UNIVERSITY OF NAIROBI

AN OPTIMIZED DEEP LEARNING MODEL FOR PREDICTION OF TOMATO LEAF DISEASES IN KENYA

BY

NDUNGU PAUL WANYANGA

P52/3460/2019

SUPERVISOR: DR. EVANS A.K. MIRITI

August 2021

A Research Project Submitted for the partial fulfillment for the requirements for the award of Degree of Masters in Computational Intelligence at the School of Computing and Informatics, University of Nairobi

DECLARATION

STUDENT

I hereby declare that this thesis is my own work and has, to the best of my knowledge, not been submitted to any other institution of higher learning.

Name: Ndungu Paul Wanyanga **Registration Number**: P52/34260/2019

Signature: [illegible] **Date:** 18/08/2021

SUPERVISOR

This research project has been submitted for examination towards fulfilment for the award of degree of Masters in Computational Intelligence with the approval of my supervisor.

Dr. Evans A.K. Miriti
School of Computing and Informatics
University of Nairobi

Signature: [illegible] **Date:** 19-08-2021

DEDICATION

I dedicate this research to God for his grace and good health throughout the research, to my loving wife Brenda Nyambura for her continued support throughout, my supervisor Dr. Evans A.K. Miriti for the dedication and support through the research process and friends and family who supported me through the journey.

ACKNOWLEDGEMENT

I would like to sincerely acknowledged the efforts and support from my wife Brenda Nyambura for the continuous encouragement, support and patience throughout the research process. I would also like to acknowledge the support and advise offered by my supervisor Dr. Evans A.K. Miriti. It was through this advice that I was able to shape and complete my research project. I would also like to acknowledge the support and help from KARLO and in particular from Dr. Ruth Amata for the great advice and help on Tomato production in Kenya. Also, to all my friends and family who in one way or another contributed to the success of this research project.

ABSTRACT

Problem: Recent years have seen an accelerated growth in use of small devices and intelligent applications in agriculture as well as other sectors. Due to this increased use of intelligence in small devices, there has been an increased demand for machine learning models that can work within resource constrained environments. Driven by the success achieved by deep learning models across different sectors, there has been a natural trend of pushing deep learning models in the direction of mobile application and use in small devices such as micro-controllers. However, this has not been an easy task. This is attributed to the difference that exist between the constrained computational environment of these small devices and the intense resource requirement of deep learning networks. As a result, there has been a need to optimize the existing deep learning models for use in small devices without significant loss of accuracy.

Objective: The objective of this study was to develop an optimized deep learning model for the prediction of tomato leaf diseases that can be used in low-end small devices.

Methodology: CRISP-DM methodology was used in building the initial TensorFlow model. A ResNet – 50 model was adopted and through use of Transfer Learning custom layers were built for the research. TensorFlow-Lite framework was used for the conversion and optimization processes of the model. Float16 Quantization was adopted for the optimization of the TensorFlow Model. Performance metrics including accuracy, precision and recall were used for model evaluation.

Results: The initial TensorFlow model achieved an accuracy of 95.93% while the converted and optimized model achieved an accuracy of 93.75%. After optimization, the initial model was reduced from 245 MB to 61 MB representing a 75% reduction in memory size. The accuracy loss was minimal at only 2.18% and still within incredible accuracy range.

Conclusion: The study concluded that quantization should be adopted as a standard practice for deep learning models in Agriculture to enhance easy deployment and accelerated adoption and use of deep learning models among small scale tomato farmers. The study recommended full deployment of the model as a mobile application and also the exploration and comparison of different quantization approaches which were not covered by the study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CNN - Convolutional Neural Network

GPU - Graphical Processing Unit

CRISP-DM – Cross-Industry for Data Mining

KARLO – Kenya Agricultural & Livestock Research Organization

MLP - Multilayer Perceptron

RF – Random Forest

CHAPTER 1 INTRODUCTION

1.0 Background

The Agricultural sector contributes about 26% of Kenya's gross domestic product (GDP) making it one of the most significant sectors within the economy OECD (2016). Kenya's agriculture is mainly in the form of small-scale farming and concentrated on farms averaging 0.2 to 3 ha and on a subsistence basis. The small-scale farming activities account for over 70% of all the agricultural production and sustains about 75% of the national food demand FAO (2017). As a result, success of small-scale farming is critical to achieving food security. As demand for food grows in Kenya, urban populations are increasingly depending on small-scale farmers for their household supply of food. The small-scale farmers play a key part of producing food for the urban and peri-urban populations.

Among the crops grown on small scale, a common one is Tomato. Tomato is not only widely grown as a vegetable crop in Kenya but also in the world at large. Tomato is one of the most extensively used vegetable crops among Kenyans. It is used both in its raw state as well as in processed form and made available in the form of tomato paste. Tomato growing plays a critical role in achieving nutritional requirements, generation of income and creation of employment in both rural and urban populations. However, tomato production faces serious challenges related to pests and diseases. In fact, it is reported that Tomato farmers experience 80-100% crop losses due to pests and crop diseases FAO (2019).

