The Nelson Mandela AFrican Institution of Science and Technology

NM-AIST Repository **https://dspace.mm-aist.ac.tz**

Computational and Communication Science Engineering Masters Theses and Dissertations [CoCSE]

2021-09

A quantification model against tuta absoluta effects on tomato plants: a computer vision approach

Loyani, Loyani

NM-AIST

https://doi.org/10.58694/20.500.12479/1602

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

A QUANTIFICATION MODEL AGAINST *TUTA ABSOLUTA* EFFECTS ON TOMATO PLANTS: A COMPUTER VISION APPROACH

Loyani Kisula Loyani

A Dissertation Submitted in Partial Fulfillment 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

September, 2021

ABSTRACT

Tomatoes are among the most commonly cultivated crops in the world. It is considered a high-value crop and income resource for smallholder farmers in Africa. Nevertheless, its production currently endangered by *Tuta absoluta* pest. The pest has severely damaged tomato yields to the extent that growers are giving up tomato production due to the high costs and losses incurred. It causes a heavy loss in tomato produce ranging from 80 to 100% when not effectively managed. Recently, farmers have been using different methods in efforts to control the pest. These include using pheromone traps and natural enemies for population monitoring, planting resistant tomato varieties, and continuous spraying of chemical pesticides, which is now the main control method. These practices have been proven not to be effective in controlling the pest; they are time-consuming and relatively expensive. Inspired by the progression and positive outcomes of computer vision methods in diagnosing a wide variety of plant diseases and pests, this study proposes a segmentation-based quantification model for detecting and quantifying *Tuta absoluta*'s damage to tomato plants. We develop convolutional neural network models based on U-Net and Mask RCNN architectures for automatic semantic and instance segmentation respectively using data collected from the field. Experimental results show that Mask RCNN achieved a mAP of 85.67% and U-Net obtained 78.60% and 82.86% of Jaccard index and Dice Coefficient respectively. Both models were precise in segmenting the shapes of *Tuta absoluta*-infected areas in tomato leaves and determine their extent of the damage. The model was then deployed on the mobile phone to enable farmers and extension officers in Tanzania to automatically detect affected areas on tomato plants and make informed decisions on how to control the pest so as to increase tomato production and save farmers from the losses they face.

DECLARATION

I, Loyani Kisula Loyani, do hereby 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 or similar award in any other institution.

Loyani Kisula Loyani		12/10/2021
Candidate Name	**Signature**	**Date**

The above declaration is confirmed by:

Dr. Dina Machuve		13/10/2021
Supervisor Name	**Signature**	**Date**

Prof. Karen Bradshaw		
Supervisor Name	**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 acknowledgment, 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 entitled, *A Quantification Model Against Tuta Absoluta Effects on Tomato Plants: A Computer Vision Approach* 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. Dina Machuve		13/10/2021
Supervisor Name	**Signature**	**Date**

Prof. Karen Bradshaw

Supervisor Name	**Signature**	**Date**

ACKNOWLEDGEMENT

First and foremost, I thank the Almighty God for the blessings He bestowed upon me throughout my study at the Nelson Mandela African Institution of Science and Technology (NM-AIST). I thank Him for blessing me with courage, strength, endurance, tenacity, and good health to successfully complete this Master's program.

Throughout the writing of this dissertation, I have received a great deal of support and assistance.

I would like to express my deepest gratitude to my supervisor Dr. Dina Machuve for being a mentor and tirelessly providing guidance towards the completion of this work; also, to my supervisor Prof. Karen Bradshaw from Rhodes University in South Africa for her guidance and for providing insightful feedbacks especially during dissertation writing. Thank you for being such great supervisors and mentors.

The completion of this work would not have been possible without the financial support from the African Development Bank (AfDB) that funded this research through project ID No. P-Z1-IA0-016 under grant No. 2100155032816.

Many thanks to the NM-AIST for making a conducive environment that led to the completion of this work. The NM-AIST provided me with computer lab space and other tools that I needed to complete my study and achieve my research goals successfully.

In addition, I would like to thank my parents for their wise counsel and sympathetic ear. You are always there for me. Finally, I am very grateful to my lecturers, colleagues, friends, and classmates with whom we shared several challenges and achievements. Of great importance, those who kept me in their prayers.

May God bless you all.

DEDICATION

I dedicate this work to my father, Mr. Kisula Ezekiel Loyani and my lovely mother, Mrs. Elizabeth Loyani. Their love and support are beyond comparison.

TABLE OF CONTENTS

ABSTRACT i

DECLARATION ii

COPYRIGHT iii

CERTIFICATION iv

ACKNOWLEDGEMENT 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 4

1.4 Research Objectives 4

1.4.1 General Objective 4

1.4.2 Specific Objectives 4

1.5 Research Questions 5

1.6 Significance of the Study 5

1.7 Delineation of the Study 6

CHAPTER TWO 7

LITERATURE REVIEW 7

2.1 Tomato Cropping 7

2.2 The Tomato Leaf Miner (*Tuta absoluta*) 8

2.3 Computer Vision Approach 10

2.4 Plant Diseases Diagnostics using Deep Learning 11

2.5 Research Gap 14

CHAPTER THREE 16

MATERIALS AND METHODS 16

3.1 Study Area 16

3.2 Field Setup 17

3.2.1 Planting and Pest Introduction 17

3.3 The Dataset 19

3.4 Research Framework 21

3.5 Image Preprocessing 21

3.5.1 Labelling and Cropping 22

3.5.2 Image Annotation 23

3.5.3 Resizing the Images 25

3.5.4 Augmentation 25

3.6 Transfer Learning 28

3.7 Implementation 34

3.8 Training Phase 34

3.8.1 U-Net: Hyperparameters Tuning and Network Training 34
3.8.2 Mask RCNN: Hyperparameters Tuning and Network Training 35
3.9 Evaluation 37
3.9.1 Intersection over union (IoU) 37
3.9.2 Dice Coefficient (F1 Score) 38
3.9.3 Precision 38
3.9.4 Recall 39
3.9.5 Mean Average Precision (mAP) 39
3.9.6 Loss Function 40
3.10 Model Deployment 42
3.10.1 Requirement Analysis 42
3.10.2 Assumptions and Dependencies 44
3.10.3 Use Case Modelling 44
3.10.4 Activity Diagram 47
3.10.5 Sequence Diagram 47
3.10.6 Software Development Methodology 48
3.10.7 Technologies Used 49
3.11 Quantification 50
CHAPTER FOUR 51
RESULTS AND DISCUSSION 51
4.1 The Dataset 51
4.2 Loss Results 51
4.2.1 Mask RCNN Loss Results 51
4.2.2 U-Net Loss Results 53

4.3 Evaluation Metrics Results 54
4.3.1 Jaccard Index/IoU 54
4.3.2 Dice Coefficient 54
4.3.3 The mAP 56
4.4 Training Time 58
4.5 Developed Mobile Application 59
4.6 Quantification Results 64
CHAPTER FIVE 65
CONCLUSION AND RECOMMENDATIONS 65
5.1 Conclusion 65
5.2 Recommendations 65
REFERENCES 67
APPENDICES 76
RESEARCH OUTPUTS 94

LIST OF TABLES

Table 1: Data collection setup and factors considered for each experiment. 20

Table 2: Dataset distribution. 22

Table 3: Configurations used for training Mask RCNN model. 36

Table 4: Description of the Use Cases. 46

Table 5: Train/test set splits. 51

Table 6: The evaluation metrics results for semantic segmentation model. 55

Table 7: The mAP (primary metric) values of the tomato images obtained by different detection architectures. 58

Table 8: Training time. 59

LIST OF FIGURES

Figure 1: Tuta absoluta's damage on tomatoes 4

Figure 2: World production of tomatoes (FAO, 2019) 7

Figure 3: Worldwide distribution of Tuta absoluta (Soares & Campos, 2020) 9

Figure 4: Tuta absoluta's life cycle and damage images (a) Four stages of Tuta absoluta's life cycle (b) Tomato leaf with Tuta absoluta mines (c) Severe damage on tomato field (d) Damaged tomato fruits in the field (e) Damaged tomato fruit on the market 9

Figure 5: Convolutional Neural Network architecture (O'Shea & Nash, 2015) 11

Figure 6: Research study area 16

Figure 7: Experimental setup in a field 17

Figure 8: Transplanting the tomato seedlings in (a) Arusha and (b) Morogoro fields 18

Figure 9: A researcher and an agricultural expert performing infestation in the field (a) The process carried out in Arusha field (b) Process in Morogoro field 18

Figure 10: Data collection using (a) high and (b) low-resolution cameras in Arusha and Morogoro fields 20

Figure 11: Some images from the field depicting the Tuta absoluta's damage status 20

Figure 12: Research conceptual framework 21

Figure 13: Labelled and cropped images from the dataset showing the development of Tuta mines on different days 23

Figure 14: Examples of the two data annotation methods (a) Labelme annotation tool (b) VGG Image Annotator (VIA) tool 24

Figure 15: Image annotation process 24

Figure 16: Illustration of the image annotation process (a) Original image (b) Polygonal annotation of the Tuta mines contour (c) Visualization of labels (d) Extraction of the mask (e) Original image with the overlapping mask 25

Figure 17: Original image (top) and the results of the data augmentation techniques 27

Figure 18: Mask augmentations 27

Figure 19: U-Net architecture 29

Figure 20: Proposed Mask RCNN model architecture 30

Figure 21: Backbone feature maps at (a) input layer, (b) res2c_out activation layer, (c) res3c_out activation layer, and (d) res4c_out activation layer 31

Figure 22: Illustration of how RPN works (Pawang, 2020) 32

Figure 23: The RPN anchors (a) Regions of Interest (RoIs) (b) Negative anchors (c) Positive anchors (d) Top anchors before refinement (e) Top anchors with refinement (f) Refined anchors after non-max suppression 33

Figure 24: Augmented images with their corresponding annotations 35

Figure 25: Illustration of bounding boxes or masks overlaps and their corresponding IoU values 38

Figure 26: The use case diagram for Tuta absoluta segmentation application 45

Figure 27: The activity diagram for Tuta absoluta segmentation application 47

Figure 28: The sequence diagram for Tuta absoluta segmentation application 48

Figure 29: Agile methodology (Abellán, 2020) 49

Figure 30: Training and validation loss for Mask RCNN (a) Loss graph for Mask RCNN-ResNet50 (b) Loss graph for Mask RCNN-ResNet101 52

Figure 31: Training and validation loss for Mask RCNN with augmentations. (a) Loss graph for Mask RCNN-Resnet50 with augmentations. (b) Loss graph for Mask RCNN-Resnet101 with augmentations 52

Figure 32: Training and validation loss for U-Net 53

Figure 33: Intersection over Union (IoU) curve for U-Net model 54

Figure 34: Dice Coefficient curve of the U-Net model 55

Figure 35: Examples of segmentations achieved by the proposed U-Net model 56

Figure 36: The P-R Curve 57

Figure 37: Examples of segmentations carried out by the proposed Mask RCNN model ... 58

Figure 38: Tuta absoluta segmentation mobile application (a) Splash screen (b) Landing/home page 61

Figure 39: Tuta absoluta segmentation mobile application (a) The description page for tomato cropping (b) Description page for Tuta absoluta 62

Figure 40: Tuta absoluta segmentation mobile application (a) A healthy plant with no tuta mines (b) Original image and mask prediction (c) Segmentation results 63

Figure 41: Examples of quantification results carried out by the OpenCV function built on top of the proposed Mask RCNN model 64

LIST OF APPENDICES

Appendix 1: Mask RCNN Model Source Code 76

Appendix 2: U-Net Model Source Code 81

Appendix 3: Tensorflow Lite Converter Source Code 83

Appendix 4: Model Deployment Android Studio Source Code 87

Appendix 5: Mask RCNN Object Counting Source Code. 91

LIST OF ABBREVIATIONS AND SYMBOLS

CNN	Convolutional Neural Network
COCO	Common Objects in Context
CV	Computer Vision
DCNN	Deep Convolutional Neural Network
DE-Net	Dilated Encoder Network
DL	Deep Learning
FAO	Food and Agriculture Organization
Faster RCNN	Faster Region-based Convolutional Neural Network
FN	False Negative
FP	False Positive
FPN	Feature Pyramid Network
GB	Giga Bytes
GDP	Gross Domestic Product
HCDC	Hybrid Cascade Dilated Convolution
IDE	Integrated Development Environment
IoU	Intersection over Union
IPM	Integrated Pest Management
ISBI	International Symposium on Biomedical Imaging
Mask RCNN	Mask Region-based Convolutional Neural Network
ML	Machine Learning
MoA	Ministry of Agriculture
OpenCV	Open-Source Computer Vision
PR	Precision-Recall
ReLU	Rectified Linear Unit

ResNet	Residual Networks
RGB	Red, Green, and Blue
R-FCN	Region-based Fully Convolutional Networks
RoI	Regions of Interest
RPN	Regional Proposal Network
SDGs	Sustainable Development Goals
SSD	Single ShotMultibox Detector
SSD	Solid State Drive
SVM	Support Vector Machine
TFLite	TensorFlow Lite
TP	True Positive
UN	United Nations
VGG	Visual Geometry Group
VOC	Visual Object Classes
XML	Extensible Markup Language

CHAPTER ONE

INTRODUCTION

1.1 Background of the Problem

The current world population of 7.6 billion is expected to reach 8.6 billion in 2030, 9.8 billion in 2050 and 11.2 billion in 2100 (United Nations, 2017). The second Sustainable Development Goal (SDG) of the United Nations (UN) is to "End hunger, achieve food security and improved nutrition and promote sustainable agriculture" for the global population (United Nations, 2018). The explosion of the projected world population raises new demands on the quality and quantity of agricultural products and efficient use of water and other limited resources (Tzounis *et al*., 2017). The agricultural sector should be more competitive and robust to ensure global food security and meet potential demand for high quantity and quality food (Patil *et al*., 2012).

Agriculture is an important economic sector in Tanzania, contributing about 29.1% of the Gross Domestic Product (GDP) and employing 67% of the population. Agriculture is the main source of industrial raw materials, food, and foreign exchange earnings (Ministry of Agriculture [MoA], 2017). Farmers in Tanzania grow a variety of permanent and annual crops for food and economic purposes. Crops grown include cereals, roots, and tubers, fruits and vegetables. The tomato plant is grown in different parts of Tanzania with the highest production compared to other fruits and vegetables. In 2017, the total planted area for tomatoes in Tanzania was 54 520 hectares of which 50 645 hectares (that is 92.9%) were in Tanzania Mainland and 3876 hectares (that is 7.1%) were in Zanzibar (MoA, 2017). About 247 135 tonnes of tomatoes were harvested on this planted area, which is equal to 64% of all fruits and vegetables in Tanzania.

Due to their high nutritional value and health benefits, tomatoes are an essential component of most people's diet (Burton-Freeman & Reimers, 2011), and is therefore one of the most widely grown crops in the world (Rupanagudi *et al*., 2015). Tomato is considered the main source of raw materials for the tomato processing industry and can increase foreign export of a country, thereby contributing to the GDP (Arah, 2015; Çetin & Vardar, 2008). Small scale farmers and rural families rely on tomatoes to earn income for their living expenses; therefore, the crop contributes significantly to poverty reduction (Arah, 2015; Mutayoba & Ngaruko, 2018).

Currently, tomato production is threatened by the invasion of an exotic pest known as tomato leaf miner (*Tuta absoluta)* (Zekeya *et al*., 2016). In Tanzania, the pest is notoriously known as

"kantangaze". The pest spreads rapidly and it is now a serious threat to tomato production in the world (Desneux *et al*., 2011). It causes heavy losses in tomatoes ranging from 80 to 100% when proper control technologies are not employed (Chidege *et al*., 2016). *Tuta absoluta* originated in South America, then it crossed borders and spread to Europe, Middle East, Asia, and then to Africa, where it was firstly reported in 2008 in Algeria (Zekeya *et al*., 2017). The pest has moved swiftly in Africa from North to South, causing substantial and often complete loss of tomatoes both in greenhouses and open fields. Since the pest invaded African countries in 2008, it has spread into about 41 of the 54 countries in Africa, causing enormous economic losses to tomato farmers (Guimapi *et al*., 2016). In Tanzania, the first occurrence with its devastating effects of the pest on tomatoes was reported in August 2014 at Ngabobo village, Ngarenanyuki ward, Arumeru District in Arusha, and since then, it has spread to other regions (Chidege *et al*., 2016).

The pest can physiologically adapt and survive in harsh environmental conditions such as hot temperatures as high as 49 ℃ in summer, temperatures below 5 ℃, and can also tolerate dryness (Cuthbertson *et al*., 2013; Tonnang *et al*., 2015; Van Damme *et al*., 2015). Since *Tuta absoluta* is multivoltine, it can yield up to 12 generations per year and each mature female can produce between 250 and 300 eggs in its lifetime (Doğanlar & Yİğİt, 2011).

The pest transmission is through females of *Tuta absoluta* which deposit their eggs on stems, leaves, and petioles. The four development stages (egg, larva, pupa, and adult) of *Tuta absoluta* are all harmful and can infect different parts of the host plant (Guimapi *et al*., 2016). Larva, the most dangerous stage has a life span of nine (9) days before turning to pupa. It usually develops and feeds between the upper and lower epidermis in leaf mines but can also be in fruits and stems (Cuthbertson *et al*., 2013). The huge loss in tomato yields caused by *Tuta absoluta*, impels scientists to devise effective methods for managing, controlling, and overcoming the pest early.

Despite the existence of various techniques to control the pest, there has not been an effective mechanism quantifying *Tuta absoluta*'s effects at early stages before it causes yield losses to farmers. Inspired by the advancement and promising results of Deep Learning techniques in image-based plant pest and disease recognition, this research proposes a deep Convolutional Neural Network (CNN) model for early quantification of *Tuta absoluta's* damage to tomato

plants. This can enable farmers to make informed decisions in controlling the pest, improve tomato productivity, and rescue farmers from the losses they incur every year.

1.2 Statement of the Problem

Despite the efforts of the government to set aside a financial budget each year for the growth and development of the agricultural sector, farmers still face several challenges such as inadequate farming methodology, climate changes, and attacks on crops by pests and insects. The invasion of *Tuta absoluta* has extensively damaged tomato yield to the extent that growers are quitting tomato production due to the enormous costs and losses they face (Zekeya *et al*., 2017). The rot arising from secondary infection reduces tomato fruit quality, making it unfit for consumption (Food and Agriculture Organization [FAO], 2017). Figure 1 illustrates the damage caused by *Tuta absoluta* to tomatoes. Over the past few years, farmers have been using different methods in efforts to control the pest. These include the use of pheromone traps and natural enemies for monitoring the population, cultivation of resistant tomato varieties, and continual spraying of chemical pesticides, which is still the main control method (Guedes & Picanço, 2012). Excessive use of these chemicals develops resistance, has uneconomical and harmful effects on non-targeted organisms, and can also lead to irreparable damage to the environment (Materu *et al*., 2016).

However, the extension officers who provide farmers with appropriate knowledge about plant diseases and pest management are limited in numbers in Tanzania to meet farmers' demands (Maginga *et al*., 2018). Although farmers and extension officers struggle with different methods to control the pest, there has not yet been an effective way for early identification and quantification of *Tuta absoluta's* effects on tomato plants that would enhance farmers' decision making. Therefore, this study proposes a computer vision approach for quantifying the extent of damage caused by *Tuta absoluta* in tomato plants at early stages. This will enable farmers and extension officers to make informed decisions on controlling the pest and eventually increasing productivity.

Figure 1: *Tuta absoluta*'s damage on tomatoes

1.3 Rationale of the Study

The agricultural sector is the backbone of a country's economy providing basic food for individuals and raw materials for industry. Although various crops are grown in Tanzania, tomato (*Solanum Lycopersicum*) has the highest consumption, both in raw and processed forms. Tomato production is particularly socioeconomically significant because it provides jobs to women, who account for more than 60% of the labour force (Rwomushana *et al*., 2019). The damage in tomato production is also likely to negatively affect the livelihoods of small-scale farmers who are the main producers and traders of tomatoes (FAO, 2017). The increase in tomato production loss suffered by farmers due to the invasion of *Tuta absoluta*, has compelled farmers to look for effective methods to control and overcome the pest early.

This study proposes a Deep Learning model to quantify the effects of a *Tuta absoluta* in tomato plants. Determining the extent of the pest's damage early, can enable farmers and extension officers to make informed decisions on how to control the pest so as to improve tomato productivity and rescue farmers from the loss they incur every year.

1.4 Research Objectives

1.4.1 General Objective

The main objective of this research is to develop a deep learning quantification model for farmers to determine the extent of *Tuta absoluta*'s effects on tomato plants.

1.4.2 Specific Objectives

The specific objectives of this research are:

(i) To identify the requirements for developing the proposed model.

(ii) To develop a *Tuta absoluta* damage image segmentation model.

(iii) To determine the extent of tomato leaf damage at various *Tuta absoluta* levels.

(iv) To validate the developed model.

1.5 Research Questions

This research intends to answer the following questions:

(i) What are the requirements for developing a deep learning model for quantifying *Tuta absoluta'*s damage on tomato plants?
(ii) How can a *Tuta absoluta* image segmentation model be developed?
(iii) How can *Tuta absoluta's* damage to the tomato plants be quantified?
(iv) How well does the developed model perform?

1.6 Significance of the Study

The findings of this research and the development of a solution for early segmentation and quantification of *Tuta absoluta*'s damage to tomato plants can bring hope to the giving up tomato growers due to costs and losses the pest caused in tomato production. Since the solution is automated and does not require human intervention, it will reduce the workload of the limited extension officers in the country. Both farmers and extension officers will be able to understand the extent of damage in the farm due to the invasion of *Tuta absoluta* early and take appropriate measures to control the pest they spread further to a large scale. The intelligently informed decision made could improve tomato productivity and rescue farmers from the loss they incur every year.

To the research community, this enhancement will also serve as a foundation for future research into the integration of the established framework with marketing information systems, which will provide farmers with up-to-date information about selling and buying crops and fertilizers, as well as link them with nearby agrovet shops. The study also made its annotated dataset freely available to other researchers through an open access repository to facilitate further research in diagnosing *Tuta absoluta*'s damage to tomato plants (Rubanga *et al*., 2020).

1.7 Delineation of the Study

This work aimed at developing a tool for segmentation and quantification of only *Tuta absoluta*'s damage to tomato plants because it significantly affects tomato production compared to other pests and diseases. The dataset used for the development of the tool contained only colored images, that is, Red-Green-Blue (RGB) color model collected from the field. Although experimental results show that the developed tool can accurately determine *Tuta absoluta*'s severity status, further research can improve its performance and develop a decision support system that can detect different diseases in various plants and give suggestions on actions to be taken to control the disease or pest based on their severity – which is the long-term motivation of this study.

However, it is worth noting that there are some limitations to this study. The experiments in this work used insufficient annotated dataset size considering *Tuta absoluta* only not other pests and diseases as well as limited computing power, factors that may affect the performance of the model.

CHAPTER TWO

LITERATURE REVIEW

2.1 Tomato Cropping

Tomato (*Solanum Lycopersicum L.*) is one of the most widely grown and consumed crops on a global scale. Each year, about 160 million tonnes of tomatoes are produced globally (Food and Agriculture Organization Statistics [FAOSTAT], 2019). However, approximately a quarter of those 160 million tonnes are produced for processing, making tomatoes the leading processed vegetable in the world (Tomato News, 2020). In 2016, the tomato was declared the sixth most valuable crop in the world, worth US$ 87.9 billion (Rwomushana *et al*., 2019). Figure 2 illustrates the production of tomatoes in the world from 1961 to 2019. Tomato is the main source of raw materials for the tomato processing industry and may also be consumed raw, roasted, stewed, or mixed with other foods or as a sauce (Çetin & Vardar, 2008; Díez & Nuez, 2006). Due to its high nutritional value and numerous health benefits, tomato constitutes a crucial component of human diet (Burton-Freeman & Reimers, 2011). Tomato production creates an income for most producers in developing countries. In spite of the crop's numerous advantages, many challenges including pests and diseases make its production unprofitable to producers (Arah, 2015). The invasion of *Tuta absoluta* threatens tomato production resulting in significant yield losses and crop quality reduction.

