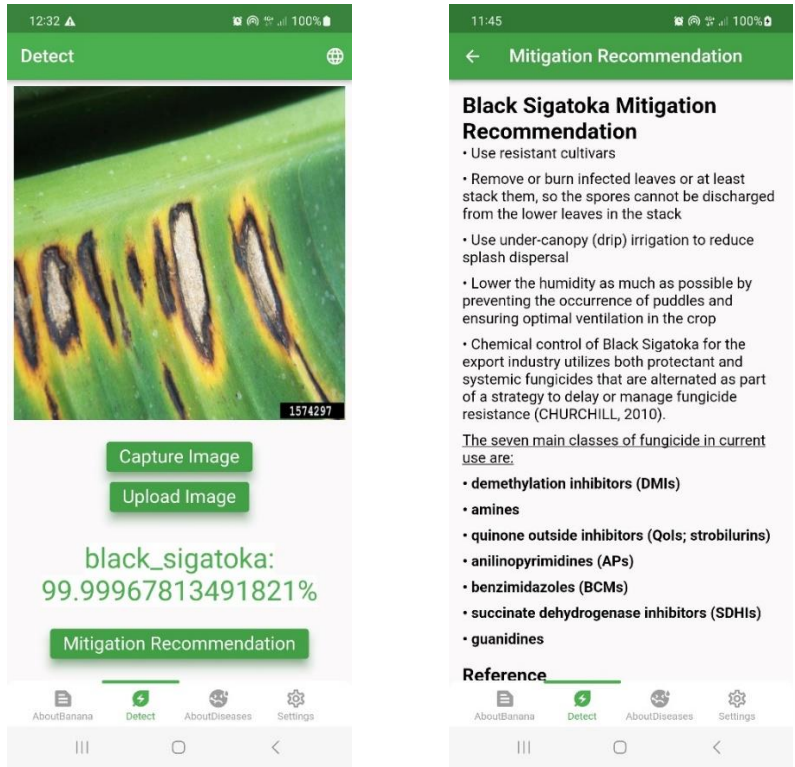


Figure 60: Banana disease detection mobile application detect page with detection results for Black Sigatoka and mitigation recommendation page for the detected disease

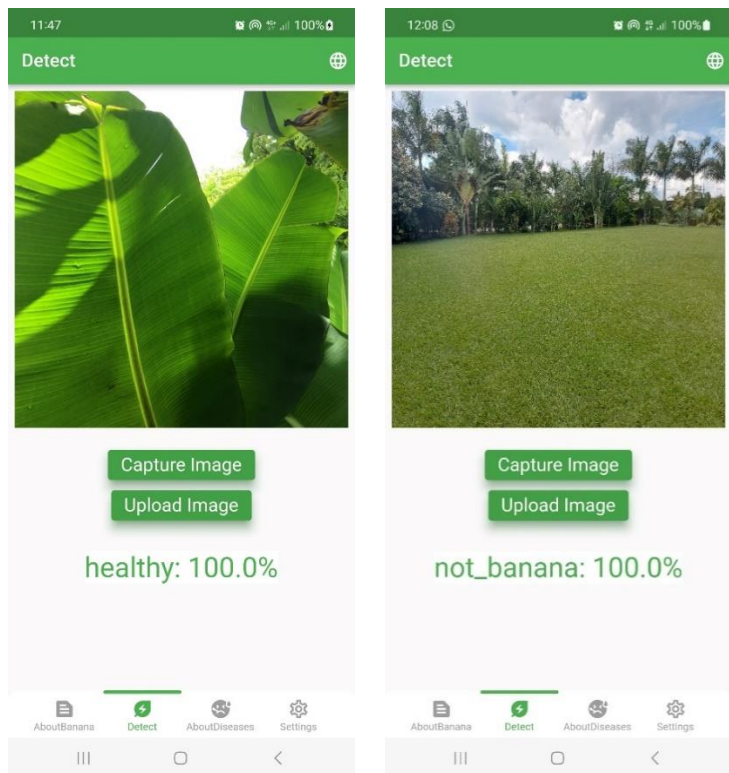


Figure 61: Banana disease detection mobile application detect page with results for a healthy banana leaf and an image that is not of a banana leaf or stalk

The mobile application also provides research-based information about bananas, including different types of bananas, the banana types that provide the highest yields, the banana types that have high demand in the market, and the best practices in banana farming in Tanzania (Fig. 62). This banana information was gathered as a requirement from farmers. Furthermore, the mobile application provides research-based information about Fusarium Wilt and Black Sigatoka banana diseases. Information about the symptoms, causes, transmission mechanism, and mitigation recommendations are provided for each disease. Users can prevent the occurrence of these two banana diseases by knowing their transmission mechanisms and avoiding them.

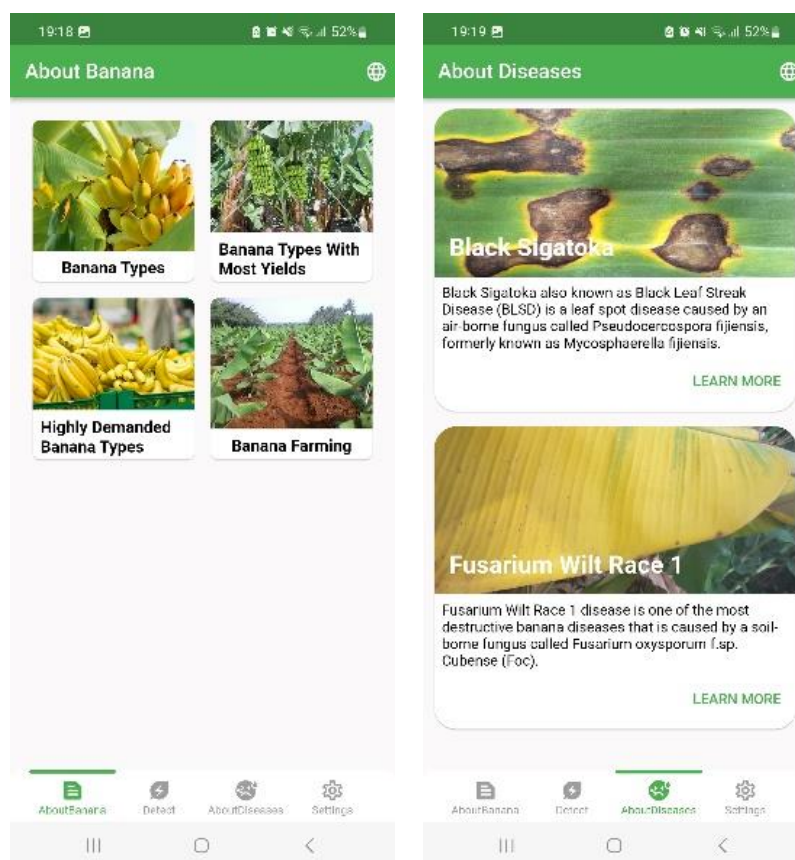


Figure 62: Banana disease detection mobile application about banana page and about diseases page

4.4 Validation of the Performance of the Developed Mobile Application Results

Table 14 summarizes the results of the responses given by farmers to the mobile application validation questionnaire. From the questionnaire, all but 1 question received a 100% response of yes from the farmers meaning that these features were working properly and the information was present. The question about the application running inference automatically in the

background using the CNN model received an 80% yes response and 20% no response. This response was due to the wrong predictions that were given by the application when the application deployed the TensorFlow Lite version of the CNN deep learning model. To improve the mobile application's predictions the mobile application deployed the original CNN TensorFlow model on a web server and an API was developed to send detection requests from the mobile application to the server and return responses.

Table 14: Results of the responses given by farmers to the mobile application validation questionnaire

Questions	Yes (%)	No (%)
Does the mobile application allow the farmer to capture an image using the mobile phone camera?	100	0
Does the mobile application allow the farmer to upload an image from the phone's gallery?	100	0
Does the mobile application allow the farmer to display the captured or uploaded image?	100	0
Does the mobile application run inference automatically on the device, in the background on the captured or uploaded image using the CNN model?	80	20
Does the mobile application allow the farmer to view detection results?	100	0
Does the mobile application allow the farmer to view mitigation recommendations when either Black Sigatoka or Fusarium Wilt diseases are detected?	100	0
Does the mobile application provide farmers with banana information including different types of bananas, banana types that provide the highest yields, banana types that have high demand in the market, and the best practices in banana farming?	100	0
Does the mobile application provide farmers with Black Sigatoka and Fusarium Wilt banana disease information including causes, symptoms, transmission mechanism, and mitigation recommendations?	100	0
Does the mobile application allow the farmer to change language from English to Swahili and vice versa?	100	0

4.5 Discussion

The achieved CNN model accuracy of 91.17% was considered good since a good CNN model should achieve an accuracy of at least 70% (Maxwell *et al.*, 2021). The CNN model also yielded a precision of 91.08%, recall of 91.62% and F-measure of 90.55%. Similar results were obtained by Sanga *et al.* (2020) when they deployed an Inceptionv3 model, which achieved an accuracy of 95.41%. Other similar results were obtained by Amara *et al.* (2017) when they used the LeNet architecture for the classification of banana leaf diseases and achieved an accuracy of 92.88%, precision of 92.99%, recall of 92.88%, and F1-score of 92.94%. Similar results were obtained by another study done by Bhuiyan *et al.* (2023) when they diagnosed banana leaf diseases using the BananaSqueezeNet model and achieved an accuracy of 96.25%, precision of 96.53%, recall of 96.25% and F1-score of 96.17%. Despite this good performance, this study used a small number of images. As a result, the assessed CNN model in this study is accurate and suitable for use in the early detection of banana diseases.

In the training times results for the CNN model experiments, the size of the images in KB was reduced from a maximum of 700 KB for images used in training CNN group 1 to a maximum of 100 KB for images used in training CNN group 2 to 5. This reduction of size helped the model to train faster while still making good predictions.

The Mask R-CNN model achieved a mean Average Precision (mAP) of 0.045 29. This result was obtained while testing in the initial experiments with a few datasets in the early stage of the research. The Mask R-CNN model was created to be compatible with TensorFlow 1. In the later stage of the research, TensorFlow 1 was deprecated and only TensorFlow 2 was used. An update was found for Mask R-CNN which was compatible with TensorFlow 2.4 but later this too was deprecated. The Mask R-CNN model was therefore not compatible with the current versions of TensorFlow which resulted in the inability to continue running experiments with different hyperparameters to improve the mean Average Precision value. A similar study done by Loyani *et al.* (2021) for the segmentation of a tomato plant paste named tuta absoluta using the Mask R-CNN model yielded a mean Average Precision of 85.67%. A similar study done by Selvaraj *et al.* (2019) focused on the detection of banana pests and diseases using a Faster R-CNN model based on ResNet50 and yielded a mean Average Precision of 99%, 70%, 97%, and 73% for the pseudostem, leaves, fruit bunch, and entire plant respectively.