Pest and diseases remain to be a significant challenge and a big hindrance to the success of small-scale farming. Pest and diseases are responsible for heavy losses through death of crops and reduced productivity. Pests and diseases can contribute from 50 percent to total crop failure FAO (2019). As a result, measures and actions aimed at reducing the impact caused by pests and diseases play an important role in enhancing productivity by small-scale farmers.

In addition to managing pests and diseases, early detection of plant diseases is crucial in preventing large scale damage to crops as early treatment will prevent spread of the diseases or advancing of the diseases to untreatable levels. Most small-scale farmers depend on experience and human observation in identifying crop infestation. While effective, these methods are most of the time inaccurate or characterized by misinformation of the actual problem.

Advancement in technology has provided an opportunity for adoption of better crop diseases management practices. Among these methods, is the use of machine learning and deep learning models embedded in smartphones and other small devices and can be used in real-time within the farm. For example, there has been an attempt by researchers at Meru University to build a system that can detect attacks on Tomatoes in real time and send an SMS alert to the farmer MUST (2016). While these are feasible solutions, they have faced a huge challenge of adoption as they are intensive respect to size and computation requirements. As a result, there is an urgent need for improving these models through optimizations to make them available to farmers who mostly have low-cost smart phone and can barely handle the models as they are.

1.1 Problem statement

With advances in technology, smart farming has adopted the use of deep learning and machine learning in real time prediction of crop diseases and pests. However, most of these adoptions do not check parameters such as computation time and hardware restrictions that come with these models. This overlook makes it impractical to implement these models at the farm level and at the convenience of the farmer. Due to huge computation requirements, its is quite impractical to use these models on small devices that are easily available to the farmer. These challenges make the adoption of these accurate technique at the farm level very problematic or even impossible due to resource requirements.

Past research solutions that use deep learning have focused on improving accuracy of the models while neglecting the need for optimizing the models. An empirical study conducted by Mengwei et al. (2019) indicated that out of 211 deep learning-based applications, only 6.32% attempted to apply any optimization techniques. This overlook makes the adoption of deep learning in small devices quite challenging. This is due to the increased memory size and huge computational requirements demanded by the non-optimized deep learning models. For example, a deep learning framework proposed by Pallagani et al. (2020) although having an accuracy of 99.24% required a minimum memory of 150 MB, daylight conditions and a minimum of 8 megapixels camera.

The solution that has been proposed for real time monitoring of tomatoes leaf disease MUST (2016) has its shortcomings. The system was not optimized for small devices and dependent on a GSM module where the system would alert the farmer through SMS in case of pest attack. In addition, the system was only capable of detecting one tomato leaf disease and thus limited in scale of use.

1.2 Research objectives

The main objective of this research study was to develop a prediction model based on deep learning for early detection of tomato diseases that had reduced computation requirements and could be easily used with small devices such as smartphones.

1.2.1 Specific objectives

I. To reduce the memory size and computational requirements of current deep learning-based disease prediction model for tomato leaf disease through integer quantization.
II. To improve generalizability of existing deep learning models in tomato leaf disease classification through use of separate training and test datasets.
III. To improve the accuracy of existing local models through use of transfer learning.

1.2.2 Research questions

I. How can deep learning models for tomato foliar disease prediction be optimized to work with small devices such as smartphones?
II. What are the best practices for improving the generalizability of a deep learning model?
III. What model enhancement approach can be adopted to increase the accuracy of existing deep learning models for tomato leaf disease prediction?

1.3 Research justification

While previously deep learning solutions have been proposed for disease identification, they tend to suffer from increased memory and computational requirements. This limitation makes its hard to deploy the deep learning models in small devices such as smart phones. This challenge only gets worse with small-scale farmers who tend to mostly have low-end phones as they are affordable. With these limited resources, it’s not ideal to deploy heavy deep learning models using these phones. This calls for research into ways of reducing the huge memory size and computational requirements that are used by these models. This study focused on the adoption of optimization techniques based on post training quantization to provide more information and best practices on optimization of deep learning models. Optimized models will be easy to deploy on the farmers phone and encourage use of the app autonomously.

Most proposed deep learning solutions for crop diseases identification mostly used public datasets that are often obtained under controlled conditions and they are used for training as well as testing Boulent et al. (2019). This possess a challenge in that while the models perform exceptionally well, they fail to generalize well when applied to a local environment for instance within the farm. This leads to misdiagnosis of the diseases hence limiting the use of deep learning model in the farm. This study will explore the use of a hybrid approach where the training data is extracted from an online database but the test data is obtained locally. With good generalizability, the model will be applicable across different local setting without fear of misdiagnosis.

In Kenya while highly needed, there is limited research on the use of deep learning models in automatic tomato diseases identification. The existing solutions applied to other crops such as late blight in potato and maize leaf disease have low accuracy. Given the sensitivity of the deep learning model as an early warning system to farmers, low accuracy can be very problematic when there is doubt in the type of disease identified. This study will make use of regularization techniques with a goal of increasing the model accuracy and efficiency. Once improved, the models will be more reliable and the farmers can use them confidently without fear of misclassification of diseases.