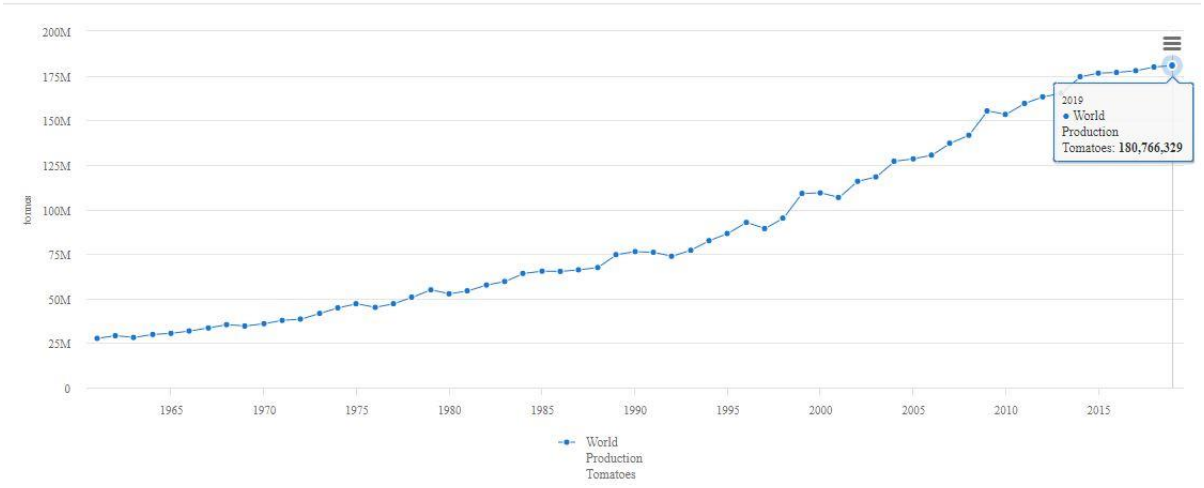


Figure 2: World production of tomatoes (FAO, 2019)

2.2 The Tomato Leaf Miner (*Tuta absoluta*)

Tomato leaf miner, *Tuta absoluta* (Meyrick 1917) (Lepidoptera: Gelechiidae) is the most devastating tomato pest in the world (Doğanlar & Yİğİt, 2011). Since its first discovery, there have been sequential genus revisions from 1917 to the 1960s. Originally known as Phthorimaea absoluta (Meyrick, 1917) , it was later transferred to the genus Gnorimoschema (Clarke, 1965), which was subsequently revised and renamed Scrobipalpula (Povolný, 1967), Scrobipalpuloides (Povolný, 1985), and finally Tuta (Povolný, 1994). The pest causes severe damage to tomato plants and substantial yield loss to farmers posing a threat to tomato production both in greenhouses and open fields. Yield losses can reach as high as 80 - 100% if no control measures are taken (Desneux *et al*., 2010). Although its primary host is the tomato, the pest can also feed and develop on other various crops in the Solanaceous family such as pepper, common beans, eggplant, potato, tobacco, and solanaceous weed (FAO, 2017; Rwomushana *et al*., 2019).

Tomato leaf miner originates from South America but has rapidly spread throughout the world as illustrated in Fig. 3. The first case of *Tuta absoluta* was recorded in Huancayo, Peru in 1917; then it spread to all South American countries between the mid-1960s and the 1990s (Soares & Campos, 2020). The pest was first detected outside of South America in eastern Spain in 2006, and in a short period, it spread quickly not only across the Mediterranean basin but also across Europe, the Middle East, Asia, and then to Africa, where it was first recorded in Algeria in 2008 (Tonnang *et al*., 2015; Van Damme *et al*., 2015). No more accurate information is available on how the pest was introduced and why it spread so quickly to new areas. But somehow, it seems to be connected to the import of tomato fruits, its ability to fly, and wind currents, though the history of invasion in Afro Eurasia indicates that *Tuta absoluta* can spread and colonize new areas rapidly without any human intervention (Desneux *et al*., 2011).

Tuta absoluta has four (4) development stages in its life cycle exhibited for about 26 – 28 days from the egg, larva, pupa to adult (Guedes & Picanço, 2012). Although larva is the most dangerous one, all four development stages of *Tuta absoluta* are harmful and can attack different parts of the host plant (Guimapi *et al*., 2016). Figure 4 shows the pest's life cycle and its damage to tomatoes. Adult females usually lay eggs on the underneath of leaves and stems; then after hatching, the larvae penetrate between the upper and lower epidermis of the leaves, fruit or stems where they feed and develop, creating conspicuous mines and galleries (Cuthbertson *et al*., 2013; Desneux *et al*., 2010). The larvae are internal feeders, being located

in the leaf mesophyll, in the apical stem, and in the fruit making it extremely hard to control, and the insecticide resistance further complicates the pest management (Doğanlar & Yİğİt, 2011; Guedes & Picanço, 2012).

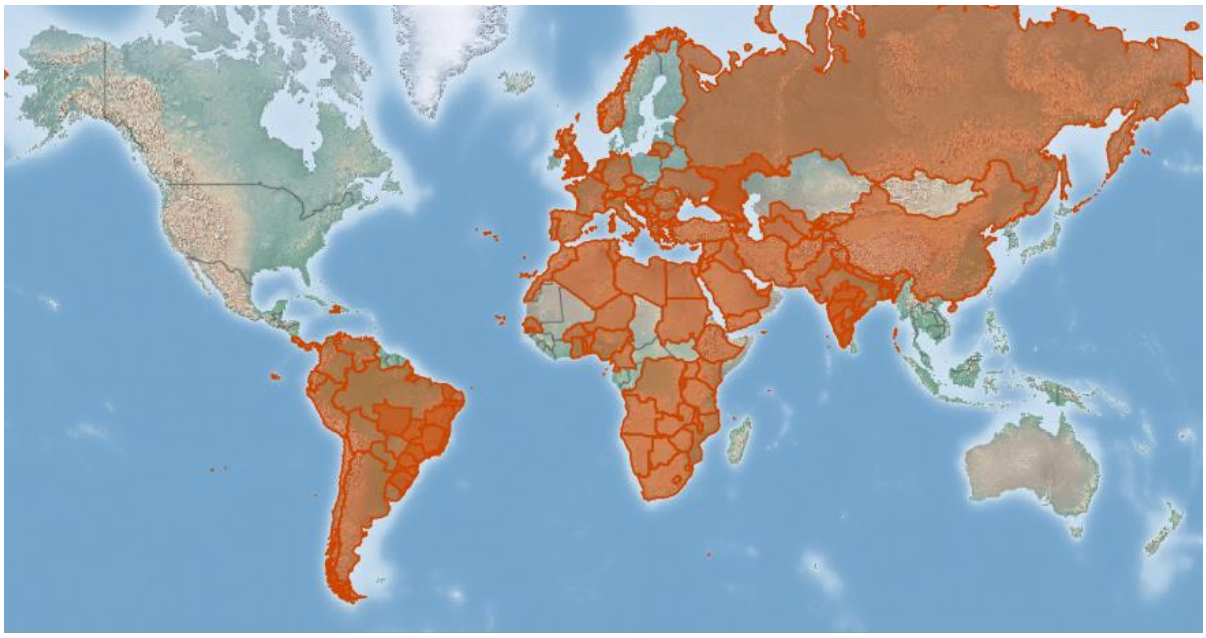


Figure 3: Worldwide distribution of *Tuta absoluta* (Soares & Campos, 2020)

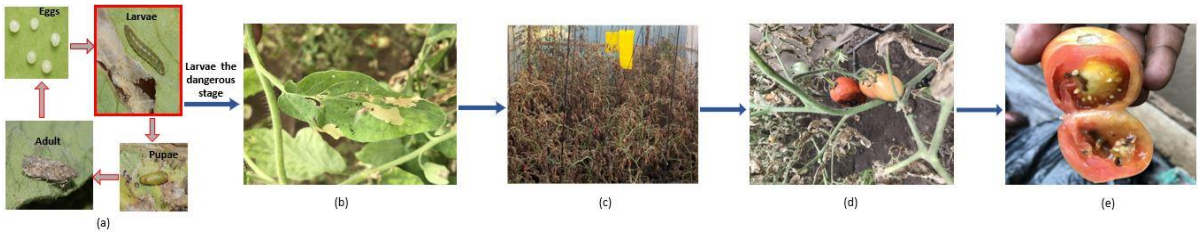


Figure 4: *Tuta absoluta's* life cycle and damage images (a) Four stages of *Tuta absoluta*'s life cycle (b) Tomato leaf with *Tuta absoluta* mines (c) Severe damage on tomato field (d) Damaged tomato fruits in the field (e) Damaged tomato fruit on the market

2.3 Computer Vision Approach

The main purpose of computer vision is to comprehend and interpret visual scenes. This involves several tasks like identifying objects in a digital image, localizing the objects, determining the objects' attributes, characterizing the relationships between objects and providing a semantic description of the visual scene. Recent advances in computer vision through the application of deep learning techniques have shown promising results in digital image processing.

Deep learning using Convolutional Neural Networks (CNNs) has a unique capability of automatically learning and extracting features based on the given dataset in their raw form without explicitly being told what features and how to extract them (Lecun *et al*., 2015; Voulodimos *et al*., 2018). The CNNs are made up of neurons that self-optimise through learning. The CNNs architecture is comprised of three types of layers stacked together. These are convolutional layers, pooling layers, and fully connected layers (O'Shea & Nash, 2015). An input image held in form of pixel values is passed through these layers to produce the final output of the class score between 0 and 1. Convolutional layers apply filters (kernels) to an input image by calculating the scalar product between their weights and the region connected to the input to create a feature map also known as activation map. Then Rectified Linear Unit (ReLU) activation function which is linear in the positive dimension but zero in the negative dimension is applied to the output of the activation produced by the previous layer. It is the source of non-linearity in CNNs. Pooling layers take feature maps as input and gradually reduce their dimensionality preserving important information. As a result, the number of parameters and the computational complexity of the model are also reduced. Fully connected layers contain neurons that are directly connected to the neurons in the two adjacent layers to classify an image. The fully connected layers attempt to generate a class score from the activations to be used for classification. Figure 5 demonstrates a common form of CNN architecture in which convolutional layers are continuously stacked between ReLus before passing through the pooling layer, and then going between one or many fully connected ReLus.

Nevertheless, in many real-world scenarios, it is very expensive or nearly impossible to collect enough training data and rebuild the CNN models from scratch. Due to the high computational resources and the large amount of labelled data required to build a model from scratch, transfer learning has become very popular and useful in deep learning. Most real-world problems lack enough labelled data points to train such complex models, so transfer learning trains deep

neural networks with comparatively small amount data (Pan & Yang, 2010). In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another.

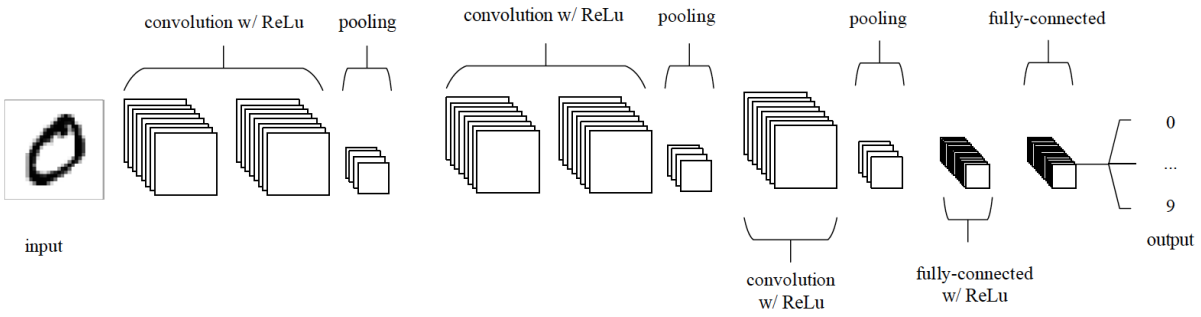


Figure 5: Convolutional Neural Network architecture (O'Shea & Nash, 2015)

2.4 Plant Diseases Diagnostics using Deep Learning

Advances in Computer Vision and Machine Learning techniques including Deep Learning, have presented promising and impressive results in identifying, classifying, quantifying, and predicting a diverse range of plant diseases and pests. Among Deep Learning methods, Convolutional Neural Networks (CNNs) have demonstrated exceptional performance in image recognition tasks (Singh *et al*., 2018).

For instance, Brahimi *et al*. (2017) presented a deep learning approach for identifying plant diseases using images of the leaves. Deep learning is capable of directly exploiting raw data without the use of handcrafted features. A dataset of 14 828 images was used to train deep CNN models based on AlexNet and GoogleNet to automatically determine nine (9) diseases and their symptoms that affect tomatoes. The model's performance was proved to be 99.185% accurate compared to shallow models like Support Vector Machine (SVM) and Random Forest.

Amara *et al*. (2017) used LeNet, a CNN architecture to automatically identify black Sigatoka and banana speckle, fungal diseases that threaten banana production. The model learns the visual features from banana leaf images and identifies the leaves affected by the two diseases and the healthy ones. In their experiment, a dataset of 3700 images was annotated into three different classes, namely banana Sigatoka, banana speckle, and healthy. The model performed well with an accuracy of 92.88%.

Also, Ferentinos (2018) used several pre-trained deep models such as AlexNet, AlexNetOWTBn, VGG, Overfeat, and GoogLeNet to identify 58 diseases from different plants. PlantVillage, an open access repository containing a dataset of 87 848 leaf images from various plants (Hughes & Salathe, 2015), was used in this study. The VGG model exhibited an excellent performance of 99.53% compared to other deep models used in this study.

The study by Zhang *et al*. (2018) proposed pre-trained CNN architectures to identify 8 tomato diseases using 5550 images from an open access repository. All the models could classify the diseases into correct classes with the accuracy of 95.83%, 95.66%, and 96.51% for AlexNet, GoogLeNet, and ResNet50, respectively.

Moreover, Fuentes *et al*. (2017) introduced the application of deep meta-architectures and feature extractors for real-time detection of different diseases and recognition of pests in tomato plants. Deep meta-architectures such as Single ShotMultibox Detector (SSD), Region-based Fully Convolutional Networks (R-FCN), and Faster Region-based Convolutional Neural Network (Faster R-CNN) combined with feature extractors like ResNet and VGG-16 were used to identify 9 different pests and diseases successfully and their location in a tomato plant. The model was trained with a dataset of 5000 images that includes nine diseases and pests, namely Cranker, Leaf mold, Plague, Gray mold, Miner, Powdery mildew, Low temperatures, Nutritional excess and Whitefly. The model showed outstanding performance with an average accuracy of 83% in recognizing the tomato diseases and pests and dealing with complex tasks including infection level, sides of the leaves and different background conditions.

Similarly, a dataset containing 4483 leaves images of different plants was used by Sladojevic *et al*. (2016) to recognize and distinguish 13 plant diseases from the healthy ones. CaffeNet, a deep CNN architecture, was trained and tested to determine and differentiate the dataset leaves images categorized into three classes; 13 infected images, healthy leaf images, and a background image class to allow good separation of plant leaves and the surroundings. The diseases recognized were, *Taphrina deformans,* powdery mildew, *Erwinia amylovora,* porosity, wilt, gray leaf spot, *Gymnosporangium sabinae,* rust, mites, downy mildew and *Venturia* in pear, cherry, peach, apple, and grapevine plants. In this study, the precision between 91% and 98% was achieved for separate class tests, and the overall accuracy of the developed model was 96.3%.

Mkonyi *et al*. (2020) proposed a VGGNet model to identify *Tuta absoluta* pest in tomato plants. In their study, three CNN architectures, namely ResNet50, VGG16, and VGG19, were used in training classifiers on a dataset of 2145 healthy and infested tomato leaf images. The VGG16 achieved a high accuracy of 91.9% on differentiating the two classes. In this study, there is still a need to detect the exact location of *Tuta absoluta*'s damage and to determine the extent of the damage.

Although several studies addresses the problem of plant leaf disease identification, few have focused on developing systems that can estimate stress severity. Liang *et al*. (2019) proposed a multitasking system called PD2SE-Net consisting of ResNet50 architecture that can diagnose diseases, recognize plant species, and estimate disease severity. A dataset from the PlantVillage repository was used to perform the experiments. The overall accuracy for disease severity estimation and plant disease classification was 91% and 98%, respectively.

In a recent study, Esgario *et al*. (2020) developed a multi-task system based on CNNs for classifying and estimating the severity of coffee leaf biotic stresses from 1747 images of *arabica* coffee leaves. Among the CNN architectures (MobileNetV2, AlexNet, ResNet50, GoogLeNet, and VGG16) used, ResNet50 had the best biotic stress classification accuracy of 95.24% and the best severity estimation accuracy of 86.51%.

Furthermore, Wang *et al*. (2017) proposed a deep learning model for automatic estimation of black rot disease severity in apple plants. A small dataset of 552 apple leaf images was used for training the VGGNet model to quantify the severity of the disease in four stages. The model performed better with an accuracy of 90.4% compared to other models employed in their study.

Additionally, Lin *et al*. (2019) proposed a segmentation model based on U-Net architecture to segment powdery mildew in cucumber plants. A dataset of 50 cucumber leaf images captured in a cucumber fruit leaf phenotype automated analysis platform was used in their experiment. The model performance was 96.08% outperforming conventional Machine Learning methods such as K-means and Random Forest.

Wang *et al*. (2019) presented a tomato disease detection model based on Faster R-CNN and Mask RCNN. The model detects and segments the locations and shapes of the infected area on tomato fruits. In their experiment, a dataset of 286 tomato fruit images obtained from the

internet was used, and the models achieved mean Average Precision (mAP) of 88.53% and 99.64% for Faster R-CNN and Mask RCNN, respectively.

Also, Pérez-Borrero *et al*. (2020) proposed a CNN model based on Mask RCNN architecture for instance segmentation of strawberries. They modified the Mask RCNN network by removing the object classifier and the bounding box regressor, replacing the non-maximum suppression algorithm with a new region grouping and filtering algorithm without increasing the complexity order. In their experiment, a dataset of 3100 strawberry images along with their annotations was used. They also proposed the Instance Intersection Over Union (I^2oU) as a new performance metric for evaluating instance segmentation. Their model achieved a mean Average Precision (mAP) of 43.85% compared to 45.36% of the original Mask RCNN and the mean I^2oU of 87.27% compared to 87.70% of the original Mask RCNN.

Tang *et al*. (2020) developed a dilated encoder network (DE-Net) model based on U-Net architecture for automatic butterfly ecological image segmentation. To capture deeper semantic features, they modified the U-Net architecture by replacing the last two pooling layers, the last three convolution layers, and all fully connected layers with the hybrid cascade dilated convolution (HCDC). A public dataset of 832 butterfly ecological images was used and the DE-Net model achieved an accuracy of 98.67%.

The study by Liu *et al*. (2020) proposed a method to segment overlapped poplar seedling leaves that are under heavy metal stress by combining Mask RCNN with Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering algorithm. Mask RCNN was used to segment leaves and then DBSCAN clustered single leaves from the detected overlapping leaves. A dataset of 2000 RGB-D images with their corresponding annotations was used to complete the task. In their experiment, the model obtained a pixel-wise Intersection over Union (p-IoU) and mean accuracy of 0.874 and 0.888, respectively.

2.5 Research Gap

Generally, these studies have achieved excellent results in image-based plant diagnosis using CNNs. However, very few works in the literature have focused on quantifying the disease's severity and to the best of the author's knowledge, there are no studies that address the quantification of *Tuta absoluta*'s effects on tomato plants. Also, some studies used a small dataset size and images from online repositories which do not reflect real field situations.

Therefore, this study proposes a deep learning-based approach for quantifying the effects of a tomato leaf miner *(Tuta absoluta)* at the early stages of the tomato plant's growth by using images collected from the field. This study will assist Tanzanian farmers and extension officers in making well-informed decisions that could increase tomato production and rescue farmers from the loss they suffer every year.

CHAPTER THREE

MATERIALS AND METHODS

3.1 Study Area

This study was conducted in Tanzania, targeting various tomato growers both in greenhouses and open fields around the country. The two regions (Arusha and Morogoro) are some of the major areas prone to *Tuta absoluta* infestation in Tanzania. The image dataset collected from the research area was used to train, test, and adjust the model accordingly so that it performs well with new data from all parts of the country. The study area is shown in Fig. 6.

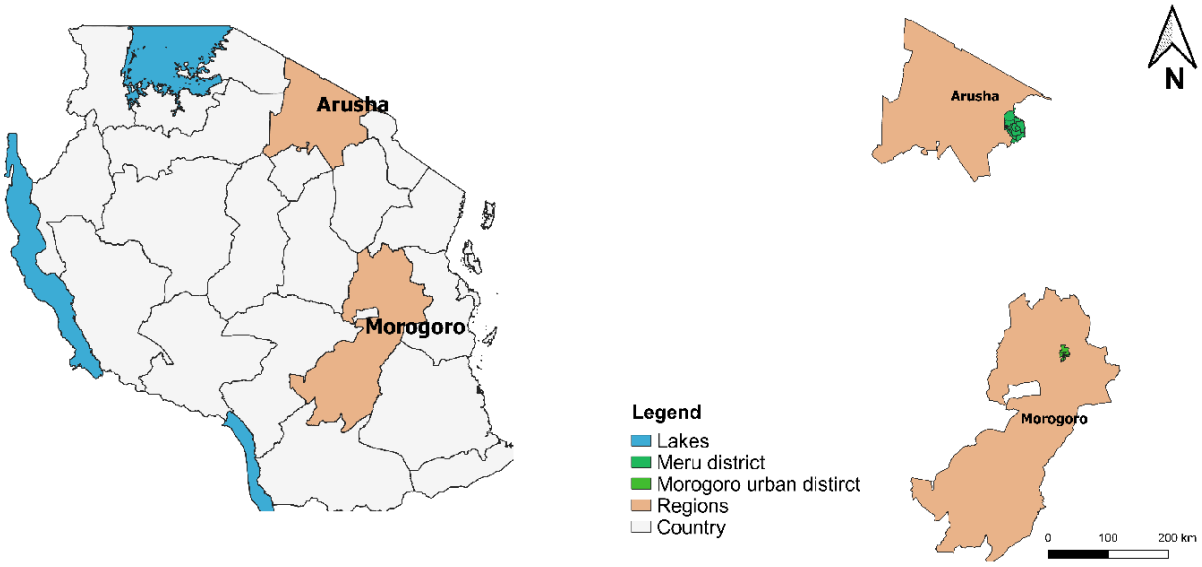


Figure 6: Research study area

3.2 Field Setup

Two (2) in-house experiments were set up in Arusha and Morogoro regions where tomato plants were grown. Net houses (Fig. 7) were constructed, one in each region considering different factors summarized in Table 1. This in-house experiment prevented any pests from coming into the net house and *Tuta absoluta* from getting out of the experimental area to maintain a controlled environment for the study.

Figure 7: Experimental setup in a field

3.2.1 Planting and Pest Introduction

Healthy tomato seedlings (free from other diseases and pests) were planted in each net house in the two regions as shown in Fig. 8. On the second day following the transplant, *Tuta absoluta* was introduced to some tomato plants by placing 2 to 8 larvae on top of the leaves of randomly selected plants. The pest immediately started to mine the leaves. Plants to infect were randomly selected so that to have a dataset of both infested and healthy tomato plants. This process was done under the supervision of an agricultural expert as shown in Fig. 9.

(a) (b)

Figure 8: Transplanting the tomato seedlings in (a) Arusha and (b) Morogoro fields

(a) (b)

Figure 9: A researcher and an agricultural expert performing infestation in the field (a) The process carried out in Arusha field (b) Process in Morogoro field

3.3 The Dataset

A dataset of 5235 tomato images was collected directly from the constructed experimental net houses. This includes 2319 images collected in Arusha and 2916 images that were collected in Morogoro. Table 1 shows factors taken into account to obtain a diverse dataset of the real field situations, such as regions in the country that are highly infested with *Tuta absoluta*, crop cycle season, commonly grown tomato varieties, and commonly practised farming systems. Images of tomato leaves at early growth stages were collected using a Canon EOS Kiss X7 camera with a resolution of 5184 x 3456 pixels for high-resolution images and a Samsung SM-G570F camera with a resolution of 320 x 240 for low-resolution images. Images of different resolutions were collected so as to train the model with images of different qualities. Since the model will be employed in the field and assuming that smallholder farmers in Tanzania use cheap phones with low resolution, the model was trained using both high and low-resolution images. The images were taken daily for two consecutive weeks after infestation focusing on capturing the top approximately 30 centimetres from the plant (Fig. 10) since the plant crown is always affected by the pest in the early growth stages of the plant.