U-Net group 8 model had the best performance from all the groups with a Dice Coefficient of

96.45% and an Intersection over Union of 96.52%. Similar results were obtained by Loyani *et al.* (2021) when they segmented a tomato plant paste called tuta absoluta using a U-Net model. Their model achieved a Dice Coefficient of 82.86% and an Intersection over Union of 78.60%. Also, similar results were obtained by Wang *et al.* (2023) when they segmented pear leaf diseases using an MFBP-UNet model. Their model achieved a Dice metric of 92%, and a Mean Intersection over Union of 86.15%. The dice coefficient usually has a higher value than Intersection over Union in the same segmentation performance.

The mobile application developed in this study did not deploy the Mask R-CNN or the U-Net image segmentation models because these models were not compatible with the flutter_tflite package used for interacting with deep learning models in the mobile application. The CNN model was compatible with this flutter package and therefore it was deployed in the mobile application.

Research shows that smallholder banana growers in East Africa have limited background knowledge of banana agronomy (CABI, 2019). There is also a limited uptake of recommendations on banana agronomy from research (CABI, 2019). The mobile application developed in this study provides research-based information about the banana plant (including banana types, banana types with most yields, highly demanded banana types and banana farming) and about Fusarium Wilt and Black Sigatoka banana diseases that can be easily consumed by farmers to improve their banana farming and production. The mobile application has an extra label and can predict other things apart from the healthy and diseased banana plants. Two languages are supported by the mobile application which are Swahili and English. The Swahili language will help farmers who do not understand the English language to be able to consume and use the information provided by the application. The provided research-based information and change language features are improvements from the mobile application developed by this study when compared to previously developed plant disease detection mobile application developed by Loyani and Machuve (2021) and Sanga *et al.* (2020). The mobile application developed in this study was able to correctly classify images of diseased and healthy banana plants and other images with a confidence score of more than 90% in less than 5 seconds per image. A similar study done by Hui *et al.* (2021) developed a mobile application that deployed an object detection model to detect grape diseases and the application yielded an accuracy of 97.9%. Another similar study done by Wang and Shabrina (2023) developed a mobile application that deploys EfficientNetB0 for the multi-class tomato plant disease

classification, and the application achieved an accuracy of 91.4%.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The objective of this research was to develop a deep-learning image segmentation model for early identification of banana diseases. To achieve this objective, the study assessed Mask R-CNN and U-Net image segmentation deep learning architectures, for instance, and semantic segmentation respectively of Fusarium Wilt and Black Sigatoka banana diseases. The study also assessed a classification Convolutional Neural Network (CNN) architecture to identify the two banana diseases. The results of the experiments showed that the Mask R-CNN ResNet101 model yielded a mAP of 0.045 29 in segmenting the two banana diseases. The results also showed that an Intersection over Union (IoU) of 93.23% and a Dice Coefficient of 96.45% was achieved by the U-Net model. The CNN model yielded an accuracy of 91.71% in classifying the two banana diseases. Additionally, the Fusarium Wilt and Black Sigatoka infected banana leaves and stalks were segmented using the Mask R-CNN and U-Net models.

This study also developed an interactive mobile application for the early detection of Fusarium Wilt and Black Sigatoka banana diseases. The mobile application deploys a CNN model that classifies these two diseases, healthy banana leaves, and images that are not of a banana leaf or stalk. The mobile application was able to correctly classify images with diseases, healthy images, and images that are not of the banana plant with a confidence of 90% and above in less than five seconds per image. The application could detect banana diseases at an early stage and provide research-based mitigation recommendations that extension officers and farmers can use to avoid yield losses and financial losses. The application also provides research-based information on banana farming and the two diseases. The feature of supporting English and Kiswahili languages plays a huge role in helping local farmers in rural areas who do not understand English. This work demonstrates how deep learning may be used to accurately identify diseased plants early enough for farmers to take the necessary precautions to reduce the damaging impacts of these diseases and save their yields.

5.2 Recommendations

This study recommends the Ministry of Agriculture and other agricultural stakeholders including Non-Governmental Organizations (NGOs) utilize the presented findings in

addressing the issue of food insecurity in Tanzania.

Farmers and agricultural extension officers are recommended to automatically detect Fusarium Wilt and Black Sigatoka diseases using the developed mobile application. They can also put to good use the research-based information provided by the mobile application and improve banana farming and production.

Furthermore, future studies could focus on developing a web-based system that will allow the administrator to update or patch the information on the mobile application easily using a web interface. The assessed models will be continually improved to provide farmers and extension officers with robust prediction models.

REFERENCES

- Afzaal, U., Bhattarai, B., Pandeya, Y. R., & Lee, J. (2021). An instance segmentation model for strawberry diseases based on mask R-CNN. *Sensors*, 21(19), 6565.
- Altendorf, S. (2019). *Food Outlook - Biannual Report on Global Food Markets*.
- Amara, J., Bouaziz, B., & Algergawy, A. (2017). A Deep Learning-based Approach for Banana Leaf Diseases Classification. *Datenbanksysteme für Business, Technologie und Web*.
- Arango-Isaza, R. E., Diaz-Trujillo, C., Dhillon, B., Aerts, A., Carlier, J., Crane, C. F., & Kema, G. H. (2016). Combating a global threat to a clonal crop: banana black Sigatoka pathogen *Pseudocercospora fijiensis* (synonym *Mycosphaerella fijiensis*) genomes reveal clues for disease control. *PLoS Genetics*, 12(8), e1005876.
- Awati, R. (2022). *Convolutional Neural Network (CNN)*. From TechTarget: <https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network#:~:text=A%20CNN%20is%20a%20kind,the%20network%20architecture%20of%20choice>.
- Bhuiyan, M. A. B., Abdullah, H. M., Arman, S. E., Rahman, S. S., & Al-Mahmud, K. (2023). BananaSqueezeNet: A very fast, lightweight convolutional neural network for the diagnosis of three prominent banana leaf diseases. *Smart Agricultural Technology*, 4, 100214.
- Bodapati, J. D., & Veeranjanyulu, N. (2019). Feature extraction and classification using deep convolutional neural networks. *Journal of Cyber Security and Mobility*, 8(2), 61-276.
- Brahimi, M., Boukhalifa, K., & Moussaoui, A. (2017). Deep learning for tomato diseases: Classification and symptoms visualization. *Applied Artificial Intelligence*, 31(4), 299-315.
- Brownlee, J. (2017). A gentle introduction to transfer learning for deep learning. *Machine Learning Mastery*, 20.
- Brownlee, J. (2019). *What is Deep Learning?* From Machine Learning Mastery. <https://machinelearningmastery.com/what-is-deep-learning>.

- Brownlee, J. (2020, 04 08). *4 Types of Classification Tasks in Machine Learning*. From towards data science. <https://machinelearningmastery.com/types-of-classification-in-machine-learning>.
- Bubici, G., Kaushal, M., Prigigallo, M. I., Gómez-Lama Cabanás, C., & Mercado-Blanco, J. (2019). Biological control agents against Fusarium wilt of banana. *Frontiers in Microbiology, 10*, 445720.
- CABI. (2019). *Improving banana agronomy practices for small scale farmers in East Africa*. (CABI) Retrieved from CABI: <https://www.cabi.org/projects/improving-banana-agronomy-practices-for-small-scale-farmers-in-east-africa>.
- Campos, H., Caligari, P. D., Brown, A., Tumuhimbise, R., Amah, D., Uwimana, B., & Swennen, R. (2017). Bananas and plantains (Musa spp.). *Genetic Improvement of Tropical Crops*, 219-240.
- Che'Ya, N. N., Mohidem, N. A., Roslin, N. A., Saberioon, M., Tarmidi, M. Z., Arif Shah, J., & Man, N. (2022). Mobile computing for pest and disease management using spectral signature analysis: A review. *Agronomy, 12*(4), 967.
- Daly, A., & Walduck, G. (2006). Fusarium Wilt of Bananas (Panama Disease). *Agnote, 151*.
- Daniel. (2016). *Bananas from Africa*. <http://inafrica24.com/modernity/bananas-from-africa>.
- Dearing, J. W., & Cox, J. G. (2018). Diffusion of innovations theory, principles, and practice. *Health Affairs, 37*(2), 183-190.
- Erdem, K. (2020). *Understanding Region of Interest - Part 2 (RoI Align)*. <https://erdem.pl/2020/02/understanding-region-of-interest-part-2-ro-i-align>.
- Erima, R., Kubiriba, J., Komutunga, E., Nowakunda, K., Namanya, P., Seruga, R., & Tushemereirwe, W. K. (2017). Banana pests and diseases spread to higher altitudes due to increasing temperature over the last 20 years. *African Journal of Environmental Science and Technology, 601-608*.
- Etebu, E., & Young-Harry, W. (2011). Control of black Sigatoka disease: Challenges and prospects. *African Journal of Agricultural Research, 6*(3), 508-514.

- FAO. (2017). *Global Programme on Banana Fusarium Wilt Disease*.
- FAO. (2021). *Acting together against banana diseases in Africa*. From Food and Agriculture Organization of United Nation. http://www.fao.org/agriculture/crops/news-events-bulletins/detail/en/item/36259/icode/en/?no_cache=1.
- FAOSTAT. (2021). *Crops and livestock products*. <https://www.fao.org/food-agriculture-statistics/statistical-domains/crop-livestock-and-food/en>.
- Fillipa, P. (2022). *Machine Learning Essentials: What is Data Annotation?* <https://resources.defined.ai/blog/machine-learning-essentials-what-is-data-annotation>.
- He, K., Gkioxari, G., Doll'ar, P., & Girshick, R. (2018). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2).
- Iyer, N. (2020). *Instance Segmentation*. <https://towardsdatascience.com/tagged/instance-segmentation>.
- Johanson, A., Tushemereirwe, W. K., & Karamura, E. B. (1996). Distribution of Sigatoka leaf spots in Uganda as determined by species-specific polymerase chain reaction (PCR). *I International Symposium on Banana: I International Conference on Banana and Plantain for Africa*, 540, 319-324.
- Jomanga, K. E., & Lucas, S. S. (2021). The Effects, Distribution and Management Options for Major Banana Diseases in Tanzania. *International Journal of Current Science Research and Review*, 1276-1295.
- Jomanga, K. E., Lucas, S. S., & Mgenzi, A. R. (2022). The Review on The Importance of Banana and Plantain Varieties that were/are Regarded as the Gold of Some Tribes in Tanzania. *International Journal of Novel Research in Life Sciences*, 50-61.
- Jordan, J. (2018). *An Overview of Semantic Image Segmentation*. <https://www.jeremyjordan.me/semantic-segmentation>.
- Hevner, A., Chatterjee, S., Hevner, A., & Chatterjee, S. (2010). Design science research in information systems. *Design Research in Information Systems: Theory and Practice*, 9-22.