1.4 Scope of the Study

The research focused on small scale farmers in Kenya and specifically those farmers practicing tomato farming. The research was limited its analysis on early detection of diseases that can be detected from the leaf part of the plant. The data for training was extracted from the plant village dataset. For testing, images from a different setting were used. The study covered foliar diseases that affect tomato plants and relevant to the local situation. The data used for research was obtained from a public online dataset and the test images were separately obtained from local farms.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

Ease in crop disease identification is an integral part of plant health management. Adoption of computer vision and deep learning for crop disease classification have proven to be a state-of-the-art solution in addressing crop disease management now and in future. This literature review revolves around the optimization of deep learning models for use in small devices such as smart phones. The literature review looks at past deep learning solutions in crop health management, their limitations and possible areas of improvement. While the focus is on a local solution, the literature review covers global solutions with an attempt to adopt best practices for optimal performance of the proposed deep learning model and its adaption for use in small devices. From the literature, it is evident that a lot of research on use of deep learning models in crop classification has been conducted and tuning to enhance performance as well. However, the literature identifies a gap in the optimization of the deep learning for use with small devices such as smart phones. In addition, a need to use dataset from uncontrolled environment is identified as well as use of transfer learning and data augmentation for improving local deep learning models.

2.1 Automatic plant diseases identification in Kenya

Toroitich (2017) proposed an artificial neural network based on back propagation for the prediction of potato late blight disease. The model was based on temperature and humidity reading obtained from sensors placed in the farm. While these methods could be adapted for use in prediction of tomato diseases, it is a bit complex to set up and requires an expert to do so. This highly limits independent use of the system by the farmer. The system was set up using a web server and required the farmer to register and log in to access the information. The web-based application further limits the use of the system in that a farmer is required to have a computer and access to internet to be able to use it. Not many small-scale farmers have access to a computer and most run their farms using a mobile phone. The working of the system requires an expert to monitor the web server and alert the farmer in case of any anomaly detection. This approach is a bit inefficient as an early warning when dealing with small scale farmers due to delays in communication between the farmer and expert.

Maina (2016) proposed a vision model using artificial neural network and back propagation. In the study images of maize leaf and features were manually extracted for identification of the specific disease. This approach would expose the model to errors as compared to automatic feature extraction. While the model was made available on an android application, the researcher failed to capture any conversion method of the model and as such it is assumed the model is implemented as is. This hugely affects the execution time of the application given the model size and highly undermines its applicability in small scale farms. In addition, disease identification is highly dependent on good feature extraction, unfortunately artificial neural networks fall short in feature extraction as compared to deep convolutional neural networks.

2.1 Use of Deep learning in automatic plant disease identification

Julio et al. (2018) presented a quantitative prediction of the potato late blight using machine learning algorithms such as multilayer perceptron (MLP), deep learning convolutional neural networks (CNN), support vector regression and random forest (RF). The algorithms were trained using datasets extracted from multispectral data obtained at canopy level using unmanned aerial vehicle (UAVs) fitted with a digital camera. This approach is quite expensive and a bit impractical for small scale farms as the UAVs will need a wide flying area to capture different images. Although CNN performed well as compared to the other models, the method required the user to manually cut images from each plot for feeding into the model. This process is tedious and exposes the model to human errors. The researchers only used a dataset of 540 images which was divided into training, validation and test dataset. The use of the same training and test dataset was likely to expose the model to overfitting and a poor performance when used on unseen data from a different location.

Mohanty et al. (2016) presented a comprehensive deep learning model for automatic plant disease identification using the plantvillage dataset. While the model performed very well with accuracies getting to highs of 99.35%, the training, validation and test data were all obtained from the same dataset collected under controlled conditions. The model performed poorly when exposed to images different from those used for training, achieving an accuracy of 31% only. The researchers were able to present 38 classes of 14 different species of plants. While this is commendable, it quite impractical for the real-world due to the size of the models and computational resources required. The researcher reported the model taking lots of time to train, multiple hours on high performance GPU. Consequently, this indicates that the resulting model is quite heavy making it a challenge to implement the researcher's recommendation of deploying this model on a smartphone.

2.3 Tomato diseases classification and detection using deep learning

Brahimi, Boukhalfa & Moussaoui (2017) proposed a deep learning approach based on convolutional neural networks for predicting nine tomato diseases. By using transfer learning, they were able to achieve a high accuracy of 99 %. This demonstrates the need for using pre-trained models especially when the data available is not significantly large. The study proposed the use of occlusion techniques as a method for localizing the diseases which enhances better understanding of the diseases. The study was limited in that the computation time was high and the size of the model was not optimized for small devices. For instance, one of the best performing AlexNet model had a size of 201MB. From this study, we can identify two ways that this research could be improved, that is, use of occlusion techniques for disease localization and the optimization of the deep learning models for small devices.