The plants with less than 3 *Tuta absoluta* were considered to have low pest damage (Low *Tuta*), those with more than 3 *Tuta absoluta* were considered to have severe pest damage and healthy ones (No *Tuta*) as shown in Fig. 11.

Figure 10: Data collection using (a) high and (b) low-resolution cameras in Arusha and Morogoro fields

Table 1: Data collection setup and factors considered for each experiment

Duration	Season	Region	Variety	Farming system	Number of Images
Oct - Dec 2019	dry/wet	north	3	drip	2319
Jan - Apr 2020	wet	east	2	drip, furrow, bund	2916

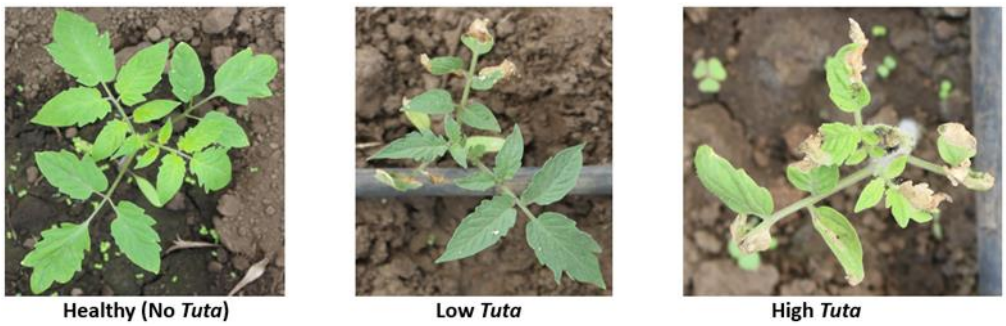


Figure 11: Some images from the field depicting the *Tuta absoluta*'s damage status

3.4 Research Framework

The research framework in Fig. 12 shows the logical flow of the study and gives a clear understanding of how the research was undertaken from data collection to model development and validation until the delivery of an optimized model. Two (2) deep meta-architectures, namely U-Net and Mask RCNN, were used to develop semantic and instance segmentation models, respectively. The segmentation model can determine the exact spot in the plant where the *Tuta absoluta* affected. Then a custom function built on top of the instance segmentation model was used to determine the extent of *Tuta absoluta* damage to the tomato plant. The model's performance is then evaluated using different evaluation metrics and the model's parameters are tuned to get an optimized model. The model is deployed on a mobile phone to enable and facilitate farmers to automatically detect affected areas on tomato plants.

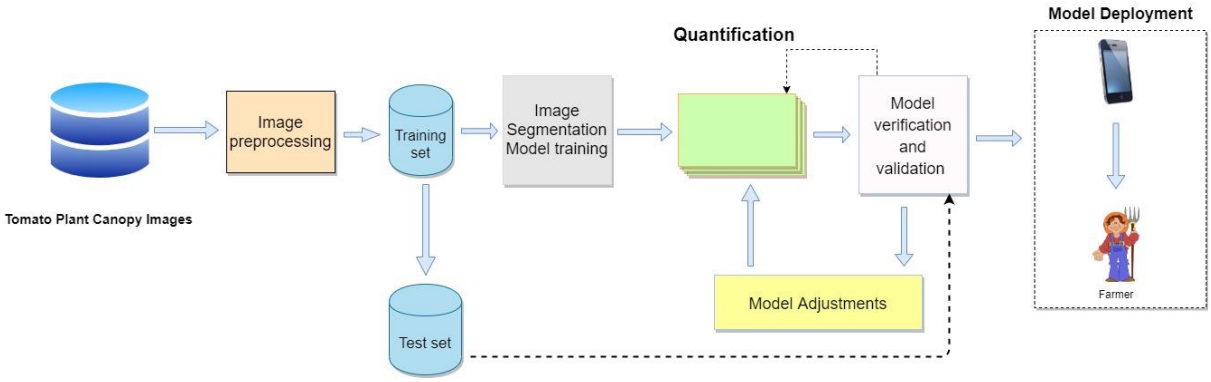


Figure 12: Research conceptual framework

3.5 Image Preprocessing

Image preprocessing is an important stage for enabling a deep learning architecture to learn and extract features from an image during model training. It is a manipulation of raw image data before feeding it to a deep model for better performance. In this work, the image preprocessing process involved labelling and cropping, image annotation, image resizing, and augmentation. The preprocessed images were then used as input data to the model.

3.5.1 Labelling and Cropping

The images were manually labelled to distinguish between healthy and infected plants by *Tuta absoluta*. This helps to make a clear distinction between the classes. The label included the block in the net house where the plant is located, the date the image was captured, the plant number, and the infestation status of the plant (healthy or non-healthy). In this study, 2319 images (1107 healthy and 1212 infested with *Tuta absoluta*) collected in Arusha and 2916 images (1870 healthy and 1046 infested with *Tuta absoluta*) collected in Morogoro were labelled, making a total of 5235 tomato images in the dataset as shown in Table 2. Then the images were manually cropped to removed unwanted objects on the background. Figure 13 shows some images from the dataset after the labelling and cropping process. It also shows the development of *Tuta* mines on different days.

Table 2: Dataset distribution

Region	Healthy	Non-Healthy	Total Dataset
Arusha	1107	1212	2319
Morogoro	1870	1046	2916
Total	**2977**	**2258**	**5235**

Figure 13: Labelled and cropped images from the dataset showing the development of *Tuta* mines on different days

3.5.2 Image Annotation

It is necessary to annotate the images indicating the area of interest in order to train a segmentation model. A total of 1212 and 1240 images with infested tomato plants were selected from the entire dataset collected from the field to be used in developing semantic and instance segmentation models, respectively. For each image, a ground truth labelled image was manually generated containing the individual segmentation of all the *Tuta absoluta's* mines present in the image. Labelme (Russell *et al*., 2008) and VGG Image Annotator (Dutta & Zisserman, 2019) open-source tools were used to annotate images for semantic and instance segmentation tasks respectively as revealed in Fig. 14. The specific operation was to define the continuous contour of all *Tuta absoluta*'s mines by marking the area and shape of the infested spot with irregular polygons and then labelling the spot with "*tuta*" as shown in Fig. 15. Each image contained at least one *tuta* mask indicating the presence of the *Tuta absoluta*'s mine in the image. The annotations obtained were saved in the VOC (Everingham *et al*., 2010) format and COCO (Lin *et al*., 2014) format with their corresponding images to be used in the semantic and instance segmentation tasks, respectively. The image annotation process with its outputs is illustrated in Fig. 16.

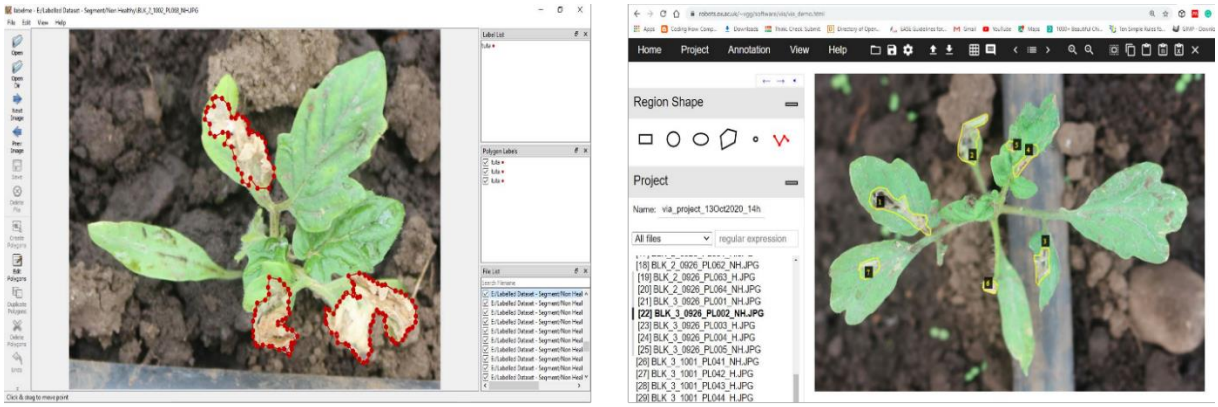


Figure 14: Examples of the two data annotation methods (a) Labelme annotation tool (b) VGG Image Annotator (VIA) tool

Figure 15: Image annotation process

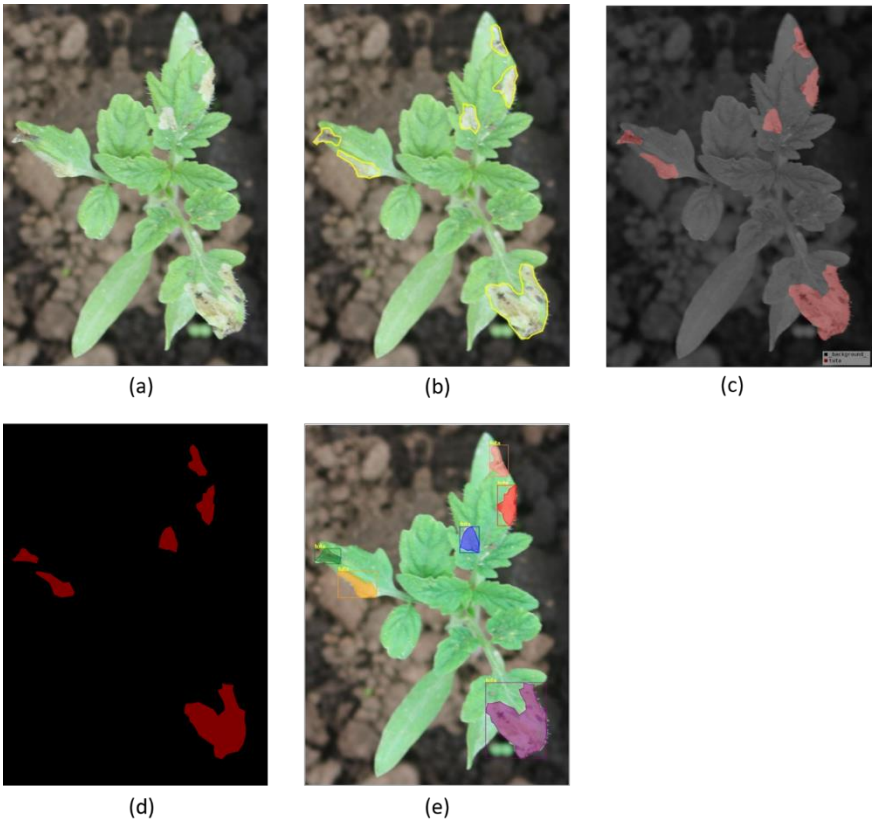


Figure 16: Illustration of the image annotation process (a) Original image (b) Polygonal annotation of the *Tuta* mines contour (c) Visualization of labels (d) Extraction of the mask (e) Original image with the overlapping mask

3.5.3 Resizing the Images

Image resizing is an important preprocessing step in computer vision. Principally, the CNN models train faster on smaller images. A large input image requires the neural network to learn from many image pixels adding up the training time and other computational costs. Therefore, many CNN architectures require that the input images are of the same size. Images in the dataset were varying in size so we used a standard resize function in Keras to resize all images to the dimensions of 512 x 512 pixels. When the dimension of an image is smaller than 512, the image is upscaled by adding zero paddings as necessary to obtain a square image.

3.5.4 Augmentation

Deep neural networks need a large amount of training data to achieve good performance and avoid overfitting. Overfitting refers to the phenomenon when a network perfectly models the training data but fails to generalize on unseen data (Lawrence & Giles, 2000). Unfortunately, we were not able to collect enough data to sufficiently train a CNN model. Data augmentation

is a solution to the problem of limited data. Image data augmentation encompasses a suite of techniques that can be used for artificially expanding the size and enhancing the quality of the training set by creating modified versions of the original images in the dataset (Shorten & Khoshgoftaar, 2019). The techniques include random rotation, shifts, shear, zooming, and flips. In this study, we performed image augmentation for some experiments. Specifically, the following set of augmentation techniques was applied to the training set only with data values in a range of (0, 1):

(i) **Horizontal flip:** All images in the training set were horizontally flipped with a probability of 0.5.

(ii) **Vertical flip:** All images in the training set were vertically flipped with a probability of 0.2.

(iii) **Crop:** A random crop was applied on images with the interval of (0, 0.1). That is, each image is cropped by 0 – 10% of its height/width.

(iv) **Gaussian Blur:** A gaussian blur with a probability of 0.5 was applied to images with a random sigma of between 0 and 0.5.

(v) **Contrast Normalization:** Contrast normalization was applied to strengthen or weaken the contrast in each image in the interval (0.75, 1.5).

(vi) **Gaussian Noise:** Gaussian noise with a probability of 0.5 was added to images. For 50% of all images, the noise is sampled once per pixel. And for the remaining 50% of all images, the noise is sampled per pixel and channel. This changes the colour of the pixels.

(vii) **Brightness modification:** A change of brightness was applied with a probability of 0.2 and a random value in the interval (0.8, 1.2) is chosen. This made some images brighter and some darker.

(viii) **Transformation:** A series of random affine transformations were applied to each image. This implies scaling or zooming images to 90 – 110% of their height/width (each axis independently). Translate/move them by -20 to +20 relative to their height/width per axis. Rotate images by -5 to +5 degrees and slightly shear them by -2 to +2 degrees.

The images generated from the aforementioned data augmentation techniques are presented in Fig. 17 and Fig. 18.

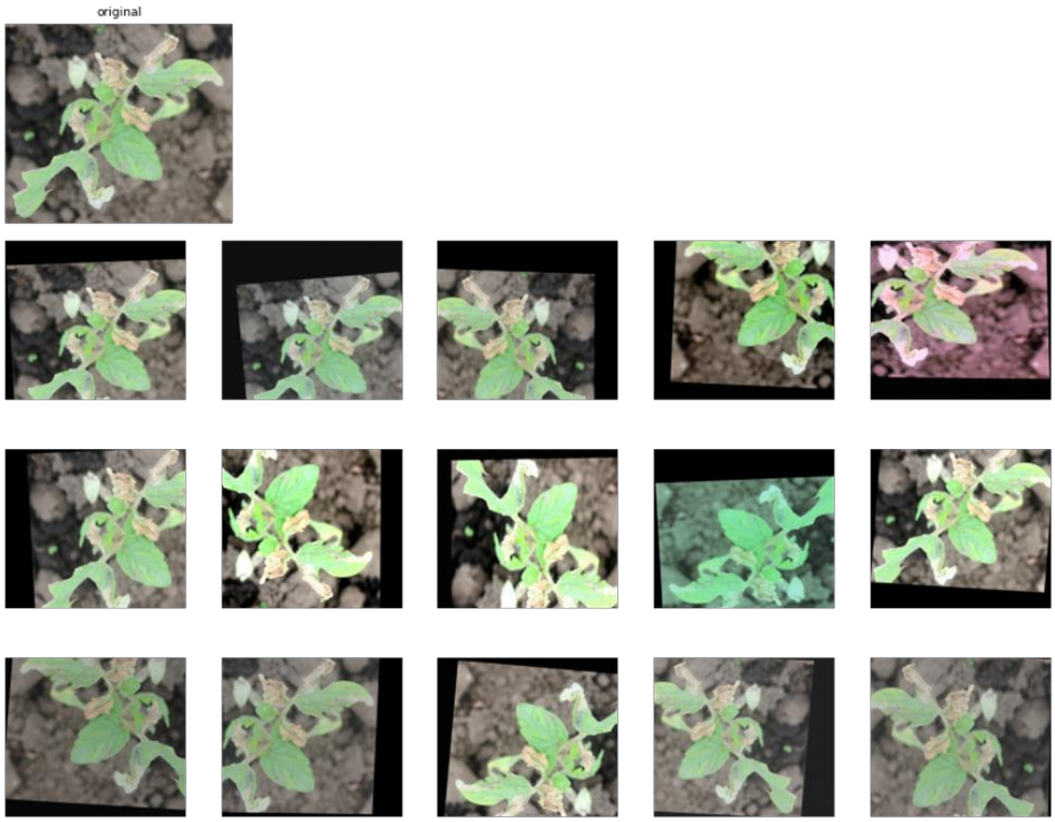


Figure 17: Original image (top) and the results of the data augmentation techniques

Figure 18: Mask augmentations

3.6 Transfer Learning

Transfer learning involves reusing pre-trained models for a specific task and fine-tuning them to a new related task, usually with a limited amount of data (Weiss *et al*., 2016). The transferring of information from a related domain improves a new model. This work employs transfer learning based on the CNN models, Mask RCNN and U-Net that have been trained and shown best performance on COCO (Lin *et al*., 2014) and International Symposium on Biomedical Imaging (ISBI) (Ronneberger *et al*., 2015) datasets for instance and semantic segmentation tasks respectively.

3.6.1 U-Net for Semantic Segmentation

Semantic segmentation classifies each pixel in an image from a predefined set of classes. The image is divided into different segments, each representing a distinct entity. In semantic segmentation, different instances of the same object are not distinguished they are given the same label. Ronneberger *et al*. (2015) introduced a U-shaped CNN architecture that is designed to be trained end-to-end with very few images and yet produce more precise segmentations. This makes it very suitable for the agricultural field since in the real world, there is not enough labelled data to train complex CNN architectures (Lin *et al*., 2019). The model has performed exceedingly well first in the biomedical image segmentation and later in many other fields outperforming the earlier segmentation methods (Ciresan *et al*., 2012). The U-Net architecture consists of three sections namely, a contraction section (also known as encoder), a bottleneck section, and an expansion section (also known as decoder) hence the name encoder-decoder structure. The encoder which is basically a stack of convolutional and max-pooling layers downsamples the input image and captures its context. It outputs a tensor that contains information about the object, its shape and size. The decoder which contains upsampling layers, takes this information and uses transposed convolutions to produce segmentation maps. This upsampling process makes the network's output the same size as the input image achieving pixel-level segmentation. The bottleneck section mediates between the encoder and decoder sections. It uses skip connections to concatenate the intermediate outputs of the encoder with the inputs to the intermediate layers of the decoder at appropriate positions. This concatenation process enables the precise localization of the target objects. The U-Net architecture is described in Fig. 19.

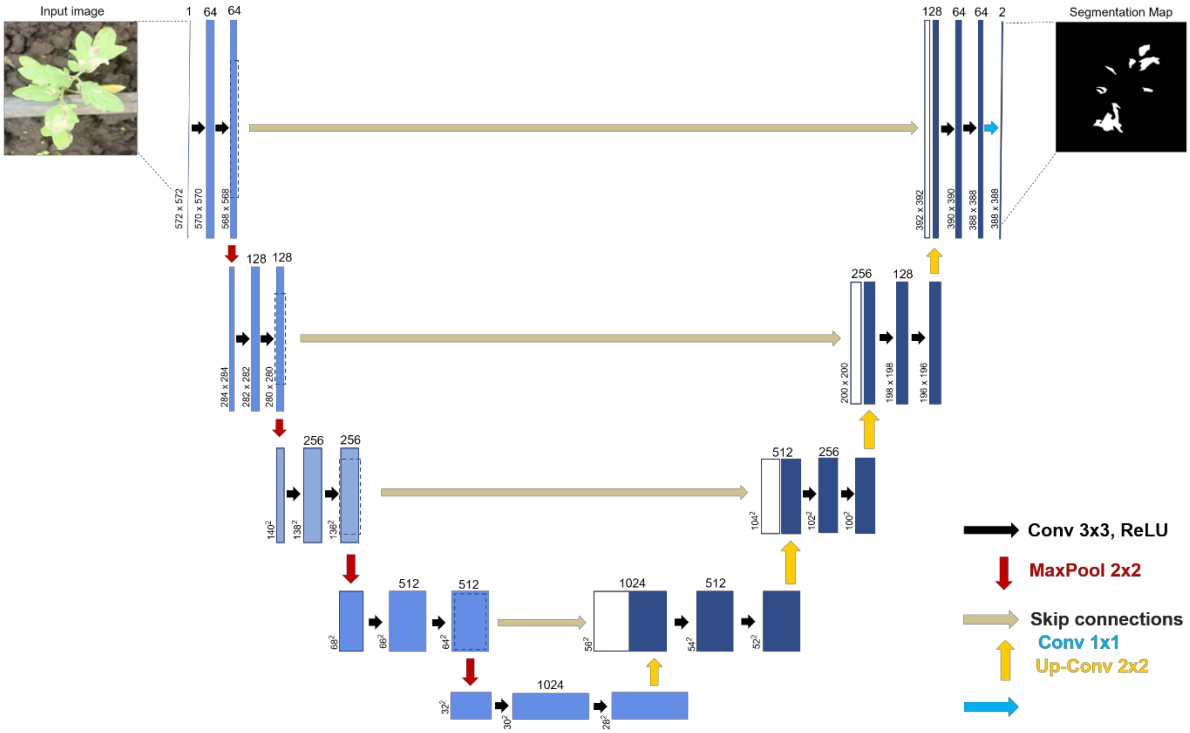


Figure 19: U-Net architecture

3.6.2 Mask RCNN for Instance Segmentation

Instance segmentation distinguishes each object instance of each pixel for every known object within an image. It integrates an object detection task that aims to detect the object class as well as predict the bounding box in an image, with a semantic segmentation task that classifies each pixel into pre-defined categories. Therefore, instance segmentation enables us to detect objects in an image while precisely segmenting a mask for each object instance.

A Mask Region-based Convolutional Neural Network is a deep neural network for instance segmentation that takes an input image and outputs a bounding box, label, and the corresponding mask (He *et al*., 2018). Basically, it is an extension of the Faster RCNN model which has two outputs for each candidate object, namely, a class label and a bounding-box offset (Ren *et al*., 2017). Mask RCNN adds a third branch that outputs the object mask decoupling class prediction and mask generation. This makes it an effective algorithm for more challenging instance segmentation tasks. Instance segmentation is a challenging task because it necessitates accurate detection of all objects in an image while precisely segmenting each instance. The architecture of the proposed Mask RCNN model is illustrated in Fig. 20.

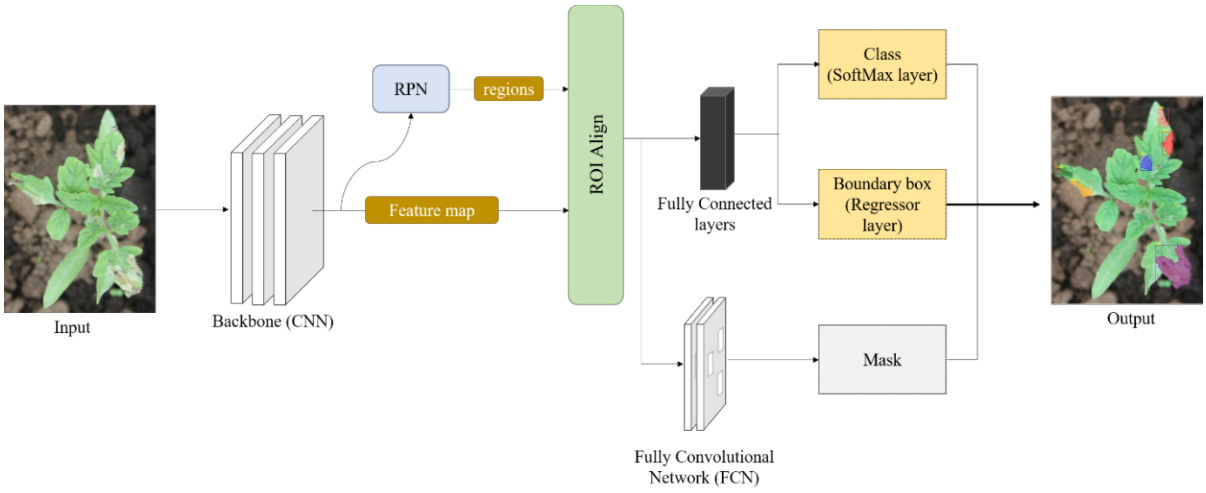


Figure 20: Proposed Mask RCNN model architecture

(i) Backbone

The CNN backbone architecture is used to extract features from an entire image. Mask RCNN uses ResNet50 and ResNet101 for feature extraction. The extracted features act as an input for the next layer. The backbone feature maps at different layers are shown in Fig. 21.