- Hurwitz, J., Kirsch, D., & Wiley, J. (2018). *Machine Learning Machine Learning For Dummies*. John Wiley & Sons, Inc.
- Goyal, M., Guo, J., Hinojosa, L., Hulsey, K., & Pedrosa, I. (2022). Automated kidney segmentation by mask R-CNN in T2-weighted magnetic resonance imaging. In *Medical Imaging 2022: Computer-Aided Diagnosis, 12033*, 803-808.
- Kimunye, J., Were, E., Swennen, R., Viljoen, A., & Mahuku, G. (2021). Sources of resistance to *Pseudocercospora fijiensis*, the cause of black Sigatoka in banana. *Plant Pathology*, *70*(7), 1651-1664.
- Lai, P. C. (2017). The literature review of technology adoption models and theories for the novelty technology. *Journal of Information Systems and Technology Management*, *14*, 21-38.
- Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3431-3440.
- Loyani, L. K., Bradshaw, K., & Machuve, D. (2021). Segmentation of Tuta Absoluta's damage on tomato plants: A computer vision approach. *Applied Artificial Intelligence*, *35*(14), 1107-1127.
- Loyani, L., & Machuve, D. (2021). A deep learning-based mobile application for segmenting tuta absoluta's damage on tomato plants. *Engineering, Technology & Applied Science Research*, *11*(5), 7730-7737.
- Marr, B. (2018). *What Are Artificial Neural Networks - A Simple Explanation For Absolutely Anyone*. <https://www.forbes.com/sites/bernardmarr/2018/09/24/what-are-artificial-neural-networks-a-simple-explanation-for-absolutely-anyone>.
- Maxwell, A. E., Warner, T. A., & Guillén, L. A. (2021). Accuracy assessment in convolutional neural network-based deep learning remote sensing studies. Part 1: Literature review. *Remote Sensing*, *13*(13), 2450.
- Mduma, N., & Leo, J. (2023). Dataset of banana leaves and stem images for object detection, classification and segmentation: A case of Tanzania. *Data in Brief*, *49*, 109322.

- Mittal, S., & Hasija, Y. (2020). Applications of deep learning in healthcare and biomedicine. *Deep Learning Techniques for Biomedical and Health Informatics*, 57-77.
- Mkonyi, L., Rubanga, D., Richard, M., Zekeya, N., Sawahiko, S., Maiseli, B., & Machuve, D. (2020). Early identification of *Tuta absoluta* in tomato plants using deep learning. *Scientific African*, 10, e00590.
- Moroney, L. (2021). *AI and Machine Learning for Coders: A Programmer's Guide to Artificial Intelligence*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
- Muimba-Kankolongo, A. (2018). Food crop production by smallholder farmers in Southern Africa: Challenges and opportunities for improvement. Academic Press, Cambridge, MA, San Diego, CA.
- Nakatumba-Nabende, J., Akera, B., Tusubira, J. F., Nsumba, S., & Mwebaze, E. (2020). A dataset of necrotized cassava root cross-section images. *Data in Brief*, 32, 106170.
- Naoki. (2017). *Up-sampling with transposed convolution*. <https://kikaben.com/up-sampling-with-transposed-convolution>.
- Narayanan, K. L., Krishnan, R. S., Robinson, Y. H., Julie, E. G., Vimal, S., Saravanan, V., & Kaliappan, M. (2022). Banana plant disease classification using hybrid convolutional neural network. *Computational Intelligence and Neuroscience*, 2022.
- NBS, N. B., Agriculture, M. O., Fisheries, M. O., President's Office, R. A., Trade, M. O., & Ministry of Agriculture, I. N. (2021). *National Sample Census of Agriculture 2019/2020*.
- Ng, H. F., Lin, C. Y., Chuah, J. H., Tan, H. K., & Leung, K. H. (2021). Plant disease detection mobile application development using deep learning. *2021 International Conference on Computer & Information Sciences*, 34-38.
- Nirmal, M. D., Jadhav, P., & Kadu, N. B. (2022). Farmer Friendly Smart App for Pomegranate Disease Identification. *2022 International Conference on Edge Computing and Applications*, 884-890.

- Owomugisha, G., Quinn, J. A., Mwebaze, E., & Lwasa, J. (2014). Automated vision-based diagnosis of banana bacterial wilt disease and black sigatoka disease. *International Conference on the Use of Mobile ICT in Africa*, 1-5.
- Peters, J. F. (2017). *Foundations of Computer Vision*. Springer.
- Ploetz, R. C., Kema, G. H., & Ma, L. J. (2015). Impact of diseases on export and smallholder production of banana. *Annual Review of Phytopathology*, 53, 269-288.
- Tutorialspoint. (2015). *Artificial Intelligence - Intelligent Systems*. https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligent_systems.htm.
- Ral, R. (2020). Introduction to Deep Learning. *International Journal of Scientific Development and Research*, 5(9),369-371.
- Ramadhani, K., Machuve, D., & Jomanga, K. (2017). Identification and Analysis of Factors in Management of Banana Fungal Diseases: Case of Sigatoka (*Mycosphaerella fijiensis*. Mulder and Fusarium (*Fusarium Oxysporum* f. sp. cubense (foc) Diseases in Arumeru District. *Journal of Biodiversity and Environmental Sciences*, 11, 69-75.
- Ramcharan, A., Baranowski, K., McCloskey, P., Ahmed, B., Legg, J., & Hughes, D. P. (2017). Deep learning for image-based cassava disease detection. *Frontiers in Plant Science*, 8, 1852.
- Rasche, C. (2019). *Computer Vision*.
- Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), 1137 - 1149.
- Ronneberger, O. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net>.
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 234-241.

- Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2008). LabelMe: A Database and Web-Based Tool for Image Annotation. *International Journal of Computer Vision*, 77(1-3), 157–173.
- Sanga, S. (2020). *Development of an early detection tool for banana diseases: A case of Mbeya and Arusha region* (Doctoral dissertation, NM-AIST).
- Sanga, S. L., Machuve, D., & Jomanga, K. (2020). Mobile-based Deep Learning Models for Banana Disease Detection. *Engineering, Technology & Applied Science Research*, 10(3), 5674-5677.
- Sanga, S., Mero, V., Machuve, D., & Mwanganda, D. (2020). Mobile-based deep learning models for banana diseases detection. *arXiv preprint arXiv:2004.03718*.
- Selvaraj, M. G., Vergara, A., Ruiz, H., Safari, N., Elayabalan, S., Ocimati, W., & Blomme, G. (2019). AI-powered banana diseases and pest detection. *Plant Methods*, 15, 1-11.
- Shijie, J., Peiyi, J., & Siping, H. (2017). Automatic detection of tomato diseases and pests based on leaf images. *2017 Chinese Automation Congress*, 2537-2510.
- Shimwale, M. (2021). *Hope for Tanzanian banana farmers with the official release of new “matooke” hybrids*. <https://www.iita.org/news-item/hope-for-tanzanian-banana-farmers-with-the-official-release-of-new-matooke-hybrids>.
- Singh, O. (2023). *What are artificial intelligence (AI) crypto coins, and how do they work?* <https://cointelegraph.com/explained/what-are-artificial-intelligence-ai-crypto-coins-and-how-do-they-work>.
- Soares, J. M., Rocha, A. J., Nascimento, F. S., Santos, A. S., Miller, R. N., Ferreira, C. F., & Amorim, E. P. (2021). Genetic improvement for resistance to black Sigatoka in bananas: A systematic review. *Frontiers in Plant Science*, 12, 657916.
- Suleiman, R. (2018). Local and regional variations in conditions for agriculture and food security in Tanzania. A review. *AgriFoSe2030 Report*, (10).
- Tyagi, M. (2021). *Image Segmentation: Part 1*. From Towards Data Science: <https://towardsdatascience.com/image-segmentation-part-1-9f3db1ac1c50>.

- United Nations. (2021). *The Sustainable Development Goal Report 2021*.
- Vézina, A. (2022). *Fusarium wilt of banana*. <https://www.promusa.org/Fusarium+wilt>.
- Vézina, A., & Rouard, M. (2021). *Fusarium oxysporum f. sp. cubense*. From ProMusa: <https://www.promusa.org/Fusarium+oxysporum+f.+sp.+cubense>.
- Vézina, A., & Van-den-Bergh, I. (2020). *Black leaf streak*. From ProMusa: <https://www.promusa.org/Black+leaf+streak>.
- Vidhya, N. P., & Priya, R. (2022). Detection and Classification of Banana Leaf diseases using Machine Learning and Deep Learning Algorithms. *2022 IEEE 19th India Council International Conference*, 1-6.
- Singh, V., & Misra, A. K. (2017). Detection of plant leaf diseases using image segmentation and soft computing techniques. *Information Processing in Agriculture*, 4(1), 41-49.
- Venkatesh, V., & Davis, F. D. (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science*, 46(2), 186-204.
- Voora, V., Larrea, C., & Bermudez, S. (2020). *Global market report: Bananas*. Winnipeg, MB, Canada: International Institute for Sustainable Development.
- Wang, A. R., & Shabrina, N. H. (2023). A deep learning-based mobile app system for visual identification of tomato plant disease. *International Journal of Electrical and Computer Engineering*, 13(6), 6992-7004.
- Wang, H., Ding, J., He, S., Feng, C., Zhang, C., Fan, G., & Zhang, Y. (2023). MFBP-UNet: A network for pear leaf disease segmentation in natural agricultural environments. *Plants*, 12(18), 3209.
- Wang, Q., Qi, F., Sun, M., Qu, J., & Xue, J. (2019). Identification of tomato disease types and detection of infected areas based on deep convolutional neural networks and object detection techniques. *Computational Intelligence and Neuroscience*, 2019.
- Wood, T. (2020). *Convolutional Neural Network*. <https://deeptai.org/machine-learning-glossary-and-terms/convolutional-neural-network>.