Krishnaswamy et al. (2018) examined several deep learning models for tomato crop classification using pre-training. In particular two deep learning-based architectures were used, that is, AlexNet and VGG16 net. In their research they were able to look at the role of different items in the architecture. These items included; number of images and significance of hyperparameters, that is, minibatch size, weight and bias learnings rate. Despite the rigor in architecture setup, the research study did not explore any data augmentation methods which would have greatly improved the quality of the images. In addition, dropout technique was not used during training which is key in preventing the network from overfitting. The researchers were able to analyze classification accuracy and execution time. After controlling for the batch size, VGG16 net resulted in the best classification accuracy of 96.19% while AlexNet yielded 95.81%. While the accuracy was quite good, the study utilized images for training and testing from the same dataset. This fails to communicate the efficiency of the models when tested using different images from those used in training.

2.4 Optimization of deep learning models for classification.

In their study, Mengwei et al. (2019) represented an empirical study of deep learning apps with a goal of demystifying how smartphone apps take advantage of deep learning. The study was able to demonstrate that adoption of deep learning in mobile apps is a huge milestone and separates the market leaders. However, they noted that only a fraction of the over 170 apps had applied any optimization techniques. Only 6.32% of the models had been quantized while all the others were non-optimized. This illustrates the gap that is in existence where most deep learning models are yet to tap into optimization benefits. In fact, the empirical study urged immediate action aimed at fixing the missing optimizations.

In their study, Pallagani et al. (2020) trained a deep convolutional neural network on a public dataset using 54, 306 images made up of both diseased and healthy plant leaves. The trained model was able to identify 14 crop species with an accuracy of 99.24%. However, the study used training data and test data from the same source where images were captured under a controlled environment. This highly undermines the generalizability of the final model and it would not perform as well when exposed to different images. A ResNet 50 model was used and converted into a tensorflow .pb file which was then loaded to an android app. However, there was no attempt to optimize the model for efficient use in mobile phones. In fact, the minimum requirement included; a memory of 150 MB, a 2GB+ RAM, daylight conditions, Android 6.0+ and 8MP camera. This indicates that the system was heavy and no optimization was done to make it mobile friendly.

2.4.1 Post Training Quantization

While there are a couple of quantization approaches, the most preferred type is the post training quantization. This refers to the quantization of the resulting model after training is completed. In this case, the quantization parameters are usually calibrated offline by processing the trained model weights and activations produced by running inference on a sample data. As a result, no more training is required Wu et al. (2020). Post quantization is usually preferred as the networks will maintain accuracy after the quantization process. However, in case of accuracy loss partial quantization is advised where most sensitive layers are unquantized. Wu et al. (2020) proposes the following workflow for integer quantization; For the weights, use a scale quantization with per-column/ per-channel granularity, symmetric integer range for quantization and max calibration; For activation, use scale quantization with per-tensor granularity. For a pre-trained network, quantization of all computationally intensive layers and running of activation calibration is advised. If the calibration fails to achieve the desired accuracy, partial quantization should be carried out. There are three main techniques of post-training quantization as illustrated in the table below;

Techniques	**Benefits**	**Hardware**
Dynamic Range Quantization	4x smaller, 2x-3x speed up	CPU
Full integer quantization	4x smaller, 3x+ speed up	CPU, Edge, TPU, Microcontrollers
Float16 Quantization	2x smaller, GPU acceleration	CPU, GPU

Table 2.1: Post Training Techniques

While dynamic range quantization and full integer quantization have more benefits, they require use of a separate dataset whereas float16 does not requires use of a separate dataset for retraining the weights. In this case, in cases where data is not easily available, float16 quantization is highly recommended. In addition, post training float16 quantization reduces the tensor flow lite model size by up to 50% while sacrificing minimal accuracy. It works by reducing full precision floating point (32-bit) to a reduced precision floating data type (IEEE FP16). Due to its minimal effect on accuracy and significance reduction in model size, it is always recommended as a great starting point for quantizing TensorFlow Lite models (TensorFlow.org).

2.6 Summary of the Research Gap

Despite their success, deep learning models are faced with several challenges. Key among them is that the local solutions lack autonomy in that most are deployed on a web platform that is not accessible to the farmer and relies on an expert Toroitich (2017). They also face accuracy issues due to the use of manual feature detection and lack of data augmentation Maina (2016). Where deep learning has been implemented and the accuracy is high, there is low generalizability of the models when used on images different from those used during training Mohanty et al. (2016). Lastly, the model suffers from huge memory size and intense computational requirements which hinders easy deployment within small devices Pallagani et al. (2020) & Brahimi et al.

(2017). From the foregoing, the is a research opportunity in deep learning model optimization, use of different data set to improve generalizability of the models and use of automatic feature extraction coupled with transfer learning to improve the model accuracy. The need for optimization is echoed by the empirical study conducted by Mengwei et al. (2019), a review of deep learning models.

2.9 Conceptual Model

The conceptual model outlines how the different parts of the classification model interact. The data used in training the model is obtained from a public dataset provided by the Plant Village organization. The data undergoes pre-processing and is then fed to the ResNet Model for training. The resulting model is quantized using float16 quantization while being converted into a TensorFlow Lite model. Known data from farms is used to test the quantized model.