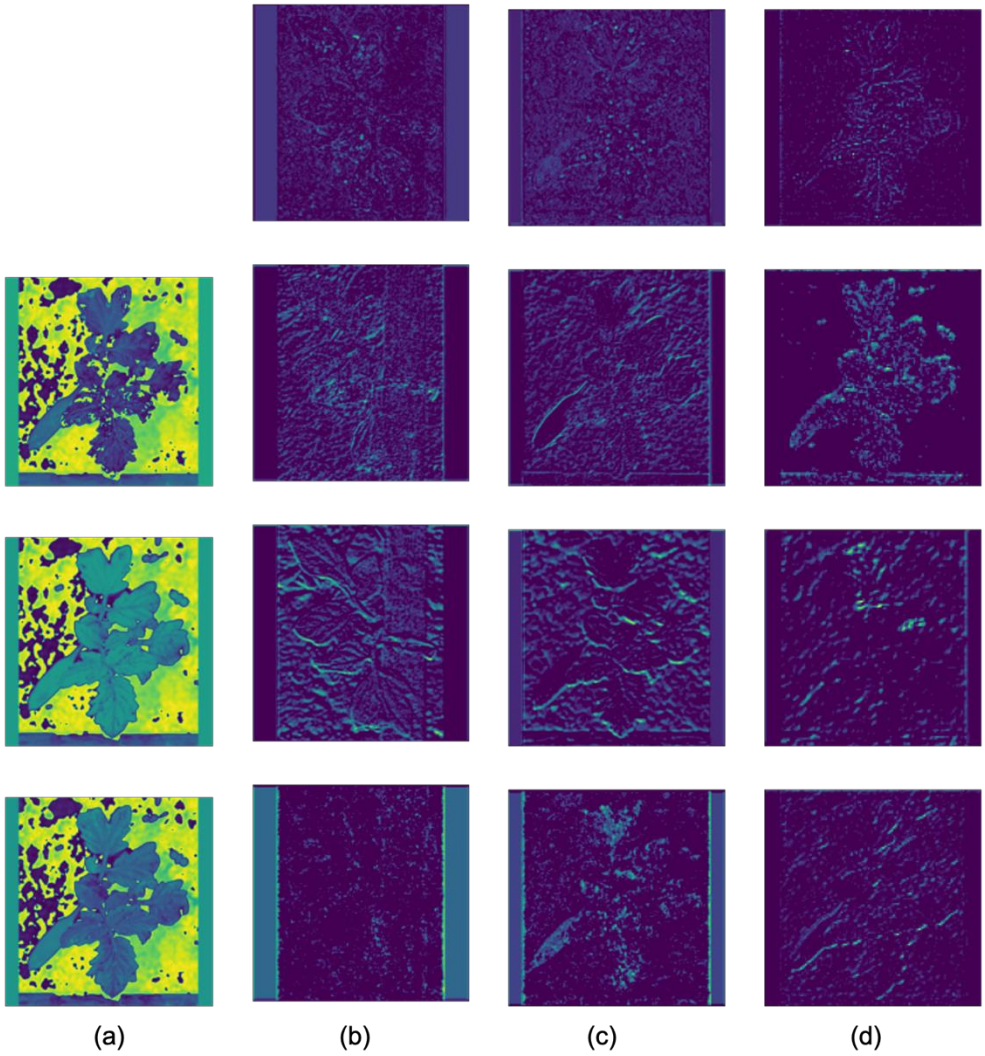


Figure 21: Backbone feature maps at (a) input layer, (b) res2c_out activation layer, (c) res3c_out activation layer, and (d) res4c_out activation layer

(ii) Region Proposal Network (RPN)

The Region Proposal Network is applied to the feature maps obtained in the previous step and outputs a set of object/region proposals, i.e., Regions of Interest (RoIs), each with its objectness score. To generate region proposals, RPN uses a sliding window over the convolutional feature maps producing anchor boxes of different shapes and sizes. Then for each anchor box, the RPN predicts the probability that an anchor is an object (i.e., objectness score) and the bounding box regressor for adjusting the anchors to best fit the object. Figure 22 illustrates how the RPN works. Using the Non-Maximum Suppression (NMS) technique, the RPN refine anchors with a high objectness score and suppress or reject all other boxes. The RPN regions of interest and anchors are shown in Fig. 23. The output of this step is the feature maps or regions that the model predicts to contain some objects.

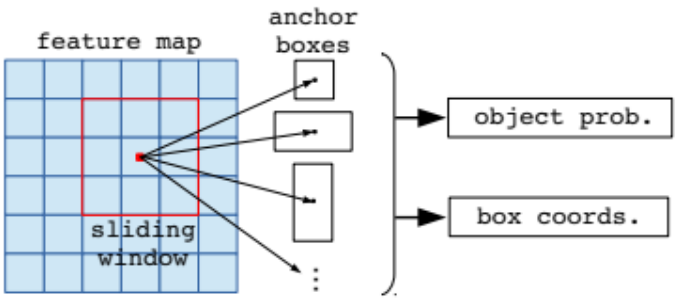


Figure 22: Illustration of how RPN works (Pawang, 2020)

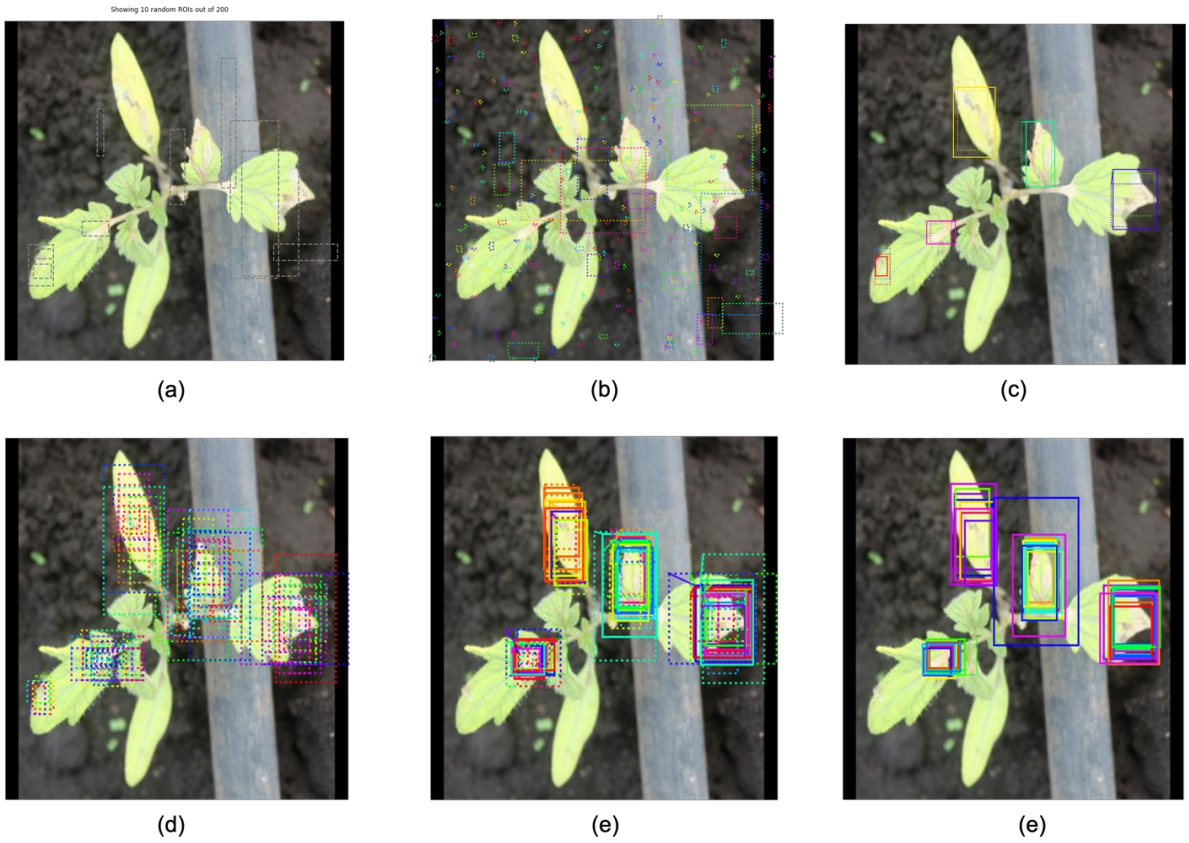


Figure 23: The RPN anchors (a) Regions of Interest (RoIs) (b) Negative anchors (c) Positive anchors (d) Top anchors before refinement (e) Top anchors with refinement (f) Refined anchors after non-max suppression

(iii) Regions of Interest (RoI) Align

Both RoIs and their corresponding feature maps from the previous step are passed through the RoI Align layer which converts them to a fixed shape and size. The RoIAlign uses binary interpolation to generate a small feature map of fixed size (e.g., 7 x 7) from each RoI. The RoI Align layer properly aligns the extracted features with the input and accurately maps RoIs from the original image onto the feature map without rounding up to integers.

(iv) Fully Connected Layers

On top of the fully connected network, a softmax layer is used to predict classes in the image. The softmax layer applies a softmax function to the input to assign decimal probabilities to each class which must add up to 1. A linear regression layer is also used alongside the softmax layer to output bounding box coordinates for predicted classes.

(v) Fully Convolutional Network

The output of the ROI Align layer also goes separately to the convolutional layer to predict the mask. This ConvNet takes an RoI as input and outputs the m*m mask representation. The mask shape normally is 28 x 28.

3.7 Implementation

The experiments were carried out on a computer, pre-installed with Windows 10 equipped with one Intel® Core™ i7-8550U 3.6 GHz CPU, Intel® Iris® Plus Graphics, 512 GB SSD storage, and 16 GB memory. Google Collaboratory with Tesla P100-PCIE GPU and 27 GB memory was utilized. To implement this work, python (Travis, 2007) programming language with Keras (Chollet, 2017) library and TensorFlow (Abadi, 2016) as backend were used.

3.8 Training Phase

3.8.1 U-Net: Hyperparameters Tuning and Network Training

A U-Net architecture was used in this implementation to develop a *Tuta absoluta* semantic segmentation model. In this custom U-Net, 32 convolutional filters were set in the initial convolutional block which will be doubled after every block while setting 4 layers in the encoder path. Since the problem in this study was binary segmentation, sigmoid was set as the activation function in the output layer. All images in the dataset were rescaled to a range of (0 – 1) and then resized to a dimension of 512 x 512 pixels. Given the insufficient size of our dataset to effectively train a CNN architecture, random data augmentation was performed to expand the training dataset size. The augmentation techniques included horizontal and vertical flipping, the zoom range of 0.2, shear in a range of 40, width and height shift in a range of 0.05, and rotation in a range of 5.0 degrees. The images with their corresponding annotations were transformed in the same way. Figure 24 shows the generated images with their corresponding annotations after augmentation.

The network was trained using 200 epochs with a learning rate of 0.01 and Adam (Kingma & Ba, 2014) as the optimization function. The IoU threshold for minimum detection probability is kept at 0.5.

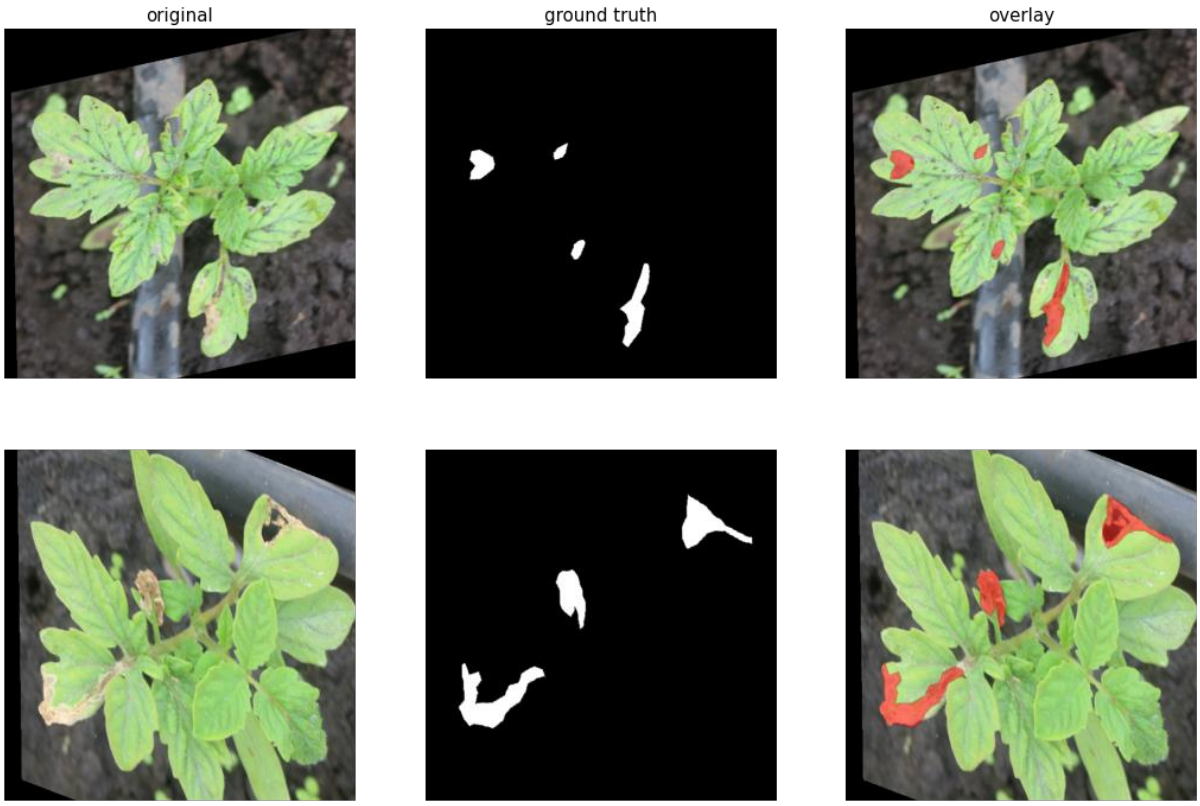


Figure 24: Augmented images with their corresponding annotations

3.8.2 Mask RCNN: Hyperparameters Tuning and Network Training

This implementation is based on an open-source Mask R-CNN by Matterport with Massachusetts Institute of Technology (MIT) license (Abdulla, 2017), built on ResNet101 and Feature Pyramid Network (FPN) as a backbone. Two CNN architectures, namely ResNet50 and ResNet101, were used separately as backbone architectures of proposed Mask RCNN model. Images are resized in square mode to a minimum dimension of 800 pixels and a maximum dimension of 1024 pixels. Since the inference is ran on one image at a time, the batch size was set to 1 where each batch has 1 image per GPU. A learning rate of 0.001, weight decay of 0.0001 and learning momentum of 0.9 have been used in this implementation. The minimum detection probability is kept at 0.7 so that RoIs with scores larger than this threshold are kept and below that are skipped. The training was developed during 200 epochs, and a model was evaluated on the validation set at the end of each epoch. Configurations used to train the Mask RCNN model are summarized in Table 3.

Table 3: Configurations used for training Mask RCNN model

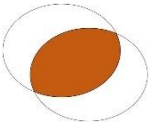
Hyperparameter(s)	Value(s)
Backbone	ResNet50 or ResNet101
Backbone Strides	[4, 8, 16, 32,64]
Batch Size	1
Detection Maximum Instances	100
Detection Minimum Confidence	0.7
Detection NMS Threshold	0.3
GPU Count	1
Images per GPU	1
Image Maximum Dimension	1024
Image Minimum Dimension	800
Image Resize Mode	Square
Image Shape	[1024 1024 3]
Learning Momentum	0.9
Learning Rate	0.001
Mask Shape	(28, 28)
Number of Classes	2
RPN Anchor Scales	(8, 16, 64, 128, 256)
RPN Anchor Stride	1
RPN NMS Threshold	0.9
Weight Decay	0.0001

3.9 Evaluation

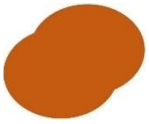
Model evaluation is a measure of how the trained model generalizes on new previously unseen data. It aims at estimating the generalization accuracy on new data. The performance of a deep learning model can be evaluated using different evaluation metrics. The choice of evaluation metrics depends on a given deep learning task (such as classification, localization, among others). In this work, the performance of instance and semantic segmentation models was evaluated using Intersection over Union, Dice Coefficient, precision, recall, and mean Average Precision (mAP) as defined below.

3.9.1 Intersection over union (IoU)

Intersection over Union also known as Jaccard Index, is a metric that evaluates how similar the predicted bounding box or mask is to the ground truth bounding box or mask. It is basically the ratio of the area where the two boxes or masks overlap (intersection between predicted box or mask and actual box or mask) to the total area of the two boxes or masks (their union). A prediction is considered to be True Positive (TP) if the IoU is greater than a defined threshold and False Positive (FP) if the IoU is less than a given threshold. The equation below illustrates the calculation of IoU.



$$IoU = \frac{Overlapping\ Area}{Union\ Area} \tag{3.1}$$



This metric ranges from 0 – 1 with 0 indicating that there is no overlap between the masks or bounding boxes and 1 signifying that there is a perfect overlap between the bounding boxes or masks as illustrated in Fig. 25. The IoU = 1, if the prediction is perfectly correct. The lower the prediction result, the lower the IoU value.

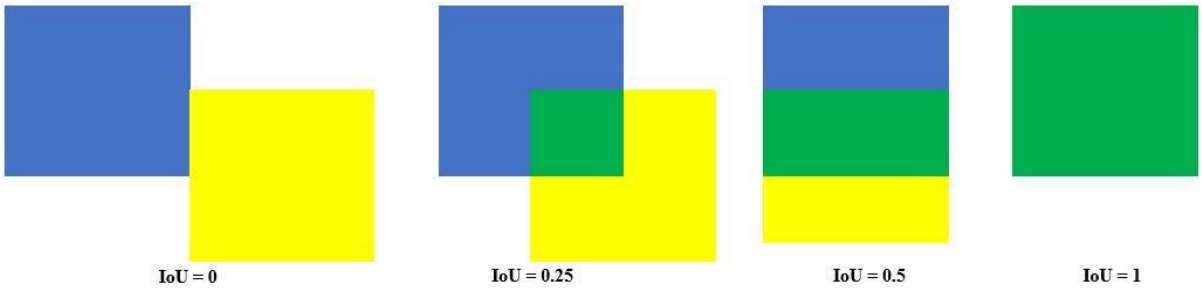


Figure 25: Illustration of bounding boxes or masks overlaps and their corresponding IoU values

3.9.2 Dice Coefficient (F1 Score)

Dice Coefficient also known as F1 Score, refers to the measure of overlap between ground truth and predicted masks. It is quite similar to Jaccard's index but doubles the count of the intersections (TPs). It is 2 times the area of overlap divided by the total number of pixels in both images. Like IoU, Dice Coefficient ranges from 0 to 1 with 1 indicating a perfect overlap while 0 indicates no overlap between the predicted and ground truth masks.

Dice coefficient is defined as:

$$Dice\ Coefficient = \frac{2 \times Intersection}{Union\ +\ Intersection} = \frac{2 \times}{+} \quad (3.2)$$

3.9.3 Precision

Precision is the measure of the percentage of correct positive predictions among all predictions made. That is, of all positive predictions, how many predictions are True Positives? To get the precision value, the ratio of True Positives to the total number of positive predictions is calculated. Precision is defined as:

$$Precision = \frac{TP}{TP + FP} \quad (3.3)$$

Where

TP (True Positive) is the number of positive samples correctly predicted as positive i.e., the number of correctly detected *tuta* mines.
FP (False Positive) is the number of negative samples that are wrongly predicted as positive i.e., the number of falsely detected *tuta* mines.

3.9.4 Recall

Recall is measuring the percentage of correct positive predictions among all actual positive cases. That is to say, of all actual positives, how many are True positive predictions? To get the recall value, the ratio of the True Positives to the total number of all samples that should have been identified as positive is calculated.

Recall is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

Where

TP (True Positive) is the number of positive samples correctly predicted as positive i.e., the number of correctly detected *tuta* mines.
FN (False Negative) is the number of negative samples that are correctly predicted as negative. i.e., the number of missed/undetected *tuta* mines.

3.9.5 Mean Average Precision (mAP)

Mean Average Precision is used as the primary evaluation metric to measure the quality of the segmentations obtained by the model. It provides the average precision of object locations in all predictions matching to ground-truth objects giving each object equal importance.

Mean Average Precision (mAP) is defined as:

$$mAP = \frac{1}{N}\sum AP \tag{3.5}$$

Where

mAP is the mean Average Precision of all classes.
AP is the Average Precision.
$\sum AP$ is the sum of the Average Precision values.
N is the number of all classes.

The Average Precision (AP) is defined by the area under the Precision-Recall (PR) curve, the x-axis being recall and the y-axis being precision. To plot the graph, multiple precision-recall value pairs are obtained by setting an IoU threshold value. Any detection with an IoU value below the set threshold is treated as a FP and TP otherwise. Calculating the precision and recall at each detection sorted by the threshold and after going through all precision-recall value pairs, the precision-recall graph is obtained.

3.9.6 Loss Function

The loss function is used to optimize the parameter values in a CNN model. It maps a set of network parameter values to a scalar value that shows how well those parameters perform the role that the network is designed to do. The value calculated by the loss function is simply referred to as "*loss*". That is to say, the loss function is a method of evaluating how well the algorithm models/fits the dataset. If a model's performance is good, the loss function will output a lower number and vice versa. Loss enables one to understand how much the predicted value differs from the actual value.

(i) U-Net Loss Function

The U-Net uses a pixel-wise cross-entropy loss that examines each pixel individually compared to the ground truth pixel then averaged over all pixels. Each pixel of the network's output is compared with the corresponding pixel in the ground truth segmentation image. In their original paper, (Ronneberger *et al*., 2015) states that "*The energy function is computed by a pixel-wise soft-max over the final feature map combined with the cross-entropy loss function".* That is, pixel-wise softmax is applied to the output image followed by the standard cross-entropy loss function. This loss weighting scheme helps the U-Net model segment *tuta* mines in tomato leaf images in a discontinuous fashion such that individual *tuta* mines can be easily identified within the binary segmentation map.

The loss is defined as:

$$L = \sum_{i=1}^{m} -\left(y_i log(p_i) + (1 - y_i) log(1 - p_i)\right) \quad (3.6)$$

Where

L is the total loss in U-Net.

m is the number of pixels in an image.

i is the index of a pixel.

y_i is the binary indicator i.e., the ground truth or real value of the i-th pixel whose value is 0 or 1.

log is the natural log.

p_i is the predicted probability/value of the i-th pixel. Its value ranges from 0 to 1.

(ii) Mask RCNN Loss Function

The loss function is defined as a complex multi-task loss function which is calculated as the weighted sum of various losses at each stage of Mask RCNN model training. This comprises of three (3) losses, namely loss due to classification, regression, and mask prediction. The regression and mask loss are only applied to positive examples.

The total loss is defined as:

$$L_T = \sum_i L_{cls}(p_i, g_i) + \sum_i g_i L_{reg}(t_i, t_i^*) + \sum_i g_i L_{mask}(m_i, m_i^*) \quad (3.7)$$

Where

$L_T = L(\{p_i\}, \{t_i\}, \{m_i\})$ is the total loss in Mask R-CNN.

i is the index of an anchor.

p_i is the predicted probability of an anchor *i* being an object.

g_i is the ground-truth probability of anchor *i*. Ground-truth label g_i is 1 if the anchor is positive and is 0 otherwise.

$t_i = \left(t_i^x, t_i^y, t_i^h, t_i^w\right)$ is a vector with the horizontal and vertical coordinates of the centre point as well as the height and width coordinates of the predicted bounding box.

t_i^* is a vector representing four (4) parameterized coordinates (x, y, h, w) of the ground-truth bounding box associated with a positive anchor *i*.

L_{cls} is the classification loss.