- Zhang, X. (2021). *Understanding Mask R-CNN Basic Architecture*.
https://www.shuffleai.blog/blog/Understanding_Mask_RCNN_Basic_Architecture.html.
- Zhu, P., Isaacs, J., Fu, B., & Ferrari, S. (2017). Deep learning feature extraction for target recognition and classification in underwater sonar images. In *2017 IEEE 56th Annual Conference on Decision and Control*, 2724-2731.
- Zou, K. H., Warfield, S. K., Bharatha, A., Tempany, C. M., Kaus, M. R., Haker, S. J., & Kikinis, R. (2004). Statistical validation of image segmentation quality based on a spatial overlap index: scientific reports. *Academic Radiology*, *11*(2), 178-189.

APPENDICES

Appendix 1: Questions that Were Used to Come Up with the Functional and Non-Functional Requirements

Question Number	Question
1	What features should I included in the mobile application?
2	What information should I add to the mobile application that will be of use to farmers?

Appendix 2: Mobile Application Validation Questionnaire

<u>Questions / Maswali</u>	<u>Yes / Ndio</u>	<u>No / Hapana</u>
Does the mobile application allow the farmer to capture an image using the mobile phone camera? Je, programu ya simu inaruhusu mkulima kunasa picha kwa kutumia kamera ya simu ya mkononi?		
Does the mobile application allow the farmer to upload an image from the phone's gallery? Je, programu ya simu inaruhusu mkulima kupakia picha kutoka kwenye ghala ya simu?		
Does the mobile application allow the farmer to display the captured or uploaded image? Je, programu ya simu inaruhusu mkulima kuonyesha picha iliyonaswa au kupakiwa?		
Does the mobile application run inference automatically on the device, in the background on the captured or uploaded image using the CNN model? Je, programu ya simu huendesha makisio kiotomatiki kwenye kifaa, chinichini kwenye picha iliyonaswa au kupakiwa kwa kutumia muundo wa CNN?		
Does the mobile application allow the farmer to view detection results? Je, programu ya simu inaruhusu mkulima kuona matokeo ya ugunduzi?		
Does the mobile application allow the farmer to view mitigation recommendations when either Black Sigatoka or Fusarium Wilt diseases are detected? Je, programu ya simu ya mkononi inaruhusu mkulima kuona mapendekezo ya kupunguza wakati magonjwa ya Black Sigatoka au Fusarium Wilt yanapogunduliwa?		
Does the mobile application provide farmers with banana information including different types of bananas, banana types that provide the highest yields, banana types that have high demand in the market, and the best practices in banana farming?		

Je, programu ya simu ya mkononi inawapa wakulima taarifa za ndizi ikiwa ni pamoja na aina tofauti za ndizi, aina za ndizi zinazotoa mavuno mengi zaidi, aina za ndizi zinazohitajika sana sokoni, na mbinu bora za kilimo cha ndizi?

Does the mobile application provide farmers with Black Sigatoka and Fusarium Wilt banana disease information including causes, symptoms, transmission mechanism, and mitigation recommendations?

Je, programu ya simu ya mkononi huwapa wakulima taarifa za ugonjwa wa ndizi za Black Sigatoka na Fusarium Wilt ikiwa ni pamoja na sababu, dalili, utaratibu wa maambukizi na mapendekezo ya kupunguza?

Does the mobile application allow the farmer to change language from English to Swahili and vice versa?

Je, programu ya simu inamruhusu mkulima kubadilisha lugha kutoka Kiingereza hadi Kiswahili na kinyume chake?

Appendix 3: Mask R-CNN Model Source Code

Importing important libraries

```
import os
import sys
import random
import math
import re
import time
import numpy as np
import cv2
import cython
import scipy
import json
import pandas as pd
import datetime
from math import nan, isnan
# Used in plotting
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon

from skimage.io import imread, imshow, imread_collection,
concatenate_images
from skimage.transform import resize
```

Importing Mask R-CNN libraries

```
from mrcnn.config import Config
from mrcnn import utils
import mrcnn.model as modellib
from mrcnn import visualize
from mrcnn.model import log
from mrcnn.visualize import display_images
```

```
%matplotlib inline
```

Defining configurations

```
class DiseasesConfig(Config):
    """Configuration for training on the dataset.
    Derives from the base Config class and overrides values specific
    to the dataset.
    """
    # Give the configuration a recognizable name
    NAME = "diseases"
```

```

# Train on 1 GPU and 1 images per GPU. We can put multiple images
on each
# GPU. Batch size is (GPU_COUNT * images_per_GPU).
GPU_COUNT = 1
IMAGES_PER_GPU = 1 # Initially used 1

# Number of classes (including background)
NUM_CLASSES = 1 + 3 # background + black_sigatoka, fusarium_wilt,
healthy

# Use small images for faster training. Set the limits of the small
side
IMAGE_MIN_DIM = 512
IMAGE_MAX_DIM = 512

# Aim to allow ROI sampling to pick 33% positive ROIs.
TRAIN_ROIS_PER_IMAGE = 200

# set number of epoch
STEPS_PER_EPOCH = 150

# set validation steps
VALIDATION_STEPS = 50

# Backbone network architecture
# Supported values are: resnet50, resnet101.
BACKBONE = 'resnet101'

# The strides of each layer of the FPN Pyramid.
BACKBONE_STRIDES = [4, 8, 16, 32, 64]

# Anchor stride
RPN_ANCHOR_STRIDE = 1

# Non-max suppression threshold to filter RPN proposals.
RPN_NMS_THRESHOLD = 0.9 #default was 0.7

# If enabled, resizes instance masks to a smaller size to reduce
# memory load. Recommended when using high-resolution images.
USE_MINI_MASK = True
MINI_MASK_SHAPE = (28, 28)

# Use smaller anchors because our image and objects are small
RPN_ANCHOR_SCALES = (8, 16, 64, 128, 256)
MAX_GT_INSTANCES = 100
POST_NMS_ROIS_INFERENCE = 1000
POST_NMS_ROIS_TRAINING = 2000

```

```

# Minimum probability value to accept a detected instance
DETECTION_MIN_CONFIDENCE = 0.7
WEIGHT_DECAY = 0.0001

config = DiseasesConfig()
config.display()

class InferenceConfig(DiseasesConfig):
    # Set batch size to 1 since we'll be running inference on
    # one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GPU
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

```

Custom function to load the dataset

```

source = "diseases"
#####
# Dataset
#####
from skimage.io import imread, imshow, imread_collection,
concatenate_images
from skimage.transform import resize

class CustomDataset(utils.Dataset):

    def load_custom(self, dataset_dir, subset):
        """Load a subset of the Banana Disease dataset.
        dataset_dir: Root directory of the dataset.
        subset: Subset to load: train or test
        subset_class: Subset to load: black_sigatoka, fusarium_wilt or
healthy
        """

        # Train or test dataset?
        assert subset in ["train2", "test2"]
        dataset_dir = os.path.join(dataset_dir, subset)
        print(dataset_dir)

        # Train or validation dataset
        filenames = os.listdir(dataset_dir)
        jsonfiles, annotations = [], []
        for filename in filenames:
            if filename.endswith(".json"):
                jsonfiles.append(filename)
                annotation =
json.load(open(os.path.join(dataset_dir, filename)))

```

```

        # Insure this picture is in this dataset
        imagename = annotation['imagePath']
        if not
os.path.isfile(os.path.join(dataset_dir, imagename)):
            print(imagename)
            continue
        if len(annotation["shapes"]) == 0:
            continue
        # you can filter what you don't want to load
        annotations.append(annotation)

    print("In {source} {subset} dataset we have {number:d}
annotation files."
        .format(source=source,
subset=subset, number=len(jsonfiles)))
    print("In {source} {subset} dataset we have {number:d} valid
annotations."
        .format(source=source,
subset=subset, number=len(annotations)))

    labelslist = []
    for annotation in annotations:
        # Get the x, y coordinaets of points of the polygons that
make up
        # the outline of each object instance. These are stores in
the
        # shape_attributes (see json format above)
        shapes = []
        classids = []

        for shape in annotation["shapes"]:
            # first we get the shape classid
            label = shape["label"]
            if labelslist.count(label) == 0:
                labelslist.append(label)
                classids.append(labelslist.index(label)+1)
                shapes.append(shape["points"])

        # load_mask() needs the image size to convert polygons to
masks.
        width = annotation["imageWidth"]
        height = annotation["imageHeight"]
        self.add_image(
            source,
            image_id=annotation["imagePath"], # use file name as a
unique image id
            path=os.path.join(dataset_dir, annotation["imagePath"]),
            width=width, height=height,

```



```

        shapes=shapes, classids=classids)

    print("In {source} {subset} dataset we have {number:d} class
item"
        .format(source=source,
subset=subset,number=len(labelslist)))
    print(labelslist)

    # Add classes.
    for labelid, labelname in enumerate(labelslist):
        self.add_class(source,labelid,labelname)

def load_mask(self,image_id):
    """
    Generate instance masks for an image.
    Returns:
    masks: A bool array of shape [height, width, instance count]
with one mask per instance.
    class_ids: a 1D array of class IDs of the instance masks.
    """
    # If not the source dataset you want, delegate to parent class.
    image_info = self.image_info[image_id]
    if image_info["source"] != source:
        return super(self.__class__, self).load_mask(image_id)

    # Convert shapes to a bitmap mask of shape
    # [height, width, instance_count]
    info = self.image_info[image_id]
    mask = np.zeros([info["height"], info["width"],
len(info["shapes"])], dtype=np.uint8)
    #printsx,printsy=zip(*points)
    for idx, points in enumerate(info["shapes"]):
        # Get indexes of pixels inside the polygon and set them to
1
        pointsy,pointsx = zip(*points)
        rr, cc = skimage.draw.polygon(pointsx, pointsy)
        mask[rr, cc, idx] = 1
    masks_np = mask.astype(bool)
    classids_np = np.array(image_info["classids"]).astype(np.int32)
    # Return mask, and array of class IDs of each instance. Since
we have
    # one class ID only, we return an array of 1s
    return masks_np, classids_np

def image_reference(self,image_id):
    """Return the path of the image."""
    info = self.image_info[image_id]
    if info["source"] == source:

```

```

        return info["path"]
    else:
        super(self.__class__, self).image_reference(image_id)

```

Configuring the dataset

```

# Configuring the datasets
config = DiseasesConfig()
dataset_train, dataset_val = CustomDataset(), CustomDataset()
dataset_train.load_custom(dataset_path, "train2")
dataset_train.prepare()
dataset_val.load_custom(dataset_path, "test2")
dataset_val.prepare()
config.NUM_CLASSES = len(dataset_train.class_info)

```

Defining the model and loading weights

```

model = modellib.MaskRCNN(mode="training", config=DiseasesConfig(),
model_dir=MODEL_DIR)
model.keras_model.summary()