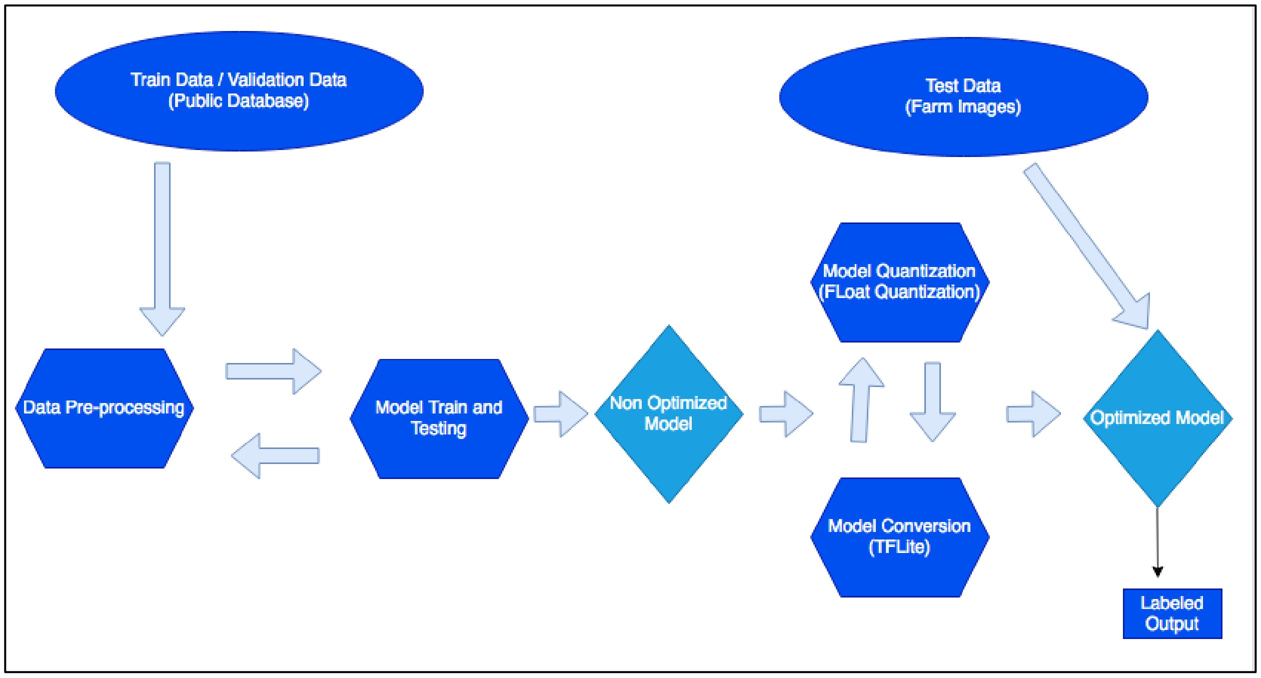


Figure 2.1 Conceptual Model

CHAPTER 3 RESEARCH METHODOLOGY

3.1 Introduction

This section covers the research design and research methodology that was applied in the study. Research methodology can be defined as the science of doing research. The research study used an applied approach taking advantage of deep learning and optimization techniques to answer the research questions. Secondary and primary data acquisition methods were applied to obtain data for the research project. Expert advice and guidance were sort from key organizations such as Kenya Agricultural & Livestock Research organization (KALRO). The data acquired from a public database provided by Plant Village organization was pre-processed in readiness for training. A deep learning architecture based on residual networks was applied in training the model. The model was then converted and optimized using the TensorFlow Lite framework. The best model was saved for use in predicting tomato leaf diseases.

3.2 Research Design

The research process was based on a standard data mining methodology. The general approach was to use a deep learning algorithm to train a model that would then be optimized through quantization to produces a model that has minimized memory size as compared to the initial TensorFlow Model.

To this end, a cross-industry process for data mining (CRISP-DM) methodology was applied in developing and deploying the model. The modelling part utilized convolutional neural networks with transfer learning based on the Res Net Architecture. Post training quantization was used in solving the problem of model optimization and the tensor flow lite framework was utilized for model conversion. For the prototype, deployment was demonstrated on the server side with a proposal of deploying the quantized model on a small device such as a smart phone.