L_{reg} is the regression loss. The term $g_i L_{reg}$ means that regression loss is only activated for positive anchors ($g_i = 1$) and is disabled otherwise ($g_i = 0$).

L_{mask} is the mask loss. The term $g_i L_{mask}$ means that mask loss is only activated for positive anchors ($g_i = 1$) and is disabled otherwise ($g_i = 0$).

3.10 Model Deployment

This refers to the integration of a machine learning model into an existing production environment such as web applications, mobile applications or IoT systems to make practical decisions based on data. The proposed model was deployed into a mobile phone to enable and facilitate farmers and extension officers to automatically detect affected areas on tomato plants using their smartphones.

Since CNN models are complex and heavy, requiring a lot of memory and storage size to run, the proposed model was converted into a lighter format using the TensorFlow Lite framework. TensorFlow Lite (TFLite) is a set of tools designed to execute TensorFlow models efficiently on mobile, IoT and other embedded devices with limited computing and memory resources. Converting models reduce their file size, increase execution speed, and introduce optimizations that do not affect accuracy. Appendix 3 describes the TFLite converter used in this work.

3.10.1 Requirement Analysis

This phase focuses on determining the conditions and services the mobile application should meet and provide respectively. It encompasses the set of tasks that lead to an understanding of what the mobile app impact will be, what the end-user wants and how they will interact with the application software. The main requirement of this research was tomato leaf images collected using high and low-resolution cameras to develop a CNN model for detecting areas affected by *Tuta absoluta* pest. Data collected from the aforementioned methodologies and field experience were used to obtain functional and non-functional requirements.

(i) Functional Requirements

Functional requirements define the services that a software must offer. They describe the specific behaviors between inputs and outputs. The functionalities of the developed mobile application include the following:

(a) Capturing images.
(b) Uploading images from the gallery.
(c) Providing general information about tomatoes and *Tuta absoluta* pest.
(d) Embedding CNN model for running inference on captured or uploaded images to segment *tuta* mines on tomato plants.
(e) Displaying segmentation results.

(ii) Non-functional requirements

Non-functional requirements specify criteria that can be used to evaluate the operation of a system. They describe the system's operational capabilities as well as constraints that enhance its functionality. Non-functional requirements of the developed mobile application include the following:

(a) **Availability:** The developed mobile application operates offline thus can be available all the time once downloaded from Google Playstore.
(b) **Reliability:** The application is reliable since it fulfills its assigned tasks, which include accurately segmenting *tuta* mine on tomato leaf images.
(c) **Usability:** The developed mobile application is simple and easy to use without guidance.
(d) **Performance:** The developed mobile application has good performance since it takes only 5 seconds to run inference on a captured tomato leaf image.
(e) **Compatibility:** The application is compatible with all mobile devices installed with the Android operating system.

3.10.2 Assumptions and Dependencies

The development of a *Tuta absoluta* segmentation mobile application was based on the following assumptions for software and hardware features:

(i) The mobile application will be available and operate offline.
(ii) The project will follow agile methodology The users of the software will be smallholder farmers and extension officers.
(iii) Most of the smallholder farmers who will be the primary users of the developed mobile application are poor and use cheap smartphones installed with the Android operating system.
(iv) The users have a good knowledge of smartphones and can well interact with installed application software.
(v) The project will follow agile methodology throughout execution.

3.10.3 Use Case Modelling

A use case model describes how various types of users interact with the system, their expectations and the system's necessary actions to achieve these goals. It depicts the use cases, actors and the relationships between them. The use case diagram of the *Tuta absoluta* segmentation mobile application is shown in Fig. 26. Each of the use case shown is described in Table 4. There two types of actors in this system that can perform the following tasks:

(i) Capture/take tomato leaf images.
(ii) Uploading tomato leaf images from the gallery.
(iii) View the uploaded/captured image.
(iv) Display general information about tomatoes.
(v) Display general information about *Tuta absoluta* pest.
(vi) Displaying segmentation results.

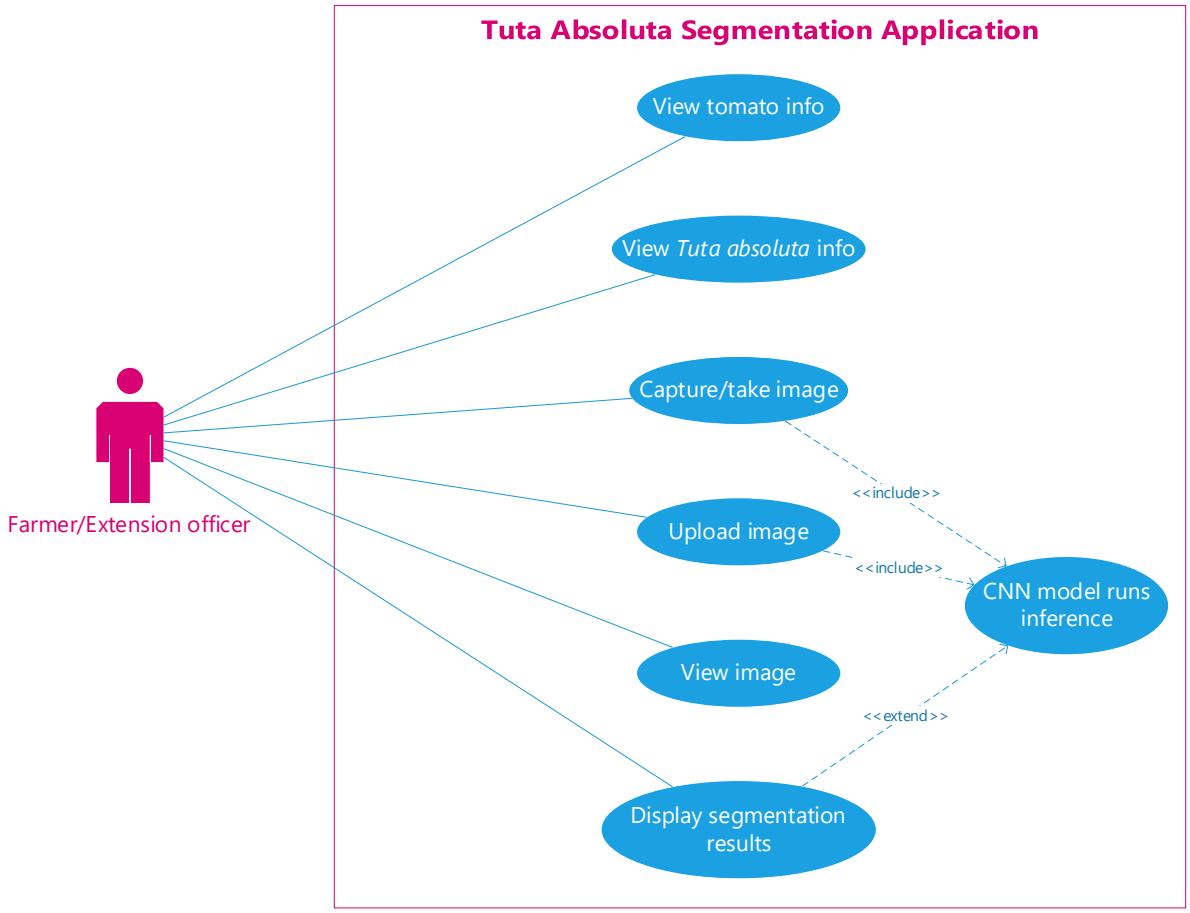


Figure 26: The use case diagram for *Tuta absoluta* segmentation application

Table 4: Description of the Use Cases

Use Case	Description	Actor (s)
View tomato info	The user(s) can display general information about tomatoes such as the scientific name, production statistics, and planting information.	Farmer or Extension officer
View *Tuta absoluta* info	The user(s) can display general information about *Tuta absoluta* such as their common and scientific names, physiology, and life cycle.	Farmer or Extension officer
Capture/take photo	The user(s) can access their mobile phone's camera to take a photo of a tomato plant. Then the system will automatically run inference on the photo using the CNN model in the background to segment *tuta* mines.	Farmer or Extension officer
Upload an image	The user(s) can upload a tomato plant image from their mobile phone's gallery. Then the system will automatically run inference on the uploaded photo using the CNN model in the background to segment *tuta* mines.	Farmer or Extension officer
View image	The user(s) can view the captured or uploaded image in the mobile application.	Farmer or Extension officer
Display segmentation results	The user(s) can display the original image, segmentation results, and overlay in the mobile application.	Farmer or Extension officer

3.10.4 Activity Diagram

The activity diagram depicts the dynamic aspect of the application software. It graphically represents a series of actions or flow of control in a system with support for iteration and concurrency. Figure 27 describes the activity diagram for *Tuta absoluta* segmentation mobile application.

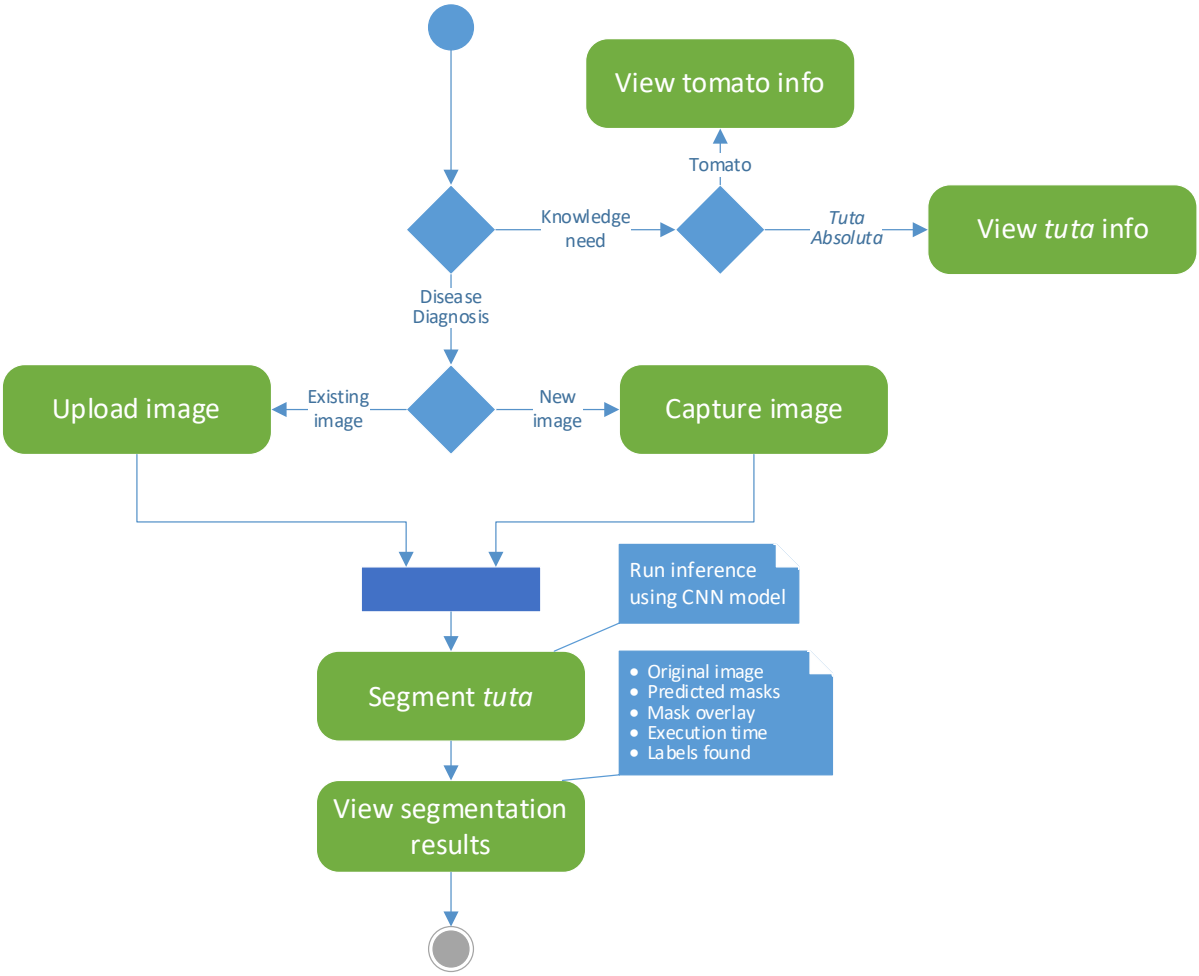


Figure 27: The activity diagram for *Tuta absoluta* segmentation application

3.10.5 Sequence Diagram

The sequence diagram describes how objects interact with each other for a particular scenario of the system. It details the way operations in the application software are performed. Figure 28 shows the sequence diagram for *Tuta absoluta* segmentation mobile application.

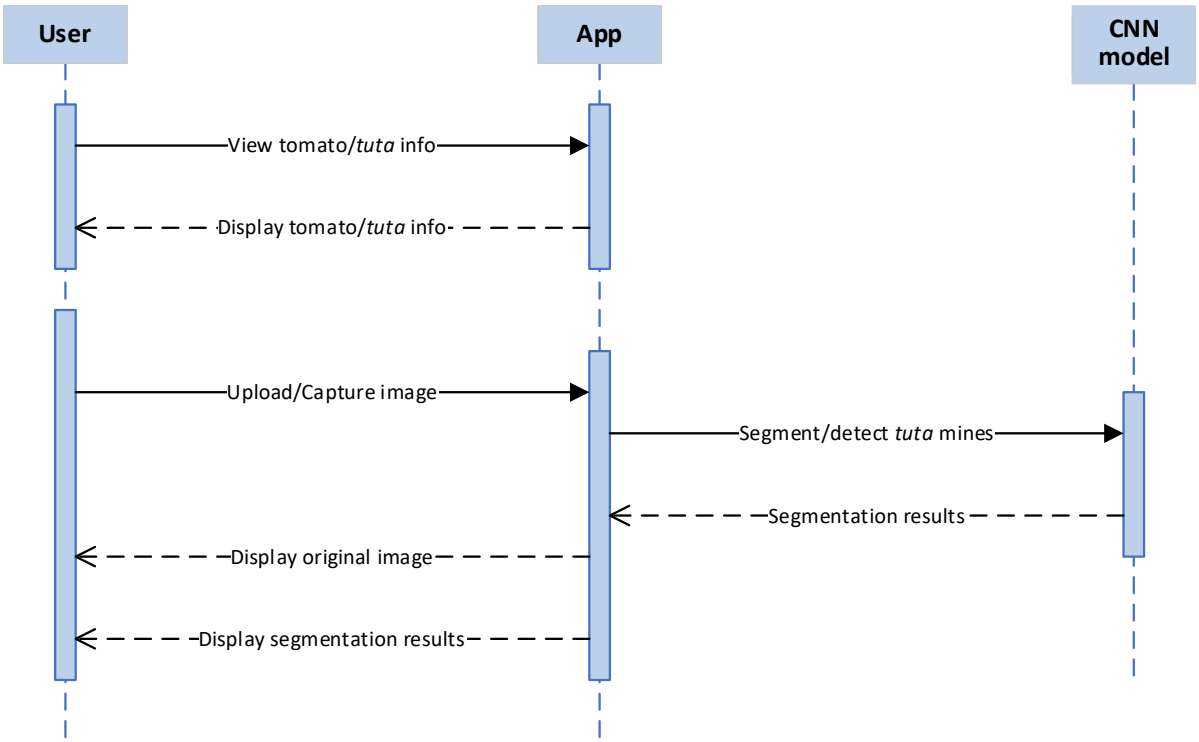


Figure 28: The sequence diagram for *Tuta absoluta* segmentation application

3.10.6 Software Development Methodology

For mobile application development, the agile approach was used. This is a software development methodology that allows developers to build a prototype, show its functionality to users, and make changes based on their input. It encourages continuous development and testing iteration throughout the software development lifecycle. This approach was important in delivering reliable, user-friendly and efficient software in a short time. Figure 29 demonstrates the agile software development life cycle.

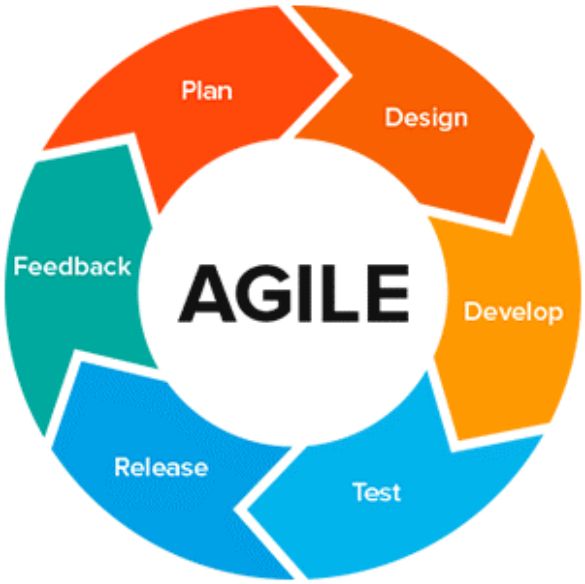


Figure 29: Agile methodology (Abellán, 2020)

3.10.7 Technologies Used

In the development of this mobile application, the python programming language was used to convert the CNN models to a mobile-compatible format. Then Extensible Markup Language (XML) and Kotlin programming language were used to define the interface and functionalities of the application, respectively. The following platforms were used in the implementation of the application software:

(i) Android Studio Integrated Development Environment (IDE).
(ii) Anaconda Platform
(iii) Samsung SM-A025F for testing the mobile application.

3.11 Quantification

On top of the proposed instance segmentation model, Mask RCNN, a custom function was developed for counting the detected *tuta* mines using Open-Source Computer Vision (OpenCV) library in Python programming language. OpenCV is an open-source and cross-platform library built to provide a common infrastructure for computer vision applications focusing on image and video processing and analysis (Bradski & Kaehler, 2009). In this implementation, the OpenCV library is used to find the detected *tuta* mines using contours with the find_contours() method, draw bounding boxes around and count them. Then putText(), an OpenCV method is used to add text at the top left corner of the image, showing the number of detected *tuta* mines in one tomato leaf image at a time. Appendix 5 describes the code for the implementation of OpenCV object counting based on the proposed Mask RCNN model.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 The Dataset

For this work, 1212 and 1240 image sets with only infested plants from the total collection of images that make up the dataset were selected. The criterion for selecting the images was that each image must contain at least one *tuta* mask indicating the presence of *Tuta absoluta*'s mine in the image. The distribution of the annotated dataset was split into training and test sets in a ratio of 80:20 respectively as shown in Table 5. The training set was used to train the model while the test set was used to evaluate the model's performance.

Table 5: Train/test set splits

Model	Data Ratio	Training set	Test set	Total
U-Net (VOC format)	80:20	969	243	1212
Mask RCNN (COCO format)	80:20	992	248	1240

4.2 Loss Results

4.2.1 Mask RCNN Loss Results

Mask RCNN with ResNet50 and ResNet101 as backbone architectures was trained separately on the annotated dataset described in Table 5 keeping a record of the training and validation loss for 200 epochs. Figure 30 demonstrates the loss diagrams of the proposed Mask RCNN model during the training process using the loss function described in Equation (3.7). The training and validation losses were estimated after each training epoch. As the training process progresses, the value of training loss rapidly decreases, followed by validation loss. As revealed in the figure, the validation loss starts to display an upward trend after several epochs while the training loss continues to decrease slightly making a gap between the two losses. This suggests that the model is overfitting. The last epoch that did not overfit is selected.

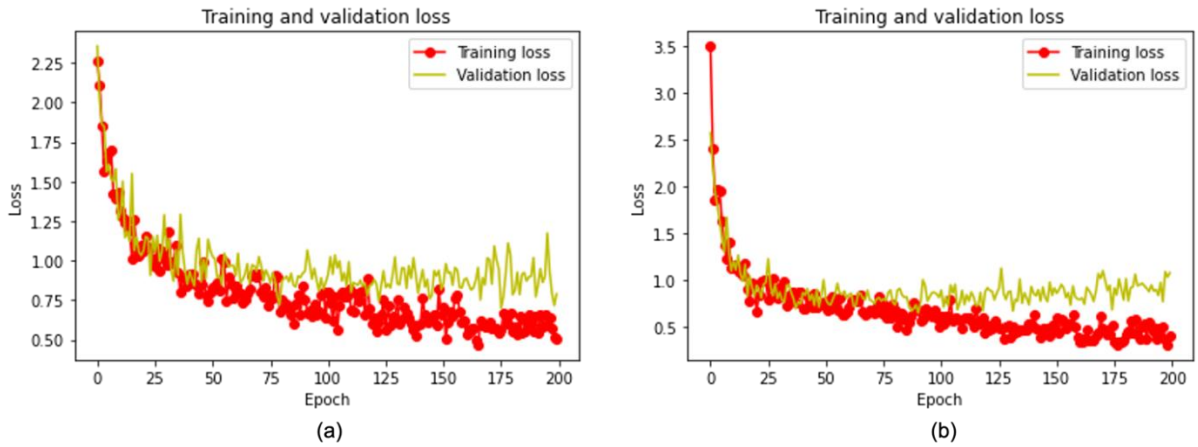


Figure 30: Training and validation loss for Mask RCNN (a) Loss graph for Mask RCNN-ResNet50 (b) Loss graph for Mask RCNN-ResNet101

Then the network was retrained with augmentation techniques described in Section 3.5.4 keeping a record of the total loss. As shown in Fig. 31, the loss function monotonically decreases during the training phase. The losses stabilize at the end of the training, indicating that the proposed model learns and segments the *tuta* mines well without overfitting.

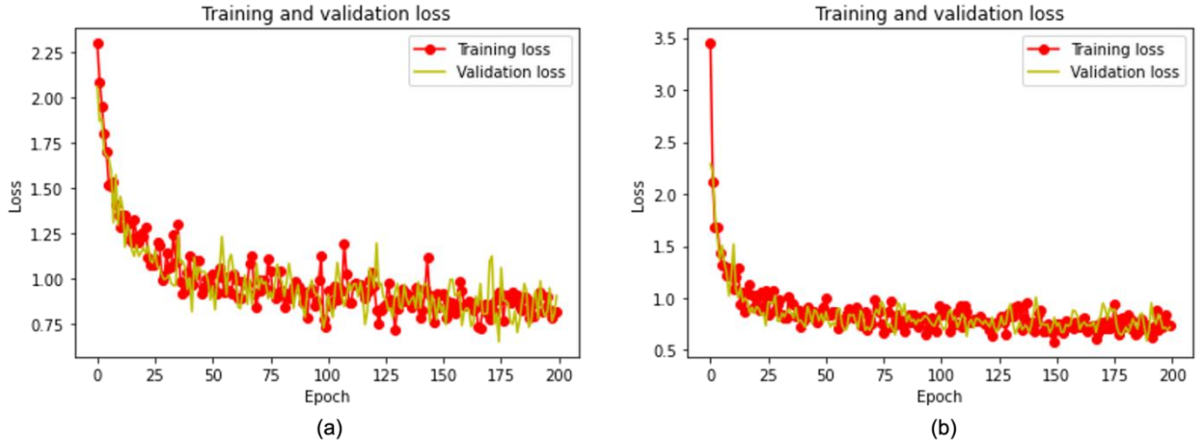


Figure 31: Training and validation loss for Mask RCNN with augmentations. (a) Loss graph for Mask RCNN-Resnet50 with augmentations. (b) Loss graph for Mask RCNN-Resnet101 with augmentations

4.2.2 U-Net Loss Results

On the other hand, U-Net demonstrated better performance with a low loss value compared to that of Mask RCNN. As shown in Fig. 32, the training loss starts at around 0.2761 and validation loss starts at 0.0604 then keeps decreasing steadily as the number of iterations increases. Approaching the 125th epoch, the validation loss slightly starts to increase while training loss continues to decrease, creating a small gap between the two losses. The small gap between training and validation loss implies that the model fits well on the features of the dataset at the early and later stages of the training process and can segment the *tuta* mines well without overfitting. It also indicates that the performance of the proposed model can be improved to a better value upon adding more data.