```

```

# Train new model using coco dataset
print("Loading weights ", weights_path)
model.load_weights(weights_path, by_name=True, exclude=[
    "mrcnn_class_logits", "mrcnn_bbox_fc",
    "mrcnn_bbox", "mrcnn_mask"])

```

Training the head layers

```

"""Train the model."""
# *** This training schedule is an example. Update to your needs ***
print("Training network heads")
start_train = time.time()
model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=5,
            layers='heads')
# augmentation=seq_of_aug)
history = model.keras_model.history.history
end_train = time.time()
minutes = round((end_train - start_train) / 60, 2)
print(f'Training took {minutes} minutes')

```

Plotting the model

```

history.keys()
# Get training statistics
#total loss
loss = history['loss']

```

```

val_loss = history['val_loss']
#rpn classification loss
rpn_class_loss = history['rpn_class_loss']
val_rpn_class_loss = history['val_rpn_class_loss']
#rpn bounding box loss
rpn_bbox_loss = history['rpn_bbox_loss']
val_rpn_bbox_loss = history['val_rpn_bbox_loss']
#Mask rcnn classification loss
mrcnn_class_loss = history['mrcnn_class_loss']
val_mrcnn_class_loss = history['val_mrcnn_class_loss']
#Mask rcnn bounding box loss
mrcnn_bbox_loss = history['mrcnn_bbox_loss']
val_mrcnn_bbox_loss = history['val_mrcnn_bbox_loss']
#mask loss
mrcnn_mask_loss = history['mrcnn_mask_loss']
val_mrcnn_mask_loss = history['val_mrcnn_mask_loss']
epochs = range(len(loss))

# Plot train & val loss
plt.plot(epochs, loss, 'ro-', label='Training loss')
plt.plot(epochs, val_loss, 'y', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.figure()

# Plot train & val rpn_class_loss
plt.plot(epochs, rpn_class_loss, 'ro-', label='rpn_class_loss')
plt.plot(epochs, val_rpn_class_loss, 'y', label='val_rpn_class_loss')
plt.title('1. rpn_class_loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.figure()

# Plot train & val rpn_bbox_loss
plt.plot(epochs, rpn_bbox_loss, 'ro-', label='rpn_bbox_loss')
plt.plot(epochs, val_rpn_bbox_loss, 'y', label='val_rpn_bbox_loss')
plt.title('2. rpn_bbox_loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.figure()

# Plot train & val mrcnn_class_loss
plt.plot(epochs, mrcnn_class_loss, 'ro-', label='mrcnn_class_loss')

```

```

plt.plot(epochs, val_mrcnn_class_loss, 'y',
label='val_mrcnn_class_loss')
plt.title('3. mrcnn_class_loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.figure()

# Plot train & val mrcnn_bbox_loss
plt.plot(epochs, mrcnn_bbox_loss, 'ro-', label='mrcnn_bbox_loss')
plt.plot(epochs, val_mrcnn_bbox_loss, 'y', label='val_mrcnn_bbox_loss')
plt.title('4. mrcnn_bbox_loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.figure()

# Plot train & val mrcnn_mask_loss
plt.plot(epochs, mrcnn_mask_loss, 'ro-', label='mrcnn_mask_loss')
plt.plot(epochs, val_mrcnn_mask_loss, 'y', label='val_mrcnn_mask_loss')
plt.title('5. mrcnn_mask_loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.show()

```

Function to test the model

```

def test(model, image_path = None, video_path=None, savedfile=None):
    assert image_path or video_path
    class_names = ['Background', 'healthy', 'BS', 'FW']

    # Image or video?
    if image_path:
        # Run model detection and generate the color splash effect
        print("Running on {}".format(image_path))
        # Read image
        image = skimage.io.imread(image_path)
        # Detect objects
        r = model.detect([image], verbose=1)[0]
        # Colorful
        import matplotlib.pyplot as plt

        _, ax = plt.subplots()
        visualize.display_instances(image, boxes=r['rois'],
        masks=r['masks'],
            class_ids = r['class_ids'], class_names=class_names,
        scores=r['scores'],

```

```

        title = "Banana Diseases Classification", ax = ax,
show_mask=True, show_bbox=True,)

    # Save output
    if savedfile == None:
        file_name =
"test_{:%Y%m%dT%H%M%S}.png".format(datetime.datetime.now())
    else:
        file_name = savedfile
    plt.savefig(file_name)
    #skimage.io.imsave(file_name, testresult)
elif video_path:
    pass
print("Saved to ", file_name)

```

Testing the model

```

model = modellib.MaskRCNN(mode="inference", config=InferenceConfig(),
model_dir=MODEL_DIR)

```

```

# Get path to saved weights
# Either set a specific path or find last trained weights
# model_path = os.path.join(ROOT_DIR, ".h5 file name here")

```

```

model_path = model.find_last()

```

```

# Load trained weights
print("Loading weights from ", model_path)
model.load_weights(model_path, by_name=True)

```

```

import os
# we test all models trained on the dataset in different stage
image_path =
'/content/drive/MyDrive/BananaDiseaseClassificationModel/Mask_RCNN/Data
set/test_old/FW_1545.jpg'
video_path =
'/content/drive/MyDrive/BananaDiseaseClassificationModel/Mask_RCNN/Data
set/test_old/FW_1545.jpg'
weights_path =
'/content/drive/MyDrive/BananaDiseaseClassificationModel/Mask_RCNN/logs
/diseases20230305T1642'
print(os.getcwd())
filenames = os.listdir(weights_path)
for filename in filenames:
    if filename.endswith(".h5"):
        print(f"Load weights from {filename}")
        model.load_weights(os.path.join(weights_path,
filename),by_name=True)
        savedfile_name = os.path.splitext(filename)[0] + ".jpg"

```

```

    test(model, image_path=image_path, video_path=video_path,
savedfile=savedfile_name)

```

Calculating the mean Average Precision

```

# Compute VOC-Style mAP @ IoU=0.5
# Running on 30 images. Increase for better accuracy.
image_ids = np.random.choice(dataset_val.image_ids,
len(dataset_val.image_ids))
APs = []
for image_id in image_ids:
    # Load image and ground truth data
    image, image_meta, gt_class_id, gt_bbox, gt_mask =\
        modellib.load_image_gt(dataset_val, InferenceConfig(),
            image_id, use_mini_mask=False)

    molded_images = np.expand_dims(modellib.mold_image(image,
InferenceConfig()), 0)
    # Run object detection
    results = model.detect([image], verbose=0)
    r = results[0]
    # Compute AP
    AP, precisions, recalls, overlaps =\
        utils.compute_ap(gt_bbox, gt_class_id, gt_mask,
            r["rois"], r["class_ids"], r["scores"],
r['masks'])
    APs.append(AP)
print("APs", APs)

print("Number of nan in APs List: ", len(APs) -
np.count_nonzero(~np.isnan(APs)))

APs = [x for x in APs if isnan(x) == False]
print("APs", APs)
print("mAP: ", np.mean(APs))

```

Appendix 4: The U-Net Model Source Code

Install keras-unet

```
pip install keras-unet
```

Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import glob
import os
import sys
import time
import math
import datetime
from PIL import Image
from tqdm import tqdm
```

Load images and their corresponding masks

```
masks = glob.glob("*.png")
orgs = list(map(lambda x: x.replace(".png", ".jpg"), masks))
```

Rendering the image in the right orientation

```
for image in orgs:

    img = Image.open(image)
    name = img.filename

    if hasattr(img, "_getexif") and img._getexif() is not None:
        exif = dict(img._getexif().items())
        orientation = exif.get(274)

        # Rotate the img based on the orientation metadata
        if orientation == 3:
            img = img.rotate(180, expand=True)
        elif orientation == 6:
            img = img.rotate(270, expand=True)
        elif orientation == 8:
            img = img.rotate(90, expand=True)
    img.save(name, 'JPEG')
```

Resizing the images and their corresponding masks then convert them into NumPy arrays

```
with tf.device(device_name): #use. GPU
    for n, id_ in tqdm(enumerate(orgs), total=len(orgs)):
```

```

imgs_list = []
masks_list = []
for image, mask in zip(orgs, masks):
    imgs_list.append(np.array(Image.open(image).resize((512,512))))
    masks_list.append(np.array(Image.open(mask).resize((512,512))))

imgs_np = np.asarray(imgs_list)
masks_np = np.asarray(masks_list)

# Save the NumPy Array in Drive
np.save('imgs_np', imgs_np);
np.save('masks_np', masks_np);

```

load the NumPy arrays from drive

```

imgs_np = np.load('imgs_np.npy')
masks_np = np.load('masks_np.npy')

```

Splitting the data into train and validation sets

```

from sklearn.model_selection import train_test_split

x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2,
random_state=0)

print("x_train: ", x_train.shape)
print("y_train: ", y_train.shape)
print("x_val: ", x_val.shape)
print("y_val: ", y_val.shape)

```

Prepare train generator with data augmentation

```

from keras_unet.utils import get_augmented

train_gen = get_augmented(
    x_train, y_train, batch_size=2,
    data_gen_args = dict(
        rotation_range=5.,
        width_shift_range=0.05,
        height_shift_range=0.05,
        shear_range=40,
        zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='constant'
    )
)

```


Configure the model

```
from keras_unet.models import custom_unet

input_shape = x_train[0].shape

model = custom_unet(
    input_shape,
    filters=32,
    use_batch_norm=True,
    dropout=0.3,
    dropout_change_per_layer=0.0,
    num_layers=4
)
model.summary()
```

Compile the model

```
from keras.callbacks import ModelCheckpoint

model_filename = 'banana_segm_model.h5'
callback_checkpoint = ModelCheckpoint(
    model_filename,
    verbose=1,
    monitor='val_loss',
    save_best_only=True,
)

#from keras.optimizers import Adam, SGD
from tensorflow.keras.optimizers import Adam, SGD
from keras_unet.metrics import iou, iou_thresholded, dice_coef
from keras_unet.losses import jaccard_distance

with tf.device(device_name):
    model.compile(
        optimizer=SGD(learning_rate=0.001, momentum=0.99),
        loss=jaccard_distance,
        metrics=[iou, iou_thresholded, dice_coef]
    )
```

Train the model

```
start_train = time.time()
history = model.fit_generator(
    train_gen,
    steps_per_epoch=400,
    epochs=100,
    validation_data=(x_val, y_val),
    callbacks=[callback_checkpoint]
```

```
)

end_train = time.time()
minutes = round((end_train - start_train) / 60, 2)
print(f'Training took {minutes} minutes')
```

Plotting the model

```
history.history.keys()