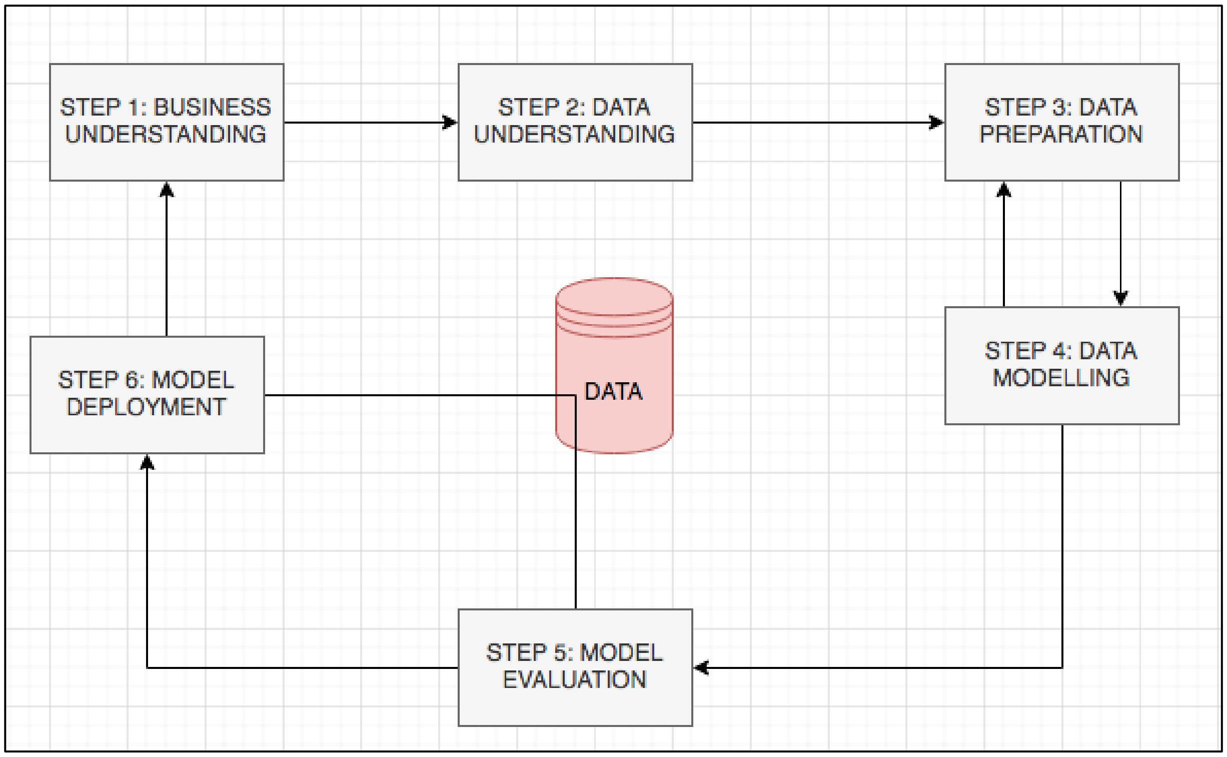


Figure 3.1 Steps of the CRISP-DM methodology

3.3 Business Understanding

Most small-scale farmers depend on extension officers and human experience in the detection and control of pest and crop diseases. Advanced technology has offered an opportunity for innovators to deploy deep learning models in farms for early detection of pests and diseases. However, the adoption of these technologies on edge has been met with huge challenges as the deep learning technologies requires lots of memory size and computation power which in most cases is not available with the phones that farmers use. Given this challenge, the research aim was to develop an optimized deep learning model that could be deployed on small devices such as phones with minimized memory size and computational requirements.

3.4 Data Understanding

The data used in this research was originally published on the online platform PlantVillage website but has since been deleted. As a result, the data is obtained from a different paper that have used and republished this data. The original data contains over 50, 000 well curated images of both healthy and infected leaves with an aim of encouraging computer vision approaches with an aim of solving the problem of yield losses. The original dataset has 34 different crops; however, our interest was in the images that pertain to tomato crop only. The tomato leaf images are selected from the original dataset where, 18,060 images are selected belong to 10 classes (nine disease categories and one healthy).

Class	Category	Number of Pictures
0	Bacterial Spot	2027
1	Early Blight	1000
2	Late Blight	1909
3	Mold Leaf	952
4	Septoria Leaf Spot	1771
5	Spider Mites	1676
6	Target Spot	1404
7	Tomato Yellow Curl Virus	5357
8	Tomato Mosaic Virus	373
9	Healthy	1591

Table 3.1: Data Categories

3.5 Data Preparation

The dataset was divided into three sets, that is, training set, validation set and testing set. The training set and validation test were obtained from the plantvillage public dataset. A split ratio of 0.2 was used, that is, 80% of the data was used for training the model while 20% was used for validating the model. The test set is made up of 16 images that were gathered differently from the train and validation sets. This is used to simulate local images that were supposed to be obtained separately from the train and validation set for the purpose of improving model generalization.

To improve image quality and quantity, the research depended greatly on the process of data augmentation. Data augmentation was applied using various methods such as; standardizing the image size, clipping and expanding, random rotation, and random color adjustment. Zooming to uniform size was also applied as the original images were of different sizes due to factors such as distance. The scaling to uniform size was useful in model training where a standard size is required as the initial input to the model. Random rotation was used to increase the angles of the images by defining specific angles as well as the direction of rotation. Color adjustment was also used to obtain uniformity given that the images were taken in different environments hence likely to have a bias in color. There was also the conversion of the three base colors RGB (red, blue, and green) in the images to HSV (hue, saturation, and color value).

In preparing the dataset for modelling, the images were resized to a standard size of 224*224 which is the required input for the ResNet 50. In addition, data augmentation was done with an aim of enlarging the dataset volume and highlight the regions of interest. The data augmentation processes that were used included, rescaling, zoom range, width shift range, height shift range, and shear range. With data preprocessing and data augmentation completed, the data was split into train and validation sets using a split of 0.2. This resulted into 18,345 train images and 4,585 validation images are obtained.