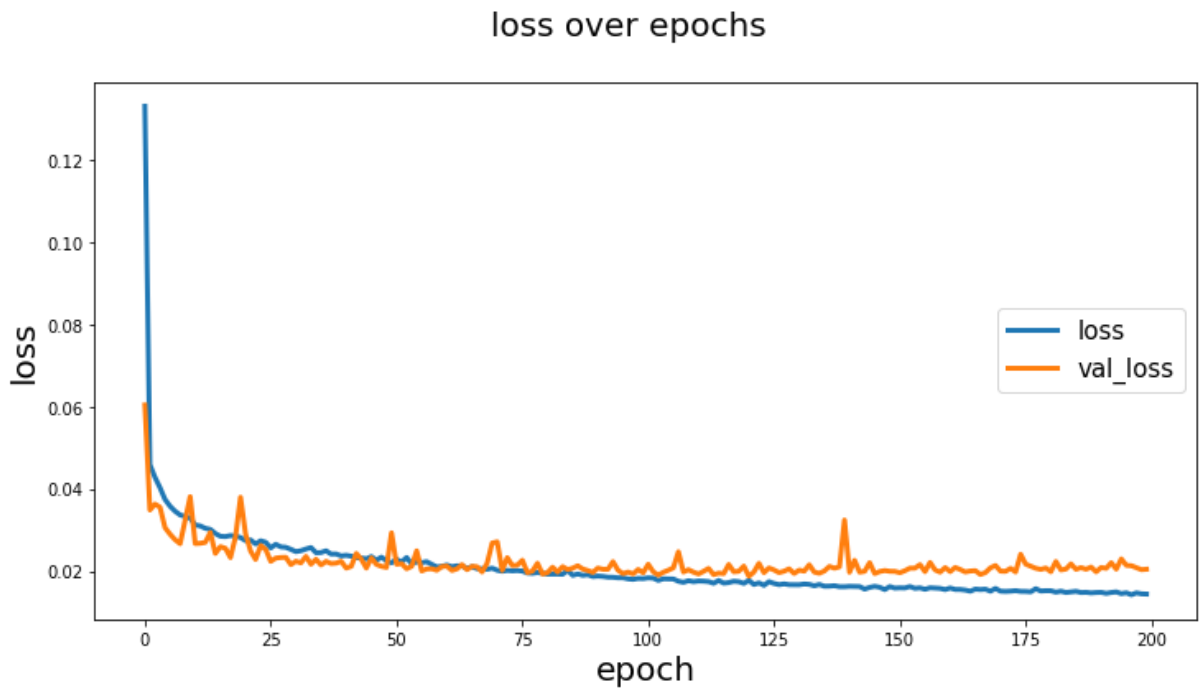


Figure 32: Training and validation loss for U-Net

4.3 Evaluation Metrics Results

4.3.1 Jaccard Index/IoU

The U-Net model was trained on our dataset for 200 epochs keeping a record of the training and validation IoU values thresholded at 0.5. As shown in Fig. 33, the training IoU starts at a minimum point of 0.3356 and the validation IoU starts at 0.5028 then keeps on increasing as the number of epochs increases. When approaching the end of the training process at the 200th epoch, the IoU is 0.7860 and the validation IoU is 0.7490. This implies that the model could learn well the features in the dataset and semantically segment the *tuta* mines well from other parts of the images.

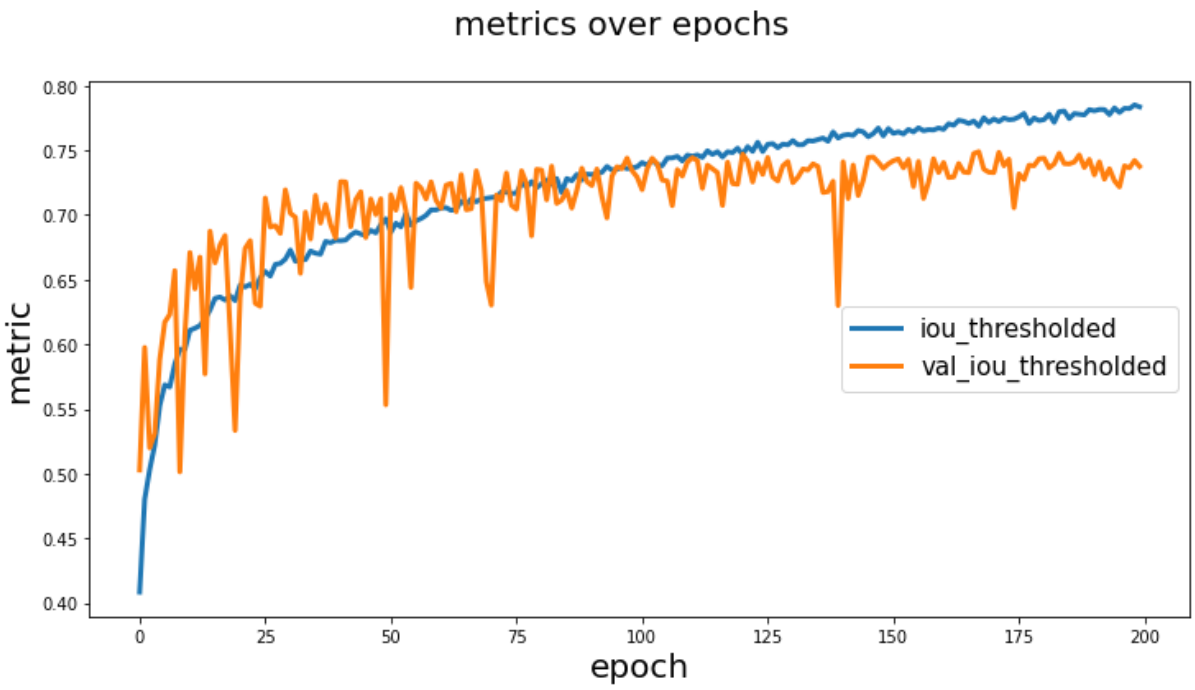


Figure 33: Intersection over Union (IoU) curve for U-Net model

4.3.2 Dice Coefficient

While training the U-Net model on the dataset described in Table 5, values of the dice coefficient at each iteration for 200 epochs were recorded. Figure 34 shows the dice coefficient curve as the training process progresses. As shown, the Dice Coefficient starts at a minimum point of 0.2092 and the validation dice coefficient starts at 0.4900 then increases as the training process progresses. In the end, U-Net achieves a high detection rate with a dice coefficient as

high as 0.8286 and a validation dice coefficient of 0.8116, implying that the model fits well on the data and can precisely segment *tuta* mines on tomato leaves.

The evaluation metrics results for the semantic segmentation model, U-Net are summarized in Table 6. Some examples of segmentation executed by the proposed U-Net model are shown in Fig. 35. As can be seen, the model generates precise segmentations of *tuta* mines in tomato plants.

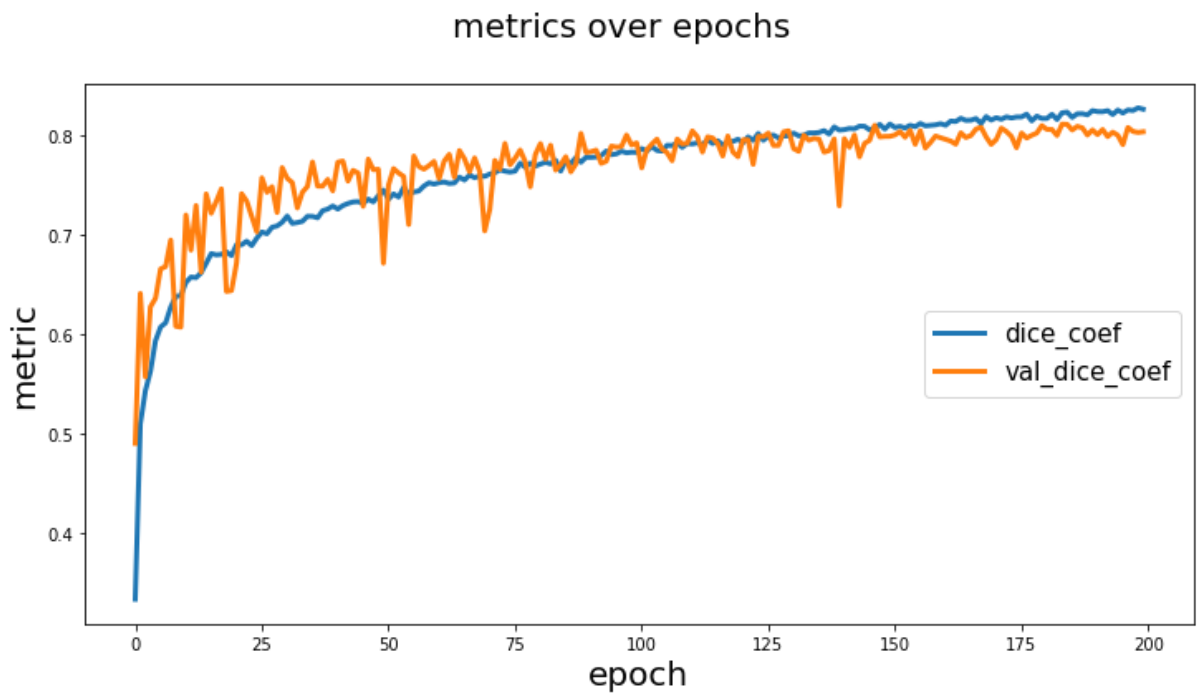


Figure 34: Dice Coefficient curve of the U-Net model

Table 6: The evaluation metrics results for semantic segmentation model

Method	Jaccard Index/IoU (%)	Dice Coefficient (%)	Validation IoU (%)	Validation Dice Coefficient (%)
U-Net	78.60	**82.86**	74.90	81.16

Figure 35: Examples of segmentations achieved by the proposed U-Net model

4.3.3 The mAP

The area under the Precision-Recall (PR) curve, which defines the Average Precision (AP), can summarize the performance of a segmentation model, the x-axis being recall and the y-axis being precision. A threshold of IoU was set at 0.5 below which any segmentation with a score less than this is considered a FP. As shown in Fig. 36, the PR curve was monotonically decreasing, which is suitable for better performance. The precision of a detector with good performance remains high as recall increases, implying that it can detect a large proportion of TP before it starts detecting FP.

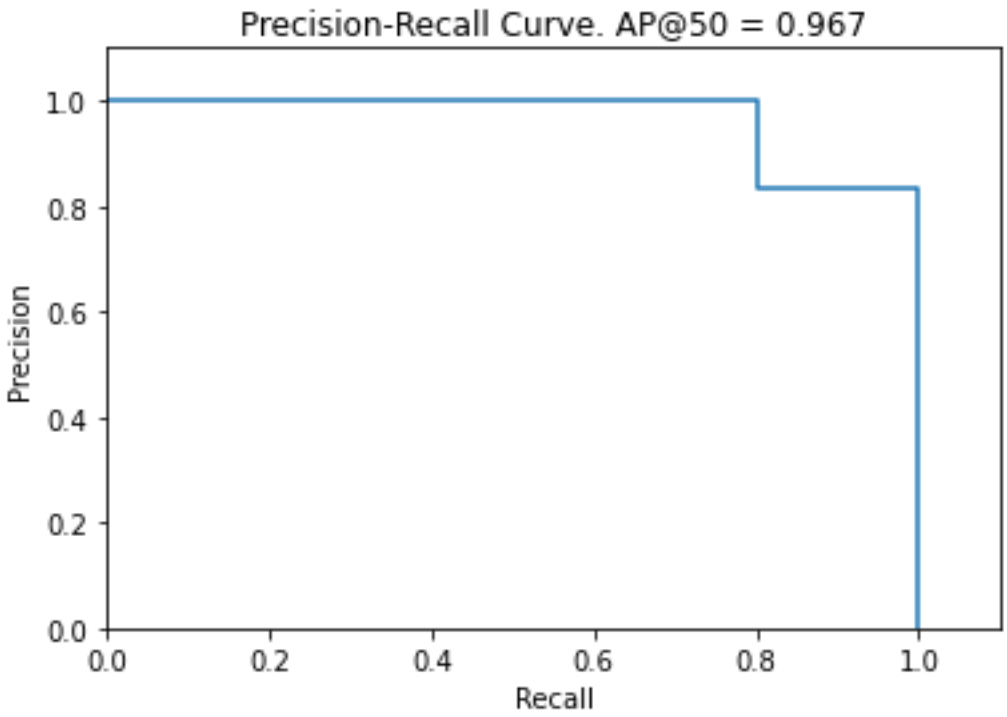


Figure 36: The P-R Curve

Table 7 presents the mAP values used to evaluate the performance of the proposed methods in detecting and segmenting *tuta* mines on tomato images with minimum detection confidence of 0.7. The mAP value of Mask RCNN-ResNet50 with augmentations is as high as 85.67%, achieving the highest detection rates on *tuta* mines in tomato plants compared to other methods. As seen in the table, the performance of Mask RCNN-ResNet50 and Mask RCNN-ResNet101 is relatively low, with mAPs of 81.01% and 81.09%, respectively. This is likely because of the complexities of backbone architectures to train on an inadequate amount of data. Examples of segmentations produced by the proposed Mask RCNN model are shown in Fig. 37. As can be seen, the model could detect even the smallest *tuta* mines on tomato leaves.

Table 7: The mAP (primary metric) values of the tomato images obtained by different detection architectures

Method(s)	mAP (%)
Mask RCNN-ResNet50	81.01
Mask RCNN-ResNet50 with augmentations	**85.67**
Mask RCNN-ResNet101	81.09
Mask RCNN-ResNet101 with augmentations	**83.60**

Figure 37: Examples of segmentations carried out by the proposed Mask RCNN model

4.4 Training Time

The efficiency of the model is another important performance criterion apart from the detection rates. Table 8 reveals the training time in minutes for each method employed in this study for all tomato leaf images. It can be seen that the training time of U-Net is 483.50 minutes which is 169.91, 187.07, 359.45, and 369.9 minutes shorter than that of Mask RCNN-ResNet50, Mask RCNN-ResNet50 with augmentations, Mask RCNN-ResNet101, and Mask RCNN-ResNet101 with augmentations, respectively. This is because the ResNet101 has a more complex structure compared to ResNet50 and U-Net hence longer training time. Additionally, Mask RCNN with

ResNet101 as a backbone, Mask RCNN with ResNet50 as a backbone, and U-Net models had 63 621 918, 44 603 678 and 7 763 041 trainable parameters, respectively. Therefore, U-Net was more efficient in this category compared to other training methods.

Table 8: Training time

Method(s)	Training time (minutes)
Mask RCNN-ResNet50	653.41
Mask RCNN-ResNet50 with augmentations	670.57
Mask RCNN-ResNet101	842.95
Mask RCNN-ResNet101 with augmentations	853.40
U-Net	483.50

4.5 Developed Mobile Application

The U-Net model was selected for mobile application deployment since it has a less complex structure, requiring less computational costs than other training methods in this study. The U-Net model was converted to TFLite format then embedded to a mobile application in Android Studio as described in Section 3.10.

A simple and user-friendly mobile application was developed to allow smooth interaction between the farmers/extension officers and the application. After installing the application in their Android smartphones, the farmer/extension officer simply clicks on the application icon which welcomes them with a colorful splash screen with a progress bar displayed only for two seconds before landing to the scrollable home page. As shown in Fig. 38, the home screen includes two clickable cards with quick facts about the tomato plant and *Tuta absoluta* as well as a clickable floating button for *tuta* segmentation. The user can click on the tomato plant card to view general information about tomatoes such as their scientific name, production statistics, and cropping information such as water, soil, and fertilizer they need to grow as shown in Fig. 39 (a). Also, the user can view general information about the tomato leaf miner such as their common and scientific names, physiology, and life cycle by clicking on the *Tuta absoluta* card from the home page as shown in Fig. 39 (b). This will help the farmer learn and understand the pest well to easily take measures to control it.

Additionally, a clickable floating button for *tuta* segmentation was included on every page in the mobile application so that the farmer can easily navigate to the disease diagnosis page to detect and segment *tuta* mines in tomato leaf images. The disease diagnosis page consists of a

frame layout that accesses and displays the device camera, a floating camera button for capturing tomato leaf images, a floating upload button for uploading images stored in the phone's gallery, and a horizontally scrollable section that displays the original tomato leaf image captured or uploaded to the application, predicted masks and the masks overlayed with the original image. It also consists of a bottom sheet layout that displays the input image size, information about the labels found during detection with their colors or no labels found, model execution time in milliseconds and the button to rerun the model. In summary, the farmer or extension officer can use the mobile application to segment *tuta* mines in tomato leaf images as follows:

(i) The user clicks the "Segment Tuta" button from any other page that will take him/her to the "Disease Diagnosis" page.

(ii) On the "Disease Diagnosis" page, the user can click the camera or upload buttons to capture or upload a tomato leaf image respectively.

(iii) The mobile application will process the image using a CNN segmentation model running in the background and then give feedback which is displayed in the application.

(iv) Suppose the image does not have *tuta* mines. In that case, the application will display a text "No labels found", otherwise the application will display three images namely, original image, predicted masks and overlay as well as the labels found with their colors and the model execution time as shown in Fig. 40.

The *Tuta absoluta* segmentation mobile application was designed to be available and operate offline once installed. This will help the poor farmers avoid the costs of buying internet bundles to access the mobile application and detect *tuta* mines in their farms.

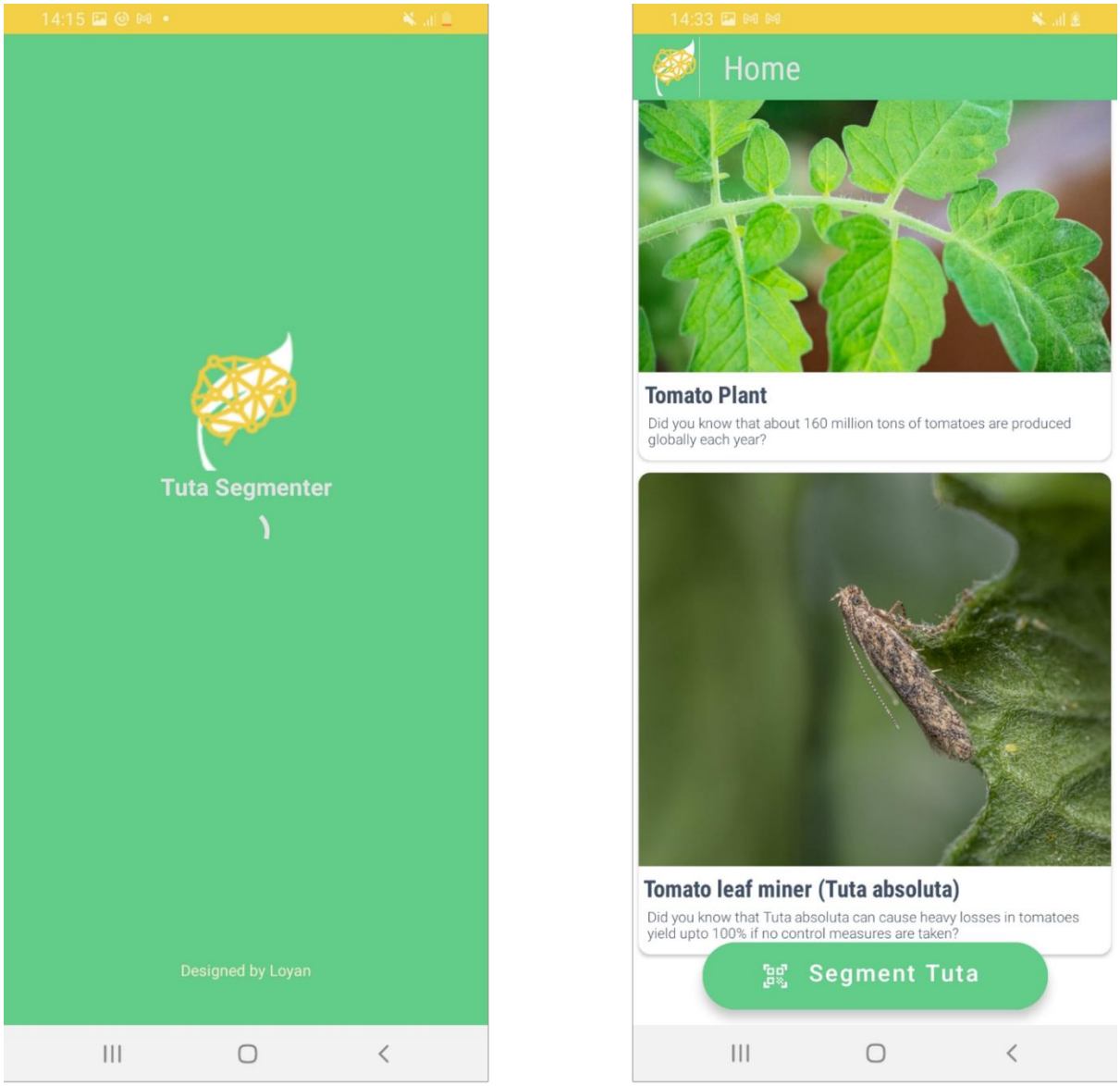


Figure 38: ***Tuta absoluta*** **segmentation mobile application (a) Splash screen (b) Landing/home page**