# Get training statistics
#iou
iou_thres = history.history['iou_thresholded']
val_iou_thres = history.history['val_iou_thresholded']
#loss
loss = history.history['loss']
val_loss = history.history['val_loss']
#dice coef
dice = history.history['dice_coef']
val_dice = history.history['val_dice_coef']

epochs = range(len(iou_thres))

# Plot train & val iou
plt.plot(epochs, iou_thres, 'b', label='iou')
plt.plot(epochs, val_iou_thres, 'y', label='val_iou')
plt.title('Training and validation IoU')
plt.ylabel('IoU')
plt.xlabel('Epoch')
plt.legend()
plt.figure()

# Plot train & val dice
plt.plot(epochs, dice, 'r', label='dice_coef')
plt.plot(epochs, val_dice, 'y', label='val_dice_coef')
plt.title('Training and validation Dice Coefficient')
plt.ylabel('Dice Coefficient')
plt.xlabel('Epoch')
plt.legend()
plt.figure()

# Plot train & val loss
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
```

```
plt.show()
```

Plot original image, ground truth, prediction, and overlay

```
model_filename =  
"/content/drive/MyDrive/BananaDiseaseClassificationModel/UNet/dataset7/  
batch13/banana_segmodel.h5"  
model.load_weights(model_filename)  
y_pred = model.predict(x_val)
```

```
from keras_unet.utils import plot_imgs
```

```
plot_imgs(org_imgs=x_val, mask_imgs=y_val, pred_imgs=y_pred,  
nm_img_to_plot=10)
```

Convert the model into TensorFlow Lite

```
# Convert the model.  
converter =  
tf.lite.TFLiteConverter.from_keras_model(model) #  
Convert a saved model with tf.lite.TFLiteConverter.from_saved_model()  
tflite_model = converter.convert()  
  
# Save the model.  
with  
open('/content/drive/MyDrive/BananaDiseaseClassificationModel/UNet/data  
set7/batch13/unet_model.tflite', 'wb') as f:  
    f.write(tflite_model)
```

Appendix 5: The CNN Model Source Code

Import libraries

```
# Import Tensorflow
import tensorflow as tf
import os
import time
import matplotlib.pyplot as plt
import numpy as np
```

```
Height_size = 512
Width_size = 512
Batch_Size = 32
```

Build the model

```
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 512x512
    # with 3 bytes color
    # This is the first convolution
    tf.keras.layers.Conv2D(16, (3,3), activation='relu',
input_shape=(Height_size, Width_size, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Define a dropout regularization layer.
    tf.keras.layers.Dropout(rate=0.2),
    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Define a dropout regularization layer.
    tf.keras.layers.Dropout(rate=0.2),
    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Define a dropout regularization layer.
    tf.keras.layers.Dropout(rate=0.2),
    # The fourth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Define a dropout regularization layer.
    tf.keras.layers.Dropout(rate=0.2),
    # Flatten the results to feed into a CNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    # 3 output neurons.
    tf.keras.layers.Dense(4, activation='softmax')
])
```

```
model.summary()
```

Compiling the model

```
from keras.callbacks import ModelCheckpoint

model_filename = 'cnn_model.h5'
callback_checkpoint = ModelCheckpoint(
    model_filename,
    verbose=1,
    monitor='val_loss',
    save_best_only=True,
)

from tensorflow.keras.optimizers import RMSprop

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss=
tf.keras.losses.CategoricalCrossentropy(from_logits=True,
name="categorical_crossentropy"),
              optimizer=opt,
              metrics=['accuracy',
tf.keras.metrics.Precision(thresholds=None),
tf.keras.metrics.Recall(thresholds=None)])
```

Train and validation datagen

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All training images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1/255)

# Flow training images in batches of 256 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/BananaDiseaseClassificationModel/CNN/Final/batch5/train/', # This is the source directory for training images
    target_size=(Height_size, Width_size), # All images will be
    resized to 512x512
    batch_size=Batch_Size,
    # Since we use sparse_categorical_crossentropy loss, we need
    categorical labels
    class_mode='categorical')

# All validation images will be rescaled by 1./255
validation_datagen = ImageDataGenerator(rescale=1/255)
```

```

# Flow validation images in batches of 128 using train_datagen
generator
validation_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/BananaDiseaseClassificationModel/CNN/Final/
batch5/test/', # This is the source directory for validation images
    target_size=(Height_size, Width_size), # All images will be
resized to 512x512
    batch_size=Batch_Size,
    # Since we use categorical_crossentropy loss, we need categorical
labels
    class_mode='categorical')

```

Train the model

```

start = time.time()
history = model.fit(
    train_generator,
    steps_per_epoch=94,
    epochs=100,
    validation_data=validation_generator,
    validation_steps = 24,
    verbose=1,
    callbacks=[callback_checkpoint])

end = time.time()
Training_time = end - start
print(f"Execution time: {round(Training_time, 5)} seconds")

```

Plot the performance graphs

```

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
t = f.suptitle('CNN Performance', fontsize=12)
f.subplots_adjust(top=0.85, wspace=0.3)

max_epoch = len(history.history['accuracy'])+1
epoch_list = list(range(1,max_epoch))
ax1.plot(epoch_list, history.history['accuracy'], label='Train
Accuracy')
ax1.plot(epoch_list, history.history['val_accuracy'], label='Validation
Accuracy')
ax1.set_xticks(np.arange(1, max_epoch, 5))
ax1.set_ylabel('Accuracy Value')
ax1.set_xlabel('Epoch')
ax1.set_title('Accuracy')
l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
ax2.plot(epoch_list, history.history['val_loss'], label='Validation
Loss')

```

```
ax2.set_xticks(np.arange(1, max_epoch, 5))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")
```

Appendix 6: Model Deployment Flutter Source Code

```
'package:banana_disease_detection/AboutDiseasesFragments/BlackSigatokaFragments/BSDiseaseMitigationRecommendation.dart';
import
'package:banana_disease_detection/AboutDiseasesFragments/FusariumWiltRace1Fragments/FWDiseaseMitigationRecommendation.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:persistent_bottom_nav_bar/persistent_tab_view.dart';
import
'package:banana_disease_detection/DetectFragments/CaptureImage.dart';
import 'package:banana_disease_detection/DetectFragments/UploadImage.dart';
import
'package:banana_disease_detection/AboutBananaFragments/BananaFarming.dart';
import
'package:banana_disease_detection/AboutBananaFragments/BananaTypes.dart';
import
'package:banana_disease_detection/AboutBananaFragments/BananaTypesWithMostYields.dart';
import
'package:banana_disease_detection/AboutBananaFragments/HighlyDemandedBananaTypes.dart';
import
'package:banana_disease_detection/AboutDiseasesFragments/BlackSigatoka.dart';
import
'package:banana_disease_detection/AboutDiseasesFragments/FusariumWiltRace1.dart';
import 'package:image_picker/image_picker.dart';
import 'dart:io';
import 'package:flutter_localizations/flutter_localizations.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:banana_disease_detection/classes/language_constants.dart';
import 'classes/language.dart';
import 'package:flutter_tflite/flutter_tflite.dart';
import 'package:flutter_spinkit/flutter_spinkit.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

void main() {
  runApp(const MyApp());
}

const String cnn = "ConvolutionalNeuralNetwork";
const String unet = "U-Net";

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  State<MyApp> createState() => _MyAppState();

  static void setLocale(BuildContext context, Locale newLocale) {
    _MyAppState? state = context.findAncestorStateOfType<_MyAppState>();
    state?.setLocale(newLocale);
  }
}
```



```

class _MyAppState extends State<MyApp> {

  Locale? _locale;

  setLocale(Locale locale) {
    setState(() {
      _locale = locale;
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      localizationsDelegates: AppLocalizations.localizationsDelegates,
      supportedLocales: AppLocalizations.supportedLocales,
      locale: _locale,
      // home: BottomNavBar(),
      home: const SplashScreen(),

      initialRoute: "/",
      routes: {
      },
    );
  }
}

class SplashScreen extends StatefulWidget {
  const SplashScreen({Key? key}) : super(key: key);

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {

  @override
  void initState() {
    // TODO: implement initState
    super.initState();
    Future.delayed(const Duration(seconds: 3)).then((value) {
      Navigator.of(context).pushReplacement(
        CupertinoPageRoute(builder: (ctx) => const BottomNavBar()));
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SizedBox(
        width: double.infinity,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: const [
            Image(image: AssetImage("assets/images/logo.png"),
              width: 300,
            ),
            SizedBox(
              height: 50,
            ),
          ],
        ),
      ),
    );
  }
}

```

```

        SpinKitFadingCircle(
            color: Colors.green,
            size: 50.0,
        ),
    ],
),
);
}
}

class BottomNavBar extends StatefulWidget {
  const BottomNavBar({Key? key}) : super(key: key);

  @override
  State<BottomNavBar> createState() => _BottomNavBarState();
}

class _BottomNavBarState extends State<BottomNavBar> {
  @override
  Widget build(BuildContext context) {
    List<Widget> _buildScreens() {
      return [
        const AboutBanana(),
        const Detect(),
        const AboutDiseases(),
        const Settings(),
      ];
    }

    List<PersistentBottomNavBarItem> _navBarsItems() {
      return [
        PersistentBottomNavBarItem(
          icon: const Icon(Icons.text_snippet),
          title: (AppLocalizations.of(context)!.menuAboutBanana),
          activeColorPrimary: Colors.green,
          inactiveColorPrimary: Colors.grey,
        ), PersistentBottomNavBarItem(
          icon: const Icon(Icons.energy_savings_leaf),
          title: (AppLocalizations.of(context)!.menuDetect),
          activeColorPrimary: Colors.green,
          inactiveColorPrimary: Colors.grey,
        ),
        PersistentBottomNavBarItem(
          icon: const Icon(Icons.sick),
          title: (AppLocalizations.of(context)!.menuAboutDiseases),
          activeColorPrimary: Colors.green,
          inactiveColorPrimary: Colors.grey,
        ), PersistentBottomNavBarItem(
          icon: const Icon(Icons.settings_outlined),
          title: (AppLocalizations.of(context)!.menuSettings),
          activeColorPrimary: Colors.green,
          inactiveColorPrimary: Colors.grey,
        ),
      ];
    }

    PersistentTabController controller;