3.6 Model Design

In identification of tomato leaf disease, this study used a ResNet architecture with pre-trained weights which has not only proved a better classification architecture but also adopts skip connection which avoids the problem of accuracy saturation due to increasing network depth. In addition, ResNet models have shown good convergence behaviors and compelling accuracy. They are represented in different layers; 18,34,50,101,152 and 1202. With an objective of saving computing resources and training time, Resnet 50 was used since it provides a good balance between depth and performance.

The ResNet 50 model consist of 5 stages each with a convolution and identity block. Each convolution block has 3 convolution layers and each identity block has 3 convolution layers. The model works by accepting a two-dimensional image (224*224) as the input. The convolution layers were used to apply various filters to the input image. The model also applies smart activation using the ReLU activation function defined as max (0,x). A max-pooling layer is applied through maximum coupling for a 2*2 matrix. Batch normalization was also applied to standardize the inputs. A dropout technique was incorporated to take prevent overfitting.

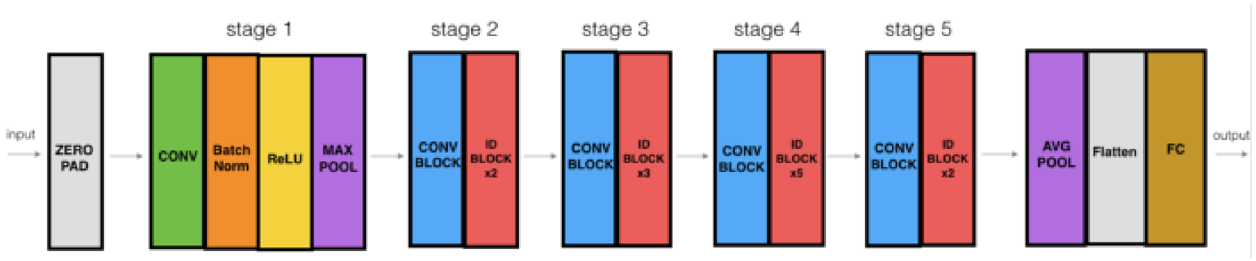


Figure 3.2 ResNet 50 Stages (source: Dwivedi, 2019)

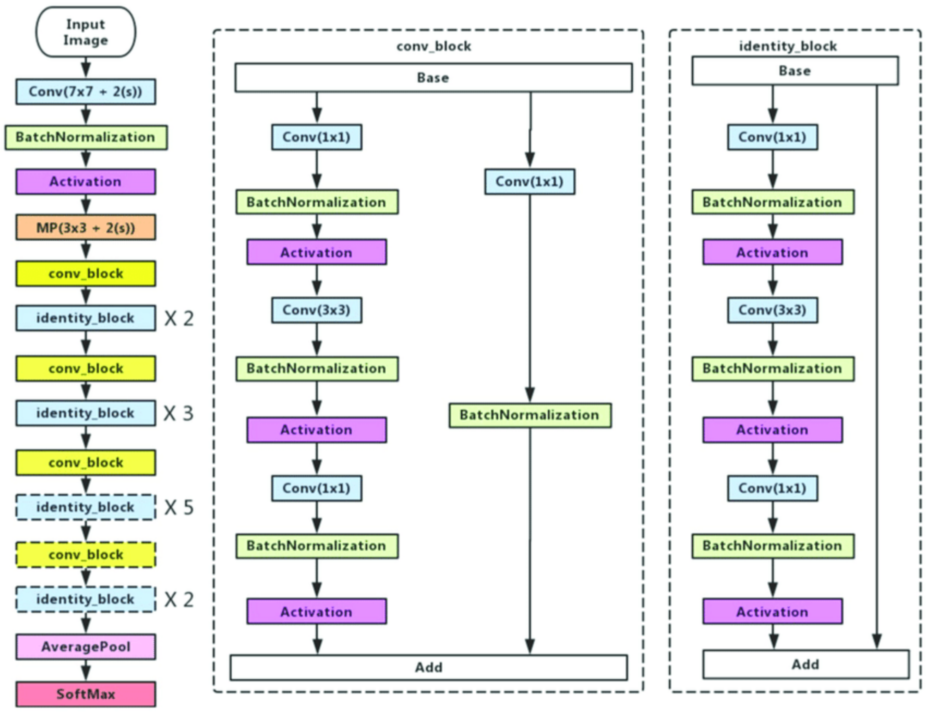


Figure 3.3 ResNet 50 Architecture (source: Yankui, 2019)

3.6.1 Transfer Learning

To improve accuracy and also reduce training time and computation power requirements, the research used transfer learning in the implementation of ResNet 50. In our case, the neural network model was first trained on the ImageNet dataset and we use the weights with our ResNet model. This is achieved by manually defining the fully connected layer such that we are able to output the required classes while taking advantage of the pretrained model. For the sake of this research, pre-trained weights without top layer weights have been used. The last layer is has then been replaced with our own layer.