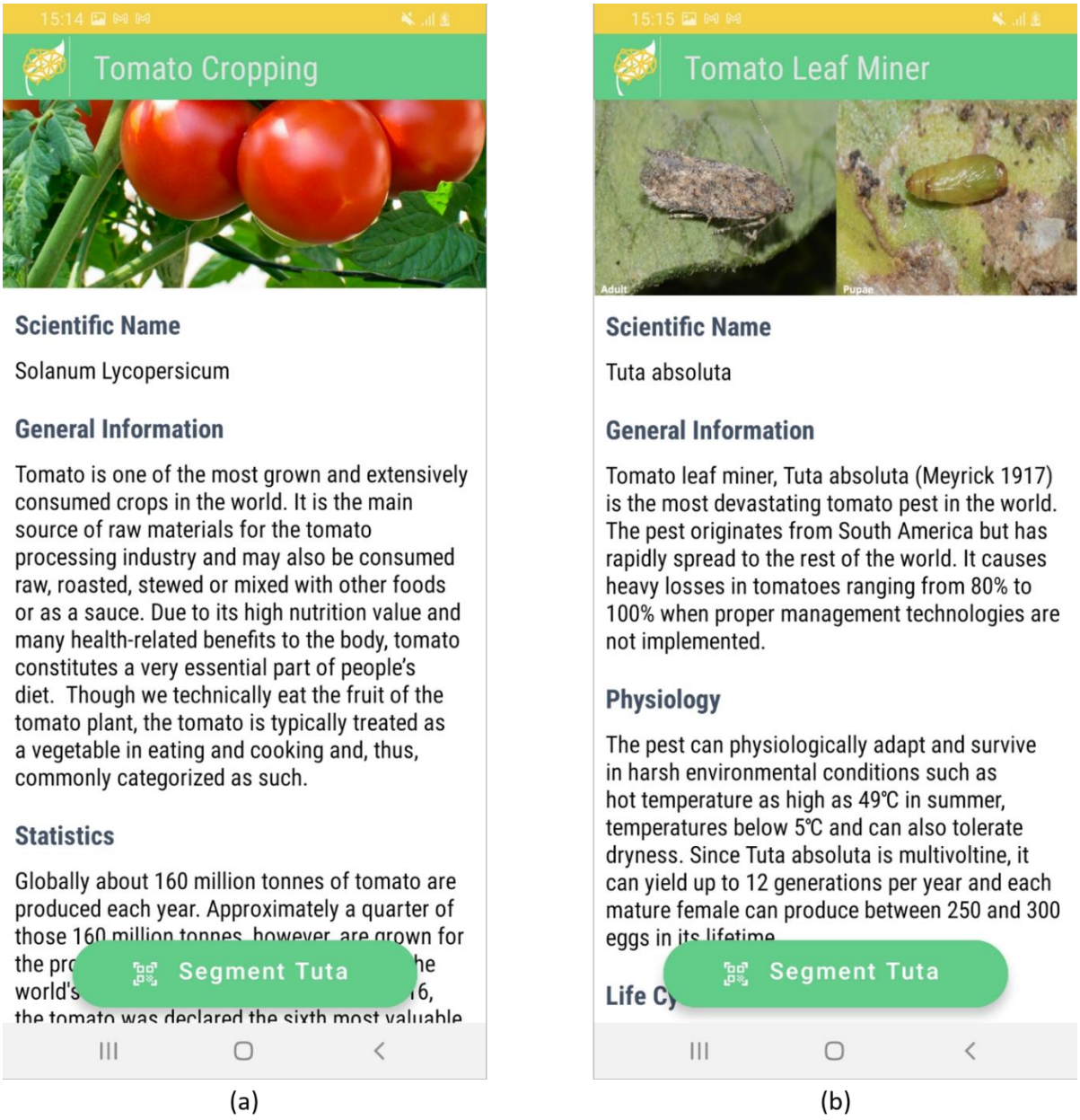
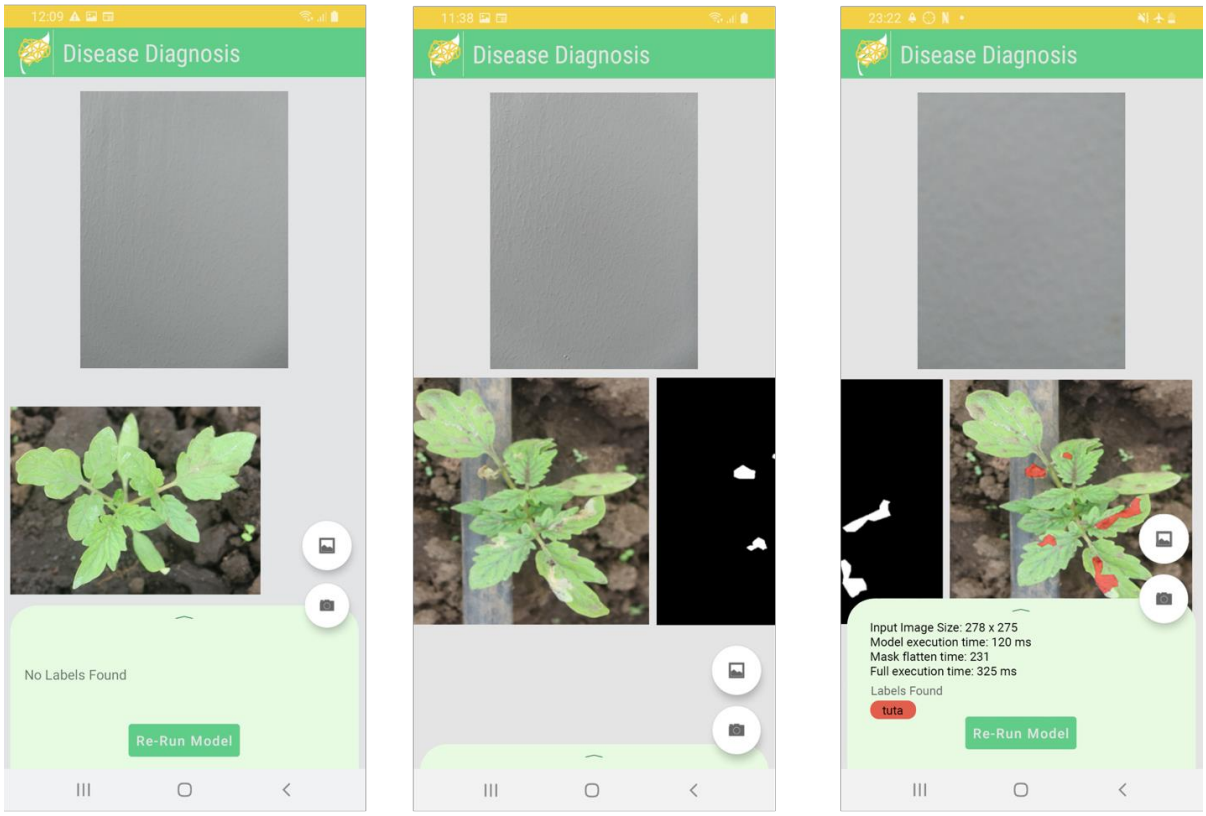


Figure 39: *Tuta absoluta* segmentation mobile application (a) The description page for tomato cropping (b) Description page for *Tuta absoluta*



(a) (b) (c)

Figure 40: *Tuta absoluta* segmentation mobile application (a) A healthy plant with no *tuta* mines (b) Original image and mask prediction (c) Segmentation results

4.6 Quantification Results

Figure 41 shows the quantification results obtained using the OpenCV library built on top of the proposed Mask RCNN model. As can be observed, the model was able to accurately find the detected regions with *tuta* mines, count them and display the number of *tuta* mines present in a tomato leaf image. This can help farmers understand the extent of the damage caused by *Tuta absoluta* to tomato plants and take appropriate measures to control the pest before it causes further damage to tomatoes in the farm.

Figure 41: Examples of quantification results carried out by the OpenCV function built on top of the proposed Mask RCNN model

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

While several studies have been conducted in the agricultural sector to diagnose various diseases and pests in plants, relatively few studies in the literature have focused on predicting disease severity. And to the best of the authors' knowledge, none discuss the segmentation and quantification of *Tuta absoluta*'s damage on tomato plants. However, to control these diseases and pests on plants without causing uneconomical damage to the environment, it needs accurate segmentations to determine the extent of damage caused. The development of a sophisticated technical solution for the early detection of *Tuta absoluta*-caused tomato plant damage is in high demand due to the need to rescue farmers' tomato productivity losses.

This study aimed to tackle the problem of precisely segmenting *Tuta absoluta*'s damage on tomato plants and determining the extent of damage at their early growth stage. To address this problem, this novel work proposed deep CNN models based on U-Net and Mask RCNN architectures which are used for automatic semantic and instance segmentation, respectively. For accomplishing this work, an annotated dataset with 2452 images collected from the field was used to train the models separately. The experimental results show that the Mask RCNN-ResNet101 model performs best achieving a mAP of 85.67%, while the U-Net model obtained 78.60% and 82.86% of Jaccard index and Dice Coefficient, respectively. Also, Mask RCNN-ResNet50 has a shorter training time than Mask RCNN-ResNet101 due to its less complex structure. Both suggested models were very accurate in segmenting the shapes of *Tuta absoluta*-infected areas in tomato leaves and determine their extent of the damage. The instance segmentation model, Mask RCNN, was then used to automatically determine the number of *tuta* mines present in tomato leaf images. This demonstrates that deep learning is the new promising technology for fully automatic and early determination of *Tuta absoluta* severity status.

5.2 Recommendations

This work has laid the foundation that could be used to provide both theoretical and practical analysis for future works in segmentation-based quantification of *Tuta absoluta*'s damage to tomato plants. It demonstrated how the integration of modern technology in agriculture,

particularly diseases and pests diagnosis, could help control and possibly overcome diseases and pests in plants and improve their productivity. This study recommends that the robustness of the proposed model be further stabilized by expanding the diversity of tomato images adding other pests and diseases that affect the plant. Even though the validation results of the proposed models indicate good segmentation accuracy, more annotated data is needed to further validate and improve the performance.

In the future, a CNN decision support system will be developed and deployed in a mobile or computer to enable farmers and extension officers to make intelligently informed decisions on how to control the pest so that to increase productivity. The system will be able to suggest actions to be taken such as the application of Integrated Pest Management (IPM) techniques to control the pest based on their severity.

REFERENCES

Abadi, M. (2016). TensorFlow: Learning Functions at Scale. *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, *51*(9), 1–1. https://doi.org/10.1145/3022670.2976746

Abdulla, W. (2017). *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. Matterport. https://github.com/matterport/Mask_RCNN. Accessed on July 30,2020.

Abellán, E. (2020, February 6). *What's the Agile Methodology and How Can It Benefit Your Enterprise?* WAM. https://www.wearemarketing.com/blog/what-is-the-agile-methodology-and-what-benefits-does-it-have-for-your-company.html. Accessed on April 5, 2021.

Amara, J., Bouaziz, B., & Algergawy, A. (2017). A Deep Learning-based Approach for Banana Leaf Diseases Classification. In B. Mitschang, D. Nicklas, F. Leymann, H. Schöning, M. Herschel, J. Teubner, T. Härder, O. Kopp, & M. Wieland (Eds.), *Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband* (pp. 78–88). Gesellschaft für Informatik e.V. https://dl.gi.de/handle/20.500.12116/944

Arah, I. K. (2015). An overview of post-harvest challenges facing tomato production in Africa. *African Studies Association of Australasia and the Pacific (AFSAAP) 37th Annual Conference*, *11*, 1-21. Dunedin, New Zealand.

Bradski, G., & Kaehler, A. (2009). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc. https://doi.org/10.1109/mra.2009.933612

Brahimi, M., Boukhalfa, K., & Moussaoui, A. (2017). Deep Learning for Tomato Diseases: Classification and Symptoms Visualization. *Applied Artificial Intelligence*, *31*(4), 299–315. https://doi.org/10.1080/08839514.2017.1315516

Burton-Freeman, B., & Reimers, K. (2011). Tomato Consumption and Health: Emerging Benefits. *American Journal of Lifestyle Medicine*, *5*(2), 182–191. https://doi.org/10.1177/1559827610 387488

Çetin, B., & Vardar, A. (2008). An economic analysis of energy requirements and input costs for tomato production in Turkey. *Renewable Energy*, *33*(3), 428–433. https://doi.org/10.1016/j.renene.2007.03.008

Chidege, M., Al-zaidi, S., Hassan, N., Abisgold, J., Kaaya, E., & Mrogoro, S. (2016). First record of tomato leafminer Tuta absoluta Meyrick (Lepidoptera: Gelechiidae) in Tanzania. *Agriculture and Food Security*, *5*(1), 1–7. https://doi.org/10.1186/s40066-016-0066-4

Chollet, F. (2017). Introduction to Keras. In *Deep Learning with Python* (1st ed., pp. 60–62). Manning Publications Co.

Ciresan, D. C., Giusti, A., Gambardella, L. M., & Schmidhuber, J. (2012). Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. *Proceedings of the 25th International Conference on Neural Information Processing Systems, 25*, 2843–2851.

Clarke, J. F. (1965). Microlepidoptera of Juan Fernandez Island. *Proceedings of the United States National Museum*, *34*(2), 36–107. https://doi.org/10.4039/Ent3436-2

Cuthbertson, A. G. S., Mathers, J. J., Blackburn, L. F., Korycinska, A., Luo, W., Jacobson, R. J., & Northing, P. (2013). Population development of Tuta absoluta (Meyrick) (Lepidoptera: Gelechiidae) under simulated UK glasshouse conditions. *Insects*, *4*(2), 185–197. https:// doi.org/10.3390/insects4020185

Desneux, N., Luna, M. G., Guillemaud, T., & Urbaneja, A. (2011). The invasive South American tomato pinworm, Tuta absoluta, continues to spread in Afro-Eurasia and beyond: the new threat to tomato world production. *Journal of Pest Science*, *84*(4), 403–408. https://doi.org/ 10.1007/s10340-011-0398-6

Desneux, N., Wajnberg, E., Wyckhuys, K. A. G., Burgio, G., Arpaia, S., Narváez-Vasquez, C. A., González-Cabrera, J., Ruescas, D. C., Tabone, E., Frandon, J., Pizzol, J., Poncet, C., Cabello, T., & Urbaneja, A. (2010). Biological invasion of European tomato crops by Tuta absoluta: Ecology, geographic expansion and prospects for biological control. *Journal of Pest Science*, *83*(3), 197–215. https://doi.org/10.1007/s10340-010-0321-6

Díez, M. J., & Nuez, F. (2006). Tomato. *Genetic Resources, Chromosome Engineering, and Crop Improvement: Vegetable Crops*, *3*, 59–113. https://doi.org/10.2307/j.ctt1ffjk9m.28

Doğanlar, M., & Yİğİt, A. (2011). Parasitoids Complex of the Tomato Leaf Miner, Tuta absoluta (Meyrick 1917), (Lepidoptera: Gelechiidae) in Hatay Turkey. *KSU Journal of Natural Sciences*, *14*(4), 28-37. https://doi.org/10.18016/ksujns.36297

Dutta, A., & Zisserman, A. (2019). The VIA annotation software for images, audio and video. *MM 2019 - Proceedings of the 27th Association for Computing Machinery (ACM) International Conference on Multimedia*, 2276–2279. https://doi.org/10.1145/3343031.3350535

Esgario, J. G. M., Krohling, R. A., & Ventura, J. A. (2020). Deep learning for classification and severity estimation of coffee leaf biotic stress. *Computers and Electronics in Agriculture*, *169*, 105162. https://doi.org/10.1016/j.compag.2019.105162

Everingham, M., Gool, L. Van, Williams, C. K. I., & Winn, J. (2010). The PASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, *88*, 303–338. https://doi .org/10.1007/s11263-009-0275-4

FAO. (2017). Transboundary Threats To Food and Nutrition Security in Southern Africa. In *Food and Agriculture Organization of the United Nations (FAO)* (1). http://www.fao.org/3/a-i7691e.pdf. Accessed on January 27, 2020.

FAO. (2019). *FAOSTAT - Crops Production*. Food and Agriculture Organization (FAO) Statistics. http://www.fao.org/faostat/en/?#compare. Accessed on January 18, 2020.

FAOSTAT. (2019). *Tomato Production Worldwide*. Food and Agriculture Organization (FAO). http://www.fao.org/faostat/en/?#data/QC. Accessed on March 27, 2020.

Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, *145*(1), 311–318. https://doi.org/10.1016/j.compa g.2018.01.009

Fuentes, A., Yoon, S., Kim, S. C., & Park, D. S. (2017). A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition. *Sensors*, *17*(9), 2022-2043. https://doi.org/ 10.3390/s17092022

Guedes, R. N. C., & Picanço, M. C. (2012). The tomato borer Tuta absoluta in South America: Pest status, management and insecticide resistance. *European and Mediterranean Plant Protection Organization Bulletin*, *42*(2), 211–216. https://doi.org/10.1111/epp.2557

Guimapi, R. Y. A., Mohamed, S. A., Okeyo, G. O., Ndjomatchoua, F. T., Ekesi, S., & Tonnang, H. E. Z. (2016). Modeling the risk of invasion and spread of Tuta absoluta in Africa. *Ecological Complexity*, *28*, 77–93. https://doi.org/10.1016/j.ecocom.2016.08.001

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2018). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(2), 386–397. https://doi.org/10.1109/TPAMI.2018.2844175

Hughes, D. P., & Salathe, M. (2015). *An open access repository of images on plant health to enable the development of mobile disease diagnostics*. http://arxiv.org/abs/1511.08060

Kingma, D. P., & Ba, J. (2014, December 22). *Adam: A Method for Stochastic Optimization*. http://arxiv.org/abs/1412.6980

Lawrence, S., & Giles, C. L. (2000). Overfitting and Neural Networks: Conjugate Gradient and Backpropagation. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. Neural Computing: New Challenges and Perspectives for the New Millennium*, *1*, 114–119. https://doi.org/10.1109/IJCNN.2000.857823

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Liang, Q., Xiang, S., Hu, Y., Coppola, G., Zhang, D., & Sun, W. (2019). PD 2 SE-Net: Computer-assisted plant disease diagnosis and severity estimation network. *Computers and Electronics in Agriculture*, *157*, 518–529. https://doi.org/10.1016/j.compag.2019.01.034

Lin, K., Gong, L., Huang, Y., Liu, C., & Pan, J. (2019). Deep learning-based segmentation and quantification of cucumber powdery mildew using convolutional neural network. *Frontiers in Plant Science*, *10*(2), 1–10. https://doi.org/10.3389/fpls.2019.00155

Lin, T., Zitnick, C. L., & Doll, P. (2014). Microsoft COCO: Common Objects in Context. *European Conference on Computer Vision*, *8693*, 1–15. https://doi.org/10.1007/978-3-319-10602-1_48

Liu, X., Hu, C., & Li, P. (2020). Automatic segmentation of overlapped poplar seedling leaves combining Mask R-CNN and DBSCAN. *Computers and Electronics in Agriculture*, *178*(5), 105753. https://doi.org/10.1016/j.compag.2020.105753

Maginga, T. J., Nordey, T., & Ally, M. (2018). Extension System for Improving the Management of Vegetable Cropping Systems. *Journal of Information Systems Engineering and Management*, *3*(4), 29-39. https://doi.org/10.20897/jisem/3940

Materu, C. L., Shao, E. A., Losujaki, E., & Chidege, M. (2016). Farmer's Perception Knowledge and Practices on Management of Tuta Absoluta Meyerick (Lepidotera Gelechiidae) in Tomato Growing Areas in Tanzania. *International Journal of Research in Agriculture and Forestry*, *3*(2), 1–5.

Meyrick, E. (1917). Descriptions of South American micro-lepidoptera. *Transactions of the Entomological Society of London*, *1917*, 1–52. https://www.cabdirect.org/cabdirect/abstract/20057001788

Ministry of Agriculture. (2017). *Annual Agriculture Sample Survey Crop and Livestock Report*. https://www.nbs.go.tz/nbs/takwimu/Agriculture/2016-17_AASS Report _Final.pdf

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. https://doi.org/10.1016/j.sciaf.2020.e00590

Mutayoba, V., & Ngaruko, D. (2018). Assessing Tomato Farming and Marketing Among Smallholders in High Potential Agricultural Areas of Tanzania. *International Journal of Economics, Commerce and Management*, *VI*(8), 577–590.

O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. *Computing Research Repository*, *abs/1511.0*, 1–11. http://arxiv.org/abs/1511.08458

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Patil, V. C., Al-Gaadi, K. A., Biradar, D. P., & Rangaswamy, M. (2012). Internet of Things (Iot) and Cloud Computing for Agriculture: An Overview. *Agro-Informatics and Precision Agriculture 2012*, *India*, 292–296. http://insait.in/AIPA2012/ articles/054.pdf

Pawang, S. (2020, March 1). *Mask R-CNN | ML - GeeksforGeeks*. GeeksforGeeks. https://www. geeksforgeeks.org/mask-r-cnn-ml/. Accessed on January 18, 2021.

Pérez-Borrero, I., Marín-santos, D., Gegúndez-Arias, M. E., & Cortés-Ancos, E. (2020). A fast and accurate deep learning method for strawberry instance segmentation. *Computers and Electronics in Agriculture*, *178*, 105-736. https://doi.org/10.1016/j.compag.2020.105736

Povolný, D. (1967). Genitalia of some nearctic and neotropic members of the tribe Gnorimoschemini (Lepidoptera, Gelechiidae). *Acta Entomologica Musei Nationalis Pragae*, *37*, 51–127.

Povolný, D. (1985). Gnorimoschemini of Southern America III: the scrobipalpuloid genera (Insecta, Lepidoptera: Gelechiidae). *Steenstrupia (Copenhagen)*, *11*(1), 1-91. https://pascal-francis.inist.fr/vibad/index.php?action=getRecordDetail&idt=9082178

Povolný, D. (1994). Gnorimoschemini of southern South America VI: identification keys, checklist of Neotropical taxa and general considerations (Insecta, Lepidoptera, Gelechiidae). *Steenstrupia*, *20*(1), 1–42.

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

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*, *9351*, 234–241. https://doi.org/10.1007/978-3-319-24574-4_28

Rupanagudi, S. R., Ranjani, B. S., Nagaraj, P., Bhat, V. G., & Thippeswamy, G. (2015). A novel cloud computing based smart farming system for early detection of borer insects in tomatoes. *Proceedings - 2015 International Conference on Communication, Information and Computing Technology, 2015*, 1-6. https://doi.org/10.1109/ICCICT.2015. 7045722

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. http://people.csail.mit.edu/brussell/research/AIM-2005-025-new.pdf

Rwomushana, I., Beale, T., Chipabika, G., Day, R., Gonzalez-Moreno, P., Lamontagne-Godwin, J., Makale, F., Pratt, C., & Justice, T. (2019). *Tomato leafminer (Tuta absoluta): Impacts and coping strategies for Africa*. CABI (No. 12). https://doi.org/10.1079/ CABICOMM-62-8100

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, *6*(1), 1–48. https://doi.org/10.1186/s40537-019-0197-0

Singh, A. K., Ganapathysubramanian, B., Sarkar, S., & Singh, A. (2018). Deep Learning for Plant Stress Phenotyping: Trends and Future Perspectives. *Trends in Plant Science*, *23*(10), 883–898. https://doi.org/10.1016/j.tplants.2018.07.004

Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., & Stefanovic, D. (2016). Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification. *Computational Intelligence and Neuroscience*, *2016*, 1–11. https://doi.org/10.1155/2016/ 3289801

Soares, M. A., & Campos, M. R. (2020). *Invasive Species Compendium: Datasheet report for Tuta absoluta (tomato leafminer)*. European and Mediterranean Plant Protection Organization (EPPO), Food and Agriculture Organization (FAO). https://www.cabi.org/ isc/datasheet/49260#tosummaryOfInvasiveness

Tang, H., Wang, B., & Chen, X. (2020). Deep learning techniques for automatic butterfly segmentation in ecological images. *Computers and Electronics in Agriculture*, *178*(5), 105739. https://doi.org/10.1016/j.compag.2020.105739

Tomato News. (2020). *The global tomato processing industry*. The Tomato Online Conference. http://www.tomatonews.com/en/background_47.html. Accessed on January 16, 2021.