```

```

controller = PersistentTabController(initialIndex: 1);
return PersistentTabView(
  context,
  screens: _buildScreens(),
  items: _navBarsItems(),
  controller: controller,
  confineInSafeArea: true,
  backgroundColor: Colors.white,
  handleAndroidBackButtonPress: true,
  resizeToAvoidBottomInset: true,
  stateManagement: true,
  hideNavigationBarWhenKeyboardShows: true,
  decoration: NavBarDecoration(
    borderRadius: BorderRadius.circular(10.0),
    colorBehindNavBar: Colors.white,
  ),
  popAllScreensOnTapOfSelectedTab: true,
  popActionScreens: PopActionScreensType.all,
  itemAnimationProperties: const ItemAnimationProperties(
    duration: Duration(milliseconds: 200),
    curve: Curves.ease,
  ),
  screenTransitionAnimation: const ScreenTransitionAnimation(
    animateTabTransition: true,
    curve: Curves.ease,
    duration: Duration(milliseconds: 200),
  ),
  navBarStyle:
    NavBarStyle.style3,

);
}
}

// AboutBanana
class AboutBanana extends StatefulWidget {
  const AboutBanana({Key? key}) : super(key: key);

  @override
  State<AboutBanana> createState() => _AboutBananaState();
}

class _AboutBananaState extends State<AboutBanana> {
  @override
  Widget build(BuildContext context) {

    const banana_types = 'assets/images/banana_types.jpg';
    const banana_types_with_most_yields =
'assets/images/banana_types_with_most_yields.jpg';
    const highly_demanded_banana_types =
'assets/images/highly_demanded_banana_types.jpg';
    const banana_farming = 'assets/images/banana_farming.jpg';

    return Scaffold(
      appBar:
        AppBar(
          title: Text(AppLocalizations.of(context)!.titleAboutBanana),
          backgroundColor: Colors.green,

```

```

actions: <Widget>[
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: DropdownButton<Language>(
      underline: const SizedBox(),
      icon: const Icon(
        Icons.language,
        color: Colors.white,
      ),
      onChanged: (Language? language) async {
        if (language != null) {
          MyApp.setLocale(context, Locale(language.languageCode,
            ''));
        }
      },
      items: Language.languageList()
        .map<DropdownMenuItem<Language>>(
          (e) => DropdownMenuItem<Language>(
            value: e,
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceAround,
              children: <Widget>[
                Text(
                  e.flag,
                  style: const TextStyle(fontSize: 30),
                ),
                Text(e.name)
              ],
            ),
          ),
        )
        .toList(),
    ),
  ],
),

body: GridView.count(
  primary: false,
  padding: const EdgeInsets.all(20),
  crossAxisSpacing: 10,
  mainAxisSpacing: 10,
  crossAxisCount: 2,
  children: <Widget>[
    GestureDetector(
      onTap: () {
        PersistentNavBarNavigator.pushNewScreenWithRouteSettings(
          context,
          settings: const RouteSettings(name: "/banana"),
          screen: const BananaTypes(),
          pageTransitionAnimation:
            PageTransitionAnimation.fade,
        );
      },
    ),
    child: Card(
      clipBehavior: Clip.antiAlias,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      color: Colors.white,
      child: Column(

```

```

        children: [
          Image.asset(
            banana_types,
            height: 140,
            width: 180,
            fit: BoxFit.cover,
          ),
          Padding(
            padding: const EdgeInsets.all(8).copyWith(bottom: 0),
            child: Text(
              AppLocalizations.of(context)!.cardBananaTypes,
              style: const TextStyle(
                fontWeight: FontWeight.bold,
                color: Colors.black,
                fontSize: 16,
              ),
            ),
          ),
        ],
      ),
    ),
  ),
  GestureDetector(
    onTap: () {
      PersistentNavBarNavigator.pushNewScreenWithRouteSettings(
        context,
        settings: const RouteSettings(name: "/banana"),
        screen: const BananaTypesWithMostYields(),
        pageTransitionAnimation:
          PageTransitionAnimation.fade,
      );
    },
    child: Card(
      clipBehavior: Clip.antiAlias,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      color: Colors.white,
      child: Column(
        children: [
          Image.asset(
            banana_types_with_most_yields,
            height: 120,
            width: 180,
            fit: BoxFit.cover,
          ),
          Padding(
            padding: const EdgeInsets.all(8).copyWith(bottom: 0),
            child: Text(
              AppLocalizations.of(context)!.cardBananaTypesWithMostYields,
              style: const TextStyle(
                fontWeight: FontWeight.bold,
                color: Colors.black,
                fontSize: 16,
              ),
            ),
          ),
        ],
      ),
    ),
  ),
),

```

```

),
GestureDetector(
  onTap: () {
    PersistentNavBarNavigator.pushNewScreenWithRouteSettings(
      context,
      settings: const RouteSettings(name: "/banana"),
      screen: const HighlyDemandedBananaTypes(),
      pageTransitionAnimation:
        PageTransitionAnimation.fade,
    );
  },
  child: Card(
    clipBehavior: Clip.antiAlias,
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(8),
    ),
    color: Colors.white,
    child: Column(
      children: [
        Image.asset(
          highly_demanded_banana_types,
          height: 120,
          width: 180,
          fit: BoxFit.cover,
        ),
        Padding(
          padding: const EdgeInsets.all(8).copyWith(bottom: 0),
          child: Text(
AppLocalizations.of(context)!.cardHighlyDemandedBananaTypes,
            style: const TextStyle(
              fontWeight: FontWeight.bold,
              color: Colors.black,
              fontSize: 16,
            ),
          ),
        ),
      ],
    ),
  ),
  GestureDetector(
    onTap: () {
      PersistentNavBarNavigator.pushNewScreenWithRouteSettings(
        context,
        settings: const RouteSettings(name: "/banana"),
        screen: const BananaFarming(),
        pageTransitionAnimation:
          PageTransitionAnimation.fade,
      );
    },
    child: Card(
      clipBehavior: Clip.antiAlias,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      color: Colors.white,
      child: Column(
        children: [
          Image.asset(
            banana_farming,

```

```

        height: 140,
        width: 180,
        fit: BoxFit.cover,
      ),
      Padding(
        padding: const EdgeInsets.all(8).copyWith(bottom: 0),
        child: Text(
          AppLocalizations.of(context)!.cardBananaFarming,
          style: const TextStyle(
            fontWeight: FontWeight.bold,
            color: Colors.black,
            fontSize: 16,
          ),
        ),
      ),
    ],
  ),
);
}
}

```

```
// Detect
```

```

class Detect extends StatefulWidget {
  const Detect({Key? key}) : super(key: key);

  @override
  State<Detect> createState() => _DetectState();
}

class _DetectState extends State<Detect> {

  String _model = cnn;
  File? file;
  List? _outputs;
  ImagePicker image = ImagePicker();

  Future predictImage(XFile image) async {
    switch (_model) {
      case unet:
        await segmentImage(image);
        break;
      default:
        await classifyImage(image);
        // await recognizeImageBinary(image);
    }
  }

  @override
  void initState() {
    super.initState();
    loadModel().then((value) {setState((){});});
  }

  Future segmentImage(XFile image) async {
    int startTime = DateTime.now().millisecondsSinceEpoch;
    var output = await Tflite.runSegmentationOnImage(
      path: image.path,

```

```

        imageMean: 127.5,
        imageStd: 127.5,
    );

    setState(() {
        _outputs = output!;
    });
    print(_outputs);
    int endTime = DateTime.now().millisecondsSinceEpoch;
    print("Inference took ${endTime - startTime}ms");
}

Future classifyImage(XFile image) async {
    int startTime = DateTime.now().millisecondsSinceEpoch;
    var output = await Tflite.runModelOnImage(
        path: image.path,
        numResults: 1,
        threshold: 0.05,
        imageMean: 127.5,
        imageStd: 127.5,
    );

    setState(() {
        _outputs = output!;
    });
    print(_outputs);
    int endTime = DateTime.now().millisecondsSinceEpoch;
    print("Inference took ${endTime - startTime}ms");
}

Future<void> sendImage(String path) async {
    String url = 'http://192.168.1.248:5000/api/';

    Map<String, String> headers = {
        'Content-Type': 'application/json', // Adjust the content type as
needed
    };

    Map<String, String> body = {
        'imageBase64': path,
    };

    String jsonBody = jsonEncode(body);

    try {
        final response = await http.post(
            Uri.parse(url),
            headers: headers,
            body: jsonBody,
        );

        if (response.statusCode == 200) {
            print('Image uploaded successfully!');
            Map<String, dynamic> data = jsonDecode(response.body);
            List<Map<String, dynamic>> output = [
                {'confidence': data["confidence_score"], 'label':
data['class_name']}]];
            print("output test");

            setState(() {
                _outputs = output!;
            });
        }
    }
}

```



```

    });
    print(_outputs);

    } else {
        print('Image upload failed. Status code: ${response.statusCode}');
        print('Response body: ${response.body}');
    }
} catch (e) {
    print('Error sending image: $e');
}
}

Future<String> imageToBase64(XFile file) async {
    List<int> imageBytes = await file!.readAsBytes();
    return base64Encode(imageBytes);
}

Future loadModel() async {
    try {
        String res;
        switch (_model) {
            case unet:
                res = (await Tflite.loadModel(
                    model: "assets/model/unet_sept.tflite",
                    labels: "assets/model/unet_labels.txt",
                    // useGpuDelegate: true,
                ))!;
                break;
            default:
                res = (await Tflite.loadModel(
                    model: "assets/model/cnn_model.tflite",
                    labels: "assets/model/label.txt",
                    // useGpuDelegate: true,
                ))!;
        }
        print(res);
    } on PlatformException {
        print('Failed to load model.');
```

```

    }
}

@override
void dispose() {
    Tflite.close();
    super.dispose();
}

```

```

@override
Widget build(BuildContext context) {

    final ButtonStyle style = ElevatedButton.styleFrom(
        textStyle:
        const TextStyle(fontSize: 20),
        backgroundColor: Colors.green[600],
        shadowColor: Colors.green[600],
        elevation: 10,
    );

```

```

    List<Widget> stackChildren = [];

```

```

if (_model == cnn) {
  if (_outputs == null) {
    print("Null");
  } else {
    stackChildren.add(Center(
      child: Column(
        children: _outputs!.map((res) {
          return Column(
            children: <Widget>[
              Text(
                "${res["label"]}: ${res["confidence"]}% ",
                textAlign: TextAlign.center,
                style: TextStyle(
                  color: Colors.green,
                  fontSize: 30.0,
                  background: Paint()
                    ..color = Colors.white,
                ),
              ),
              ),
              if(res["label"] == 'healthy' || res["label"] ==
'not_banana')...[] else...[
                const SizedBox(height: 20),
                ElevatedButton(
                  style: style,
                  onPressed: () {
                    if (res["label"] == 'black_sigatoka') {
                      PersistentNavBarNavigator
                        .pushNewScreenWithRouteSettings(
                          context,
                          settings: const RouteSettings(name: "/home"),
                          screen: const
BSDiseaseMitigationRecommendation(),
                          pageTransitionAnimation:
PageTransitionAnimation
                              .fade,
                        );
                    } else if (res["label"] == 'fusarium_wilt'){
                      PersistentNavBarNavigator
                        .pushNewScreenWithRouteSettings(
                          context,
                          settings: const RouteSettings(name: "/home"),
                          screen: const
FWDiseaseMitigationRecommendation(),
                          pageTransitionAnimation:
PageTransitionAnimation
                              .fade,
                        );
                    } else {
                      }
                    },
                    child: Text(AppLocalizations.of(context)!
                      .buttonMitigationRecommendation),
                  ),
                ],
              ],
            );
          }).toList(),
        ),
      ));
    }
  }
}

```

```

} else if (_model == unet) {
  if (_outputs == null) {} else {
    stackChildren.add(Positioned(
      child: Container(
        decoration: BoxDecoration(
          image: DecorationImage(
            alignment: Alignment.topCenter,
            image: MemoryImage(
              Uint8List.fromList(
                _outputs!.map((e) =>
int.parse(e.toString()).toList()
              )
            ),
            fit: BoxFit.fill)),
        child: Opacity(
          opacity: 0.3, child: Image.file(File(file!.path))),
      ),
    );
  }
}

return Scaffold(
  appBar:
  AppBar(
    title: Text(AppLocalizations.of(context)!.titleDetect),
    backgroundColor: Colors.green,
    actions: <Widget>[
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: DropdownButton<Language>(
          underline: const SizedBox(),
          icon: const Icon(
            Icons.language,
            color: Colors.white,
          ),
          onChanged: (Language? language) async {
            if (language != null) {
              MyApp.setLocale(context, Locale(language.languageCode,
''));
            }
          },
          items: Language.languageList()
            .map<DropdownMenuItem<Language>>(
              (e) => DropdownMenuItem<Language>(
                value: e,
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.spaceAround,
                  children: <Widget>[
                    Text(
                      e.flag,
                      style: const TextStyle(fontSize: 30),
                    ),
                    Text(e.name)
                  ],
                ),
              ),
            ).toList(),
        ),
      ),
    ],
  ),
);

```

```

),
body: CustomScrollView(
  slivers: <Widget>[
    SliverToBoxAdapter(
      child: Padding(
        padding: const EdgeInsets.all(8).copyWith(bottom: 0),
        child: Column(
          children: <Widget>[
            Container(
              height: 400,
              width: 400,
              color: Colors.black12,
              child: file == null
                ? const Icon(
                  Icons.image,
                  size: 50,
                )
                : Image.file(
                  file!,
                  fit: BoxFit.fill,
                ),
            ),
            const SizedBox(height: 20),
            ElevatedButton(
              style: style,
              onPressed: () {
                getcam();
              },
              child:
Text(AppLocalizations.of(context)!.buttonCaptureImage),
            ),
            ElevatedButton(
              style: style,
              onPressed: () {
                getgall();
              },
              child:
Text(AppLocalizations.of(context)!.buttonUploadImage),
            ),
            const SizedBox(height: 30),
            Stack(
              children: stackChildren,
            ),
          ],
        ),
      ),
    ],
  ),
);
}

// Camera Functions
getcam() async {
  // ignore: deprecated_member_use
  var img = await image.pickImage(source: ImageSource.camera);
  if (img == null) return null;
  setState(() {
    file = File(img!.path);
  });
}

```

```

        //predictImage (img);

        Uint8List imagebytes = await img.readAsBytes(); //convert to bytes
        String base64string = base64.encode(imagebytes); //convert bytes to
base64 string
        sendImage(base64string);

    }

    // Gallery Functions
    getgall() async {
        // ignore: deprecated_member_use
        var img = await image.pickImage(source: ImageSource.gallery);
        if (img == null) return null;
        setState(() {
            file = File(img!.path);
        });
        // predictImage (img);

        Uint8List imagebytes = await img.readAsBytes(); //convert to bytes
        String base64string = base64.encode(imagebytes); //convert bytes to
base64 string
        sendImage(base64string);

    }
}

// AboutDiseases
class AboutDiseases extends StatefulWidget {
    const AboutDiseases({Key? key}) : super(key: key);

    @override
    State<AboutDiseases> createState() => _AboutDiseasesState();
}

class _AboutDiseasesState extends State<AboutDiseases> {

    @override
    Widget build(BuildContext context) {
        const urlBSImage = 'assets/images/Black_Sigatoka.jpg';
        const urlFWImage = 'assets/images/Fusarium_Wilt.jpg';

        return Scaffold(
            appBar:
                AppBar(
                    title: Text(AppLocalizations.of(context)!.titleAboutDiseases),
                    backgroundColor: Colors.green,
                    actions: <Widget>[
                        Padding(
                            padding: const EdgeInsets.all(8.0),
                            child: DropdownButton<Language>(
                                underline: const SizedBox(),
                                icon: const Icon(
                                    Icons.language,
                                    color: Colors.white,
                                ),
                            ),
                            onChanged: (Language? language) async {
                                if (language != null) {
                                    MyApp.setLocale(context, Locale(language.languageCode,
''));
                                }
                            }
                        ]
                    )
                )
        );
    }
}

```



```

        child: Text(
AppLocalizations.of(context)!.textFusariumWiltRace1,
        style: const TextStyle(
            fontWeight: FontWeight.bold,
            color: Colors.white,
            fontSize: 24,
        ),
    ),
),
),
],
),
Padding(
padding: const EdgeInsets.all(8).copyWith(bottom:
0),
        child: Text(
AppLocalizations.of(context)!.textFusariumWiltRace1Intro,
        style: const TextStyle(fontSize: 14, color:
Colors.black),
    ),
),
AppBar(
    alignment: MainAxisAlignment.end,
    children: [
        TextButton(
            onPressed: () {
PersistentNavBarNavigator.pushNewScreenWithRouteSettings(
                context,
                settings: const RouteSettings(name:
"/diseases"),
                screen: const FusariumWiltRace1(),
                pageTransitionAnimation:
PageTransitionAnimation.fade,
            );
        },
        child: Text(
AppLocalizations.of(context)!.buttonFWLearnMore,
        style: const TextStyle(fontSize: 14, color:
Colors.green),
    ),
    ],
),
),
),
),
),
),
),
),
),
);
}
}

// Settings
class Settings extends StatefulWidget {
    const Settings({Key? key}) : super(key: key);

```



```

@override
State<Settings> createState() => _SettingsState();
}

class _SettingsState extends State<Settings> {

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar:
      AppBar(
        title: Text(AppLocalizations.of(context)!.titleSettings),
        backgroundColor: Colors.green
      ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        children: <Widget>[
          DropdownButton<Language>(
            iconSize: 30,
            hint: Text(AppLocalizations.of(context)!.textChangeLanguage),
            onChanged: (Language? language) async {
              if (language != null) {
                MyApp.setLocale(context, Locale(language.languageCode,
''));
              }
            },
            items: Language.languageList()
              .map<DropdownMenuItem<Language>>(
                (e) => DropdownMenuItem<Language>(
                  value: e,
                  child: Row(
                    mainAxisAlignment: MainAxisAlignment.spaceAround,
                    children: <Widget>[
                      Text(
                        e.flag,
                        style: const TextStyle(fontSize: 30),
                      ),
                      Text(e.name)
                    ],
                  ),
                ),
              ).toList(),
          ],
        ),
      ),
    );
}
}

```

RESEARCH OUTPUTS

i) **Journal Paper**

Elinisa, C. A., & Mduma, N. (2024). Mobile-Based Convolutional Neural Network Model for the Early Identification of Banana Diseases. *Smart Agricultural Technology*, 100423.


ii) **Poster Presentation**

OUTPUT 2: Poster Presentation



1 The Nelson Mandela African Institution of science and Technology

Image Segmentation Deep Learning Model for Early Detection of Banana Diseases



Anglophone Africa

Christian A. Elinisa 1
Ciira wa Maina 2
Anthony Vodacek 3
Neema Mduma 1

2 Dedan Kimathy University of Technology
3 Rochester Institute of Technology

- Banana production is highly affected by Fusarium Wilt and Black Sigatoka fungal diseases (Sanga et al., 2020).
- These diseases cause great yield losses in banana production ranging from 30% to 100% (Bubici et al., 2019; Vezina & Van den Bergh, 2020).
- There is a limitation in agricultural extension services (Sanga et al., 2020).
- No effective way for early detection of Fusarium Wilt and Black Sigatoka diseases.
- Therefore an early detection solution that use deep learning deployed in a mobile application.

CONCEPTUAL FRAMEWORK




RESEARCH METHODOLOGY

Model Development

- CNN model for Classification
- U-Net model for Semantic Segmentation
- Mask R-CNN model for Instance Segmentation

THE DATASET

Study area - Arusha, Kagera, Kilimanjaro, Mbeya, and Dar es Salaam regions of Tanzania
 Data collection tool - Mobile phone cameras
 Dataset - 27,360 banana images






Healthy



Black Sigatoka



Fusarium Wilt

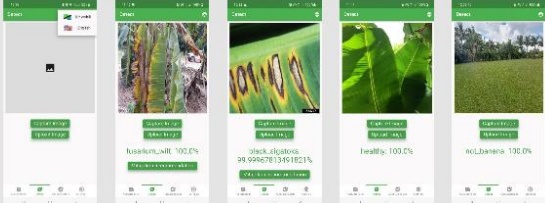


Not Banana

RESULTS



Mask R-CNN model results



Mobile Application results

- The mobile application can be used by Farmers and extension officers to automatically detect Fusarium Wilt and Black Sigatoka diseases.
- Following the provided research-based mitigation recommendations early can help farmers and extension officers avoid yields and financial losses.
- In the future - a web-based system can be developed to allow the administrator to update the information on the mobile application easily.

Source:

- Sanga, S., Mero, V., Machuve, D., & Mwanganda, D. (2020). Mobile-Based Deep Learning Models for Banana Diseases Detection. The International Conference on Learning Representations (ICLR). doi:10.48550/arXiv.2004.03718
- Bubici, G., Kaushal, M., Prigigallo, M. I., Cabanás, C. G.-L., & Mercado-Blanco, J. (2019). Biological Control Agents Against Fusarium Wilt of Banana. *Frontiers in Microbiology*.
- Vézina, A., & Van den Bergh, I. (2020, 8 5). Black leaf streak. From ProMusa: <https://www.promusa.org/Black+leaf+streak>.