Tonnang, H. E. Z., Mohamed, S. F., Khamis, F., & Ekesi, S. (2015). Identification and risk assessment for worldwide invasion and spread of tuta absoluta with a focus on Sub-Saharan Africa: Implications for phytosanitary measures and management. *PLoS ONE*, *10*(8), 1–19. https://doi.org/10.1371/journal.pone.0135283

Travis, E. O. (2007). Python for Scientific Computing. *Computing in Science and Engineering*, *9*(3), 10–20. https://doi.org/10.1109/MCSE.2007.58

Tzounis, A., Katsoulas, N., & Bartzanas, T. (2017). Internet of Things in agriculture, recent advances and future challenges. *Biosystems Engineering*, *164*, 31–48. https://doi.org/ 10.1016/j.biosystemseng.2017.09.007

United Nations. (2017). *World Population Prospects: The 2017 Revision, Key Findings and Advance Tables*. https://esa.un.org/unpd/wpp/publications/files/wpp2017_keyfindings. pdf

United Nations. (2018). *United Nations, The Sustainable Development Goals Report 2018*. 40. https://doi.org/10.29171/azu_acku_pamphlet_k3240_s878_2016

Van Damme, V., Berkvens, N., Moerkens, R., Berckmoes, E., Wittemans, L., De Vis, R., Casteels, H., Tirry, L., & De Clercq, P. (2015). Overwintering potential of the invasive leafminer Tuta absoluta (Meyrick) (*Lepidoptera: Gelechiidae*) as a pest in greenhouse tomato production in Western Europe. *Journal of Pest Science*, *88*(3), 533–541. https://doi.org/10.1007/s10340-014-0636-9

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*, *2018*, 1-14. https://doi.org/10.1155/2018/7068349

Wang, G., Sun, Y., & Wang, J. (2017). Automatic Image-Based Plant Disease Severity Estimation Using Deep Learning. *Computational Intelligence and Neuroscience*, *2017*, 1-8. https://doi. org/ 10.1155/2017/2917536

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*, 1-15. https:// doi.org/ 10.1155/2019/9142753

Weiss, K., Khoshgoftaar, T. M., & Wang, D. D. (2016). A survey of transfer learning. *Journal of Big Data*, *3*(1), 1-40. https://doi.org/10.1186/s40537-016-0043-6

Zekeya, N., Chacha, M., Ndakidemi, P., Materu, C., Chidege, M., & Mbega, E. (2016). Tomato Leafminer (Tuta absoluta Meyrick 1917): A Threat to Tomato Production in Africa. *Journal of Agriculture and Ecology Research International*, *10*(1), 1–10. https://doi.org/ 10.9734/ jaeri/2016/28886

Zekeya, N., Ndakidemi, P. A., Chacha, M., & Mbega, E. (2017). Tomato Leafminer, Tuta absoluta (Meyrick 1917), an emerging agricultural pest in Sub-Saharan Africa: Current and prospective management strategies. *African Journal of Agricultural Research*, *12*(6), 389–396. https://doi.org/10.5897/AJAR2016.11515

Zhang, K., Wu, Q., Liu, A., & Meng, X. (2018). Can deep learning identify tomato leaf disease? *Advances in Multimedia*, *2018*, 1-10. https://doi.org/10.1155/2018/6710865

APPENDICES

Appendix 1: Mask RCNN Model Source Code

Import important libraries

```
import os
import sys
import random
import math
import re
import time
import datetime
import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt
import json
import skimage.draw
# for visualization
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon

%matplotlib inline
```

Setup configurations

```
class TutaConfig(Config):
    # Give the configuration a recognizable name
    NAME = "tuta"

    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

    # Number of classes (including background)
    NUM_CLASSES = 1 + 1  # Background + tuta

    #resize the images to a
    IMAGE_MIN_DIM = 512
    IMAGE_MAX_DIM = 512

    # Number of training steps per epoch
    STEPS_PER_EPOCH = 1000

    VALIDATION_STEPS = 50

    # Backbone network architecture
    # Supported values are: resnet50, resnet101.
```

```
    BACKBONE = 'resnet50'

    # 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

  # Length of square anchor side in pixels
    RPN_ANCHOR_SCALES = (8, 16, 64, 128, 256)

    # Minimum probability value to accept a detected instance
    DETECTION_MIN_CONFIDENCE = 0.9
    WEIGHT_DECAY = 0.0001
config = TutaConfig()
config.display()
```

Custom function to load the dataset

```
class TutaDataset(utils.Dataset):

    def load_dataset(self, dataset_dir, subset):

        # Add classes. We have only one class to add.
        self.add_class("tuta", 1, "tuta")

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

        annotations = json.load(open(os.path.join(dataset_dir, "via_region_data.json")))
        annotations = list(annotations.values())

        # The VIA tool saves images in the JSON even if they don't have any
        # annotations. Skip unannotated images.
        annotations = [a for a in annotations if a['regions']]

        # Add images
        for a in annotations:
            # Get the x, y coordinaets of points of the polygons that make up
            if type(a['regions']) is dict:
                polygons = [r['shape_attributes'] for r in a['regions'].values()]
            else:
                polygons = [r['shape_attributes'] for r in a['regions']]

            # load_mask() needs the image size to convert polygons to masks.
            image_path = os.path.join(dataset_dir, a['filename'])
```

```
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]

            self.add_image(
                "tuta",
                image_id=a['filename'],
                path=image_path,
                width=width, height=height,
                polygons=polygons)

    def load_mask(self, image_id):
        image_info = self.image_info[image_id]
        if image_info["source"] != "tuta":
            return super(self.__class__, self).load_mask(image_id)

        # Convert polygons 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["polygons"])],
                        dtype=np.uint8)
        for i, p in enumerate(info["polygons"]):
            # Get indexes of pixels inside the polygon and set them to 1
            rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'])
            mask[rr, cc, i] = 1

        # Return mask, and array of class IDs of each instance.
        return mask.astype(np.bool), np.ones([mask.shape[-1]], dtype=np.int32)

    def image_reference(self, image_id):
        """Return the path of the image."""
        info = self.image_info[image_id]
        if info["source"] == "tuta":
            return info["path"]
        else:
            super(self.__class__, self).image_reference(image_id)
```

Load the training and validation dataset

```
# Training dataset.
with tf.device(device_name):
  dataset_train = TutaDataset()
  dataset_train.load_dataset('Experiments/TutaDatasetVIA/', "train")
  dataset_train.prepare()
  print("Images with annotations: {}\nClasses: {}".format(len(dataset_train.image_ids), datase
t_train.class_names))
# Validation dataset
dataset_val = TutaDataset()
dataset_val.load_dataset('Experiments/TutaDatasetVIA/', 'test')
dataset_val.prepare()
print("Images with annotations: {}\nClasses: {}".format(len(dataset_val.image_ids), dataset_va
l.class_names))
```

Build the Model

```
# Create model in training mode
with tf.device(device_name):
  model = modellib.MaskRCNN(mode="training",
                            config=config,
                            model_dir=MODEL_DIR)
  model.keras_model.summary()
```

Train the Model

```
# Train the head branches
# Passing layers="heads" freezes all layers except the head
# layers.
print("Training network heads")
start_train = time.time()
model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=20,
            layers='heads'
                        )
history = model.keras_model.history.history
end_train = time.time()
minutes = round((end_train - start_train) / 60, 2)
print(f'Training took {minutes} minutes')
```

Use our developed model for detection

```
class InferenceConfig(TutaConfig):
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

inference_config = InferenceConfig()

# Recreate the model in inference mode
```

```
with tf.device(device_name):
  model = modellib.MaskRCNN(mode="inference",
                            config=inference_config,
                            model_dir=MODEL_DIR)
model_path = model.find_last()
```

Using our developed model for detection

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

Appendix 2: U-Net Model Source Code

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
```

Load images with their masks

```
masks = glob.glob("Annotations/*.png")
og_images = glob.glob("Images/*.JPG")
```

Resizing the images and their corresponding masks

```
with tf.device(device_name):
  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)
```

Compile the model

```
from keras.optimizers import Adam, SGD
from keras_unet.metrics import iou, jaccard_coef, dice_coef
from keras_unet.losses import jaccard_distance
with tf.device(device_name):
  model.compile(
      optimizer=Adam(),
      #optimizer=SGD(lr=0.01, momentum=0.99),
      loss='binary_crossentropy',
      #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=1000,
    epochs=200,
    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')
```

Appendix 3: Tensorflow Lite Converter Source Code

Import TensorFlow

```
import tensorflow as tf
```

Defining the custom metrics since they are not save with the model during development

```
#iou
def iou(y_true, y_pred, smooth=1.):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) - intersection + smooth)

#iou thresholded
def iou_thresholded(y_true, y_pred, threshold=0.5, smooth=1.):
    y_pred = threshold_binarize(y_pred, threshold)
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) - intersection + smooth)

#dice coefficient
def dice_coef(y_true, y_pred, smooth=1.):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (
                K.sum(y_true_f) + K.sum(y_pred_f) + smooth)
```

Convert the Keras model to a TensorFlow Lite model and write the .tflite file

```
# Convert the model.
converter    =    tf.compat.v1.lite.TFLiteConverter.from_keras_model_file('segm_model_v3.h5',
custom_objects={'iou':iou, 'iou_thresholded':iou_thresholded, 'dice_coef':dice_coef})
tflite_model = converter.convert()

# Save the model.
with open('unet_model.tflite', 'wb') as f:
  f.write(tflite_model)
```

Check the output data type and shape.

```
import tensorflow as tf
import numpy as np

interpreter = tf.lite.Interpreter(model_path="unet_model.tflite")
interpreter.allocate_tensors()

print(interpreter.get_input_details()[0]['shape'])
print(interpreter.get_input_details()[0]['dtype'])
```

```
print(interpreter.get_output_details()[0]['shape'])
print(interpreter.get_output_details()[0]['dtype'])
```

Adding metadata to tflite model

```
!pip install tflite-support
```

Model information: Metadata starts by creating a new model info:

```
from tflite_support import flatbuffers
from tflite_support import metadata as _metadata
from tflite_support import metadata_schema_py_generated as _metadata_fb


""" ... """
"""Creates the metadata for an image classifier."""

# Creates model info.
model_meta = _metadata_fb.ModelMetadataT()
model_meta.name = "UNet Image Segmentation"
model_meta.description = ("Detect and locate affected areas in "
                          "tomato leaf image caused by "
                          "the tomato leaf miner tuta absoluta")
model_meta.version = "v1"
model_meta.author = "Loyan"
model_meta.license = ("Apache License. Version 2.0 "
                      "http://www.apache.org/licenses/LICENSE-2.0.")
```

Input / output information: This section shows you how to describe your model's input and output signature. This metadata may be used by automatic code generators to create pre- and post- processing code. To create input or output information about a tensor:

```
# Creates input info.
input_meta = _metadata_fb.TensorMetadataT()

# Creates output info.
output_meta = _metadata_fb.TensorMetadataT()
Image input
input_meta.name = "image"
input_meta.description = (
    "Input image to be segmented. The expected image is {0} x {1}, with "
    "three channels (red, blue, and green) per pixel. Each value in the "
    "tensor is a single byte between 0 and 255.".format(512, 512))
input_meta.content = _metadata_fb.ContentT()
input_meta.content.contentProperties = _metadata_fb.ImagePropertiesT()
input_meta.content.contentProperties.colorSpace = (
    _metadata_fb.ColorSpaceType.RGB)
input_meta.content.contentPropertiesType = (
```

```
    _metadata_fb.ContentProperties.ImageProperties)
input_normalization = _metadata_fb.ProcessUnitT()
input_normalization.optionsType = (
    _metadata_fb.ProcessUnitOptions.NormalizationOptions)
input_normalization.options = _metadata_fb.NormalizationOptionsT()
input_normalization.options.mean = [127.5]
input_normalization.options.std = [127.5]
input_meta.processUnits = [input_normalization]
input_stats = _metadata_fb.StatsT()
input_stats.max = [255]
input_stats.min = [0]
input_meta.stats = input_stats
```

Model Path

```
model_file = "unet_model.tflite"
```

Label output: Label can be mapped to an output tensor via an associated file using TENSOR_AXIS_LABELS.

```
import os
# Creates output info.
output_meta = _metadata_fb.TensorMetadataT()
output_meta.name = "probability"
output_meta.description = "Probabilities of the 1001 labels respectively."
output_meta.content = _metadata_fb.ContentT()
output_meta.content.content_properties = _metadata_fb.FeaturePropertiesT()
output_meta.content.contentPropertiesType = (
    _metadata_fb.ContentProperties.FeatureProperties)
output_stats = _metadata_fb.StatsT()
output_stats.max = [1.0]
output_stats.min = [0.0]
output_meta.stats = output_stats
label_file = _metadata_fb.AssociatedFileT()
label_file.name = os.path.basename("labels.txt")
label_file.description = "Labels for objects that the model can recognize."
label_file.type = _metadata_fb.AssociatedFileType.TENSOR_AXIS_LABELS
output_meta.associatedFiles = [label_file]
```

Create the metadata Flatbuffers: The following code combines the model information with the input and output information

```
# Creates subgraph info.
subgraph = _metadata_fb.SubGraphMetadataT()
subgraph.inputTensorMetadata = [input_meta]
subgraph.outputTensorMetadata = [output_meta]
model_meta.subgraphMetadata = [subgraph]
```

```
b = flatbuffers.Builder(0)
b.Finish(
    model_meta.Pack(b),
    _metadata.MetadataPopulator.METADATA_FILE_IDENTIFIER)
metadata_buf = b.Output()
```

Pack metadata and associated files into the model: Once the metadata Flatbuffers is created, the metadata and the label file are written into the TFLite file via the populate method

```
populator = _metadata.MetadataPopulator.with_model_file(model_file)
populator.load_metadata_buffer(metadata_buf)
populator.load_associated_files(["labels.txt"])
populator.populate()
```

Visualize the metadata

```
displayer = _metadata.MetadataDisplayer.with_model_file(model_file)
export_json_file = os.path.join(FLAGS.export_directory,
                    os.path.splitext(model_basename)[0] + ".json")
json_file = displayer.get_metadata_json()
# Optional: write out the metadata as a json file
with open(export_json_file, "w") as f:
  f.write(json_file)
```

Appendix 4: Model Deployment Android Studio Source Code

ImageSegmentationModelExecutor.kt

```
package org.tensorflow.lite.examples.tutaSegmentation.tflite

import android.content.Context
import android.graphics.Bitmap
import android.graphics.Color
import android.os.SystemClock
import androidx.core.graphics.ColorUtils
import android.util.Log
import java.io.FileInputStream
import java.io.IOException
import java.nio.ByteBuffer
import java.nio.ByteOrder
import java.nio.MappedByteBuffer
import java.nio.channels.FileChannel
import kotlin.random.Random
import org.tensorflow.lite.Interpreter
import org.tensorflow.lite.examples.TutaSegmentation.utils.ImageUtils
import org.tensorflow.lite.gpu.GpuDelegate

/**
 * Class responsible to run the Image Segmentation model.
 */
class ImageSegmentationModelExecutor(
 context: Context,
 private var useGPU: Boolean = false
) {
 private var gpuDelegate: GpuDelegate? = null

 private val segmentationMasks: ByteBuffer
 private val interpreter: Interpreter

 private var fullTimeExecutionTime = 0L
 private var preprocessTime = 0L
 private var imageSegmentationTime = 0L
 private var maskFlatteningTime = 0L

 private var numberThreads = 4

 init {

  interpreter = getInterpreter(context, imageSegmentationModel, useGPU)
  segmentationMasks = ByteBuffer.allocateDirect(1 * imageSize * imageSize * NUM_CLASSES * 4)
  segmentationMasks.order(ByteOrder.nativeOrder())
 }

 fun execute(data: Bitmap): ModelExecutionResult {
  try {
   fullTimeExecutionTime = SystemClock.uptimeMillis()

   preprocessTime = SystemClock.uptimeMillis()
   val scaledBitmap =
    ImageUtils.scaleBitmapAndKeepRatio(
     data,
     imageSize, imageSize
    )

   val contentArray =
    ImageUtils.bitmapToByteBuffer(
     scaledBitmap,
     imageSize,
     imageSize,
     IMAGE_MEAN,
     IMAGE_STD
    )
   preprocessTime = SystemClock.uptimeMillis() - preprocessTime

   imageSegmentationTime = SystemClock.uptimeMillis()
   interpreter.run(contentArray, segmentationMasks)
   imageSegmentationTime = SystemClock.uptimeMillis() - imageSegmentationTime
```

```
    Log.d(TAG, "Time to run the model $imageSegmentationTime")

    maskFlatteningTime = SystemClock.uptimeMillis()
    val (maskImageApplied, maskOnly, itemsFound) =
      convertBytebufferMaskToBitmap(
        segmentationMasks, imageSize, imageSize, scaledBitmap,
        segmentColors
      )
    maskFlatteningTime = SystemClock.uptimeMillis() - maskFlatteningTime
    Log.d(TAG, "Time to flatten the mask result $maskFlatteningTime")

    fullTimeExecutionTime = SystemClock.uptimeMillis() - fullTimeExecutionTime
    Log.d(TAG, "Total time execution $fullTimeExecutionTime")

    return ModelExecutionResult(
      maskImageApplied,
      scaledBitmap,
      maskOnly,
      formatExecutionLog(),
      itemsFound
    )
  } catch (e: Exception) {
    val exceptionLog = "something went wrong: ${e.message}"
    Log.d(TAG, exceptionLog)

    val emptyBitmap =
      ImageUtils.createEmptyBitmap(
        imageSize,
        imageSize
      )
    return ModelExecutionResult(
      emptyBitmap,
      emptyBitmap,
      emptyBitmap,
      exceptionLog,
      HashMap<String, Int>()
    )
  }
}

@Throws(IOException::class)
private fun loadModelFile(context: Context, modelFile: String): MappedByteBuffer {
  val fileDescriptor = context.assets.openFd(modelFile)
  val inputStream = FileInputStream(fileDescriptor.fileDescriptor)
  val fileChannel = inputStream.channel
  val startOffset = fileDescriptor.startOffset
  val declaredLength = fileDescriptor.declaredLength
  val retFile = fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength)
  fileDescriptor.close()
  return retFile
}

@Throws(IOException::class)
private fun getInterpreter(
  context: Context,
  modelName: String,
  useGpu: Boolean = false
): Interpreter {
  val tfliteOptions = Interpreter.Options()
  tfliteOptions.setNumThreads(numberThreads)

  gpuDelegate = null
  if (useGpu) {
    gpuDelegate = GpuDelegate()
    tfliteOptions.addDelegate(gpuDelegate)
  }

  return Interpreter(loadModelFile(context, modelName), tfliteOptions)
}

private fun formatExecutionLog(): String {
  val sb = StringBuilder()
  sb.append("Input Image Size: $imageSize x $imageSize\n")
  sb.append("Model execution time: $imageSegmentationTime ms\n")
  sb.append("Mask flatten time: $maskFlatteningTime ms\n")
```

```
  sb.append("Full execution time: $fullTimeExecutionTime ms\n")
  return sb.toString()
 }

 fun close() {
  interpreter.close()
  if (gpuDelegate != null) {
   gpuDelegate!!.close()
  }
 }

 private fun convertBytebufferMaskToBitmap(
  inputBuffer: ByteBuffer,
  imageWidth: Int,
  imageHeight: Int,
  backgroundImage: Bitmap,
  colors: IntArray
 ): Triple<Bitmap, Bitmap, Map<String, Int>> {
  val conf = Bitmap.Config.ARGB_8888
  val maskBitmap = Bitmap.createBitmap(imageWidth, imageHeight, conf)
  val resultBitmap = Bitmap.createBitmap(imageWidth, imageHeight, conf)
  val scaledBackgroundImage =
   ImageUtils.scaleBitmapAndKeepRatio(
    backgroundImage,
    imageWidth,
    imageHeight
   )
  val mSegmentBits = Array(imageWidth) { IntArray(imageHeight) }
  val itemsFound = HashMap<String, Int>()
  inputBuffer.rewind()

  for (y in 0 until imageHeight) {
   for (x in 0 until imageWidth) {
    var maxVal = 0f
    mSegmentBits[x][y] = 0

    for (c in 0 until NUM_CLASSES) {
     val value = inputBuffer
      .getFloat((y * imageWidth * NUM_CLASSES + x * NUM_CLASSES + c) * 4)
     if (c == 0 || value > maxVal) {
      maxVal = value
      mSegmentBits[x][y] = c
     }
    }
    val label = labelsArrays[mSegmentBits[x][y]]
    val color = colors[mSegmentBits[x][y]]
    itemsFound.put(label, color)
    val newPixelColor = ColorUtils.compositeColors(
     colors[mSegmentBits[x][y]],
     scaledBackgroundImage.getPixel(x, y)
    )
    resultBitmap.setPixel(x, y, newPixelColor)
    maskBitmap.setPixel(x, y, colors[mSegmentBits[x][y]])
   }
  }

  return Triple(resultBitmap, maskBitmap, itemsFound)
 }

 companion object {

  public const val TAG = "SegmentationInterpreter"
  private const val imageSegmentationModel = "unet_model.tflite"
  // private const val imageSize = 257
  private const val imageSize = 512
  // const val NUM_CLASSES = 21
  const val NUM_CLASSES = 2
  private const val IMAGE_MEAN = 127.5f
  private const val IMAGE_STD = 127.5f

  val segmentColors = IntArray(NUM_CLASSES)
  val labelsArrays = arrayOf(
       "_background_", "tuta"
  )
```

```
    init {

      val random = Random(System.currentTimeMillis())
      segmentColors[0] = Color.TRANSPARENT
      for (i in 1 until NUM_CLASSES) {
        segmentColors[i] = Color.argb(
          (128),
          getRandomRGBInt(
            random
          ),
          getRandomRGBInt(
            random
          ),
          getRandomRGBInt(
            random
          )
        )
      }
    }

    private fun getRandomRGBInt(random: Random) = (255 * random.nextFloat()).toInt()
  }
}
```

Appendix 5: Mask RCNN Object Counting Source Code.

Import libraries

```
import random
import itertools
import colorsys #defines conversion of color values
import numpy as np
from skimage.measure import find_contours
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon
import IPython.display
import cv2

from mrcnn import utils
```

Define classes

```
class_names = ['BG', 'tuta']
```

Visualization functions

#Display the given set of images

```
def display_images(images, titles=None, cols=4, cmap=None, norm=None,
                   interpolation=None):
    titles = titles if titles is not None else [""] * len(images)
    rows = len(images) // cols + 1
    plt.figure(figsize=(14, 14 * rows // cols))
    i = 1
    for image, title in zip(images, titles):
        plt.subplot(rows, cols, i)
        plt.title(title, fontsize=9)
        plt.axis('off')
        plt.imshow(image.astype(np.uint8), cmap=cmap,
                   norm=norm, interpolation=interpolation)
        i += 1
    plt.show()
```

#Get the detected objects, draw contours and add text

```
def get_masked_fixed_color(image, boxes, masks, class_ids, class_names,
                      colors=None, scores=None, title="",
                      figsize=(16, 16), ax=None, show=True):
    objects = dict()

    # Number of instances
    N = boxes.shape[0]
    if not N:
        print("\n*** No instances to display *** \n")
    else:
```

```
        assert boxes.shape[0] == masks.shape[-1] == class_ids.shape[0]

    # Generate random colors
    if colors == None:
       # classN = len(class_names)
       classN = ['blue', 'purple', 'red', 'green', 'orange', 'salmon', 'pink', 'gold',
                      'orchid', 'slateblue', 'limegreen', 'seagreen', 'darkgreen', 'olive',
                      'teal', 'aquamarine', 'steelblue', 'powderblue', 'dodgerblue', 'navy',
                      'magenta', 'sienna', 'maroon']
       colors = random_colors(classN)
      # colors="Red"

    masked_image = np.array(image)

    for i in range(N):
        color = colors[class_ids[i]]

        # Bounding box
        if not np.any(boxes[i]):
            # Skip this instance. Has no bbox. Likely lost in image cropping.
            continue
        y1, x1, y2, x2 = boxes[i]
        cv2.rectangle(masked_image, (x1, y1), (x2, y2), (255,255,0), thickness = 1)

        # Label
        class_id = class_ids[i]
        score = scores[i] if scores is not None else None
        label = class_names[class_id]
        if(label in objects):
            objects[label] += 1
        else:
            objects[label] = 1
        x = random.randint(x1, (x1 + x2) // 2)
        caption = "{} {:.3f}".format(label, score) if score else label
        cv2.putText(masked_image, caption, (x1 + 5, y1 + 16), cv2.FONT_HERSHEY_SIMPLEX, 0.4,
(255,0,0))

        # Mask
        mask = masks[:, :, i]
        if show:
            masked_image = apply_mask(masked_image, mask, color)

            # Mask Polygon
            # Pad to ensure proper polygons for masks that touch image edges.
            padded_mask = np.zeros((mask.shape[0] + 2, mask.shape[1] + 2), dtype=np.uint8)
            padded_mask[1:-1, 1:-1] = mask
            contours = find_contours(padded_mask, 0.5)
            for verts in contours:
                # Subtract the padding and flip (y, x) to (x, y)
                verts = np.fliplr(verts) - 1
                verts = verts.reshape((-1, 1, 2)).astype(np.int32)
```

```
                # Draw an edge on object contour
                cv2.polylines(masked_image, verts, True, color)

    print(str(objects))
    cv2.putText(masked_image, str(objects), (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.4,
(255,0,0))

    return masked_image
```

#Object counting using OpenCV

```
import cv2
import time

colors = random_colors(len(class_names))

#image1 = cv2.imread('input_images_and_videos/input.png')
image1 = cv2.imread('/content/drive/My
Drive/Experiments/TutaDatasetVIA/test/BLK_1_0924_PL002_NH.JPG')

image1 = cv2.resize(image1, None, fx=0.5, fy=0.5)

image_batch = [image1]

# Run detection
t = time.time()
results = model.detect(image_batch, verbose=0)
t = t - time.time()
print (t)

masked_image_batch = []
# Visualize results
r = results[0]
t = time.time()

for i in range(len(results)):
    r = results[i]
    im = image_batch[i]
    masked_image = get_masked_fixed_color(im, r['rois'], r['masks'], r['class_ids'],
class_names, colors, r['scores'], show=True)
    masked_image = cv2.resize(masked_image, None, fx=3, fy=3)
    masked_image_batch.append(masked_image)

t = t - time.time()
print (t)

result = cv2.imwrite("/content/drive/My
Drive/Experiments/TutaDatasetVIA/object_counting_results_resnet101aug/BLK_1_0924_PL002_NH.png"
, masked_image_batch[0])
```

RESEARCH OUTPUTS

Research Output 1: Publications

Loyani, L., Bradshaw, K., & Machuve, D. (2021). Segmentation of Tuta absoluta's damage on tomato plants: A computer vision approach. *Applied Artificial Intelligence. 2021*, 1-21. https://doi.org/10.1080/08839514.2021.1972254

Loyani, L., & Machuve D. (2021). A Deep Learning-based Mobile Application for Segmenting Tuta absoluta's Damages on Tomato Plants. *Engineering, Technology and Applied Science Research*, *11*(5), 7730-7737. https://doi.org/10.48084/etasr.4355

Research Output 2: Poster Presentations

(i) Decision Support System for Farmers against *Tuta Absoluta*'s Effects on Tomato Plants

(ii) A Deep Learning Approach for Quantifying *Tuta Absoluta*'s damage on Tomato Plants