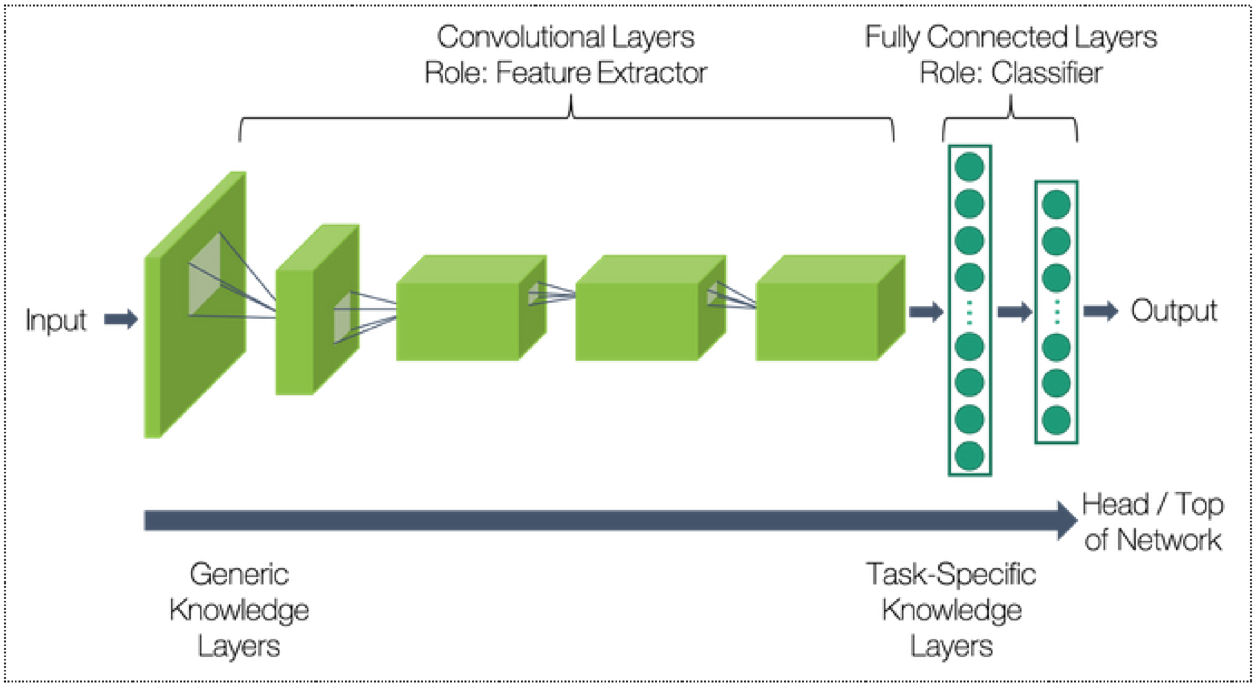


Figure 3.4: CNN Layers overview (source: Koul et al. 2019)

3.6.2 Model Implementation

Having applied transfer learning, the research compiled the model using a compiler function. The function was fed three parameters, that is, the optimizer, the loss function, and the metrics of performance. The optimizer used was rmsprop with the loss function being categorical cross entropy since the problem was a multi-classification. Early stopping was applied to prevent the network from overfitting as a result of many epochs and underfitting due too very few epochs. Considering that the last model might not be the best in terms of accuracy, model check point was applied where the best model observed during training would be saved based on the accuracy metric with the help of the model check point function. The TensorFlow model was evaluated using a validation dataset.

3.7 Model Evaluation

In evaluating the model, the research used 'loss' and 'accuracy' metrics. The loss was based on the categorical cross entropy loss function. The loss was calculated on training and validation and its interpretation based on how well the model was doing between the two data sets. On the other hand, the accuracy metric was used to measure the algorithm's performance in an interpretable way. The accuracy was determined in form of percentage and expressed how accurate the model was when tested against the validation data. For model validation, the fit method was applied with a validation split ratio of 0.2. This means that 80% of the images were used to train the model while 20% were used to validate the model. Accuracy plots were used to evaluate the model and see how it fits on the validation dataset. A confusion matrix was used to report on the accuracy of the model as well. The three main classification metrics that were used were; Accuracy, Precision and Recall. Accuracy can be defined as the percentage of correct predictions for the test data. It can be determined by dividing the number of correct predictions by the number of total predictions.

Accuracy *= correct predictions / all predictions*

Precision is defined as the fraction of relevant examples (true positives) against all the predicted examples belonging to a given class.

Precision *= true positives / (true positives + false positives)*

Recall can be defined as the fraction of examples that have been predicted to belong to a class with respect to all the examples within that class.

Recall *= true positives / (true positives + false negatives)*

These metrics will also be used as a measure of research quality

3.8 Optimization, Conversion and testing of the quantized model

As proposed earlier, the research applied post-training quantization with an aim of reducing the model size while at the same time improving CPU and hardware accelerator latency, with little degradation in model accuracy. In this case, the research did quantize the resulting TensorFlow model while converting it to TensorFlow lite format. This was achieved using the TensorFlow Lite Converter. To test the quantized model, a batch of 16 images was used. The interpreter class was used. The predictions were obtained by calling the get_tensor command. The predictions were visualized using a data frame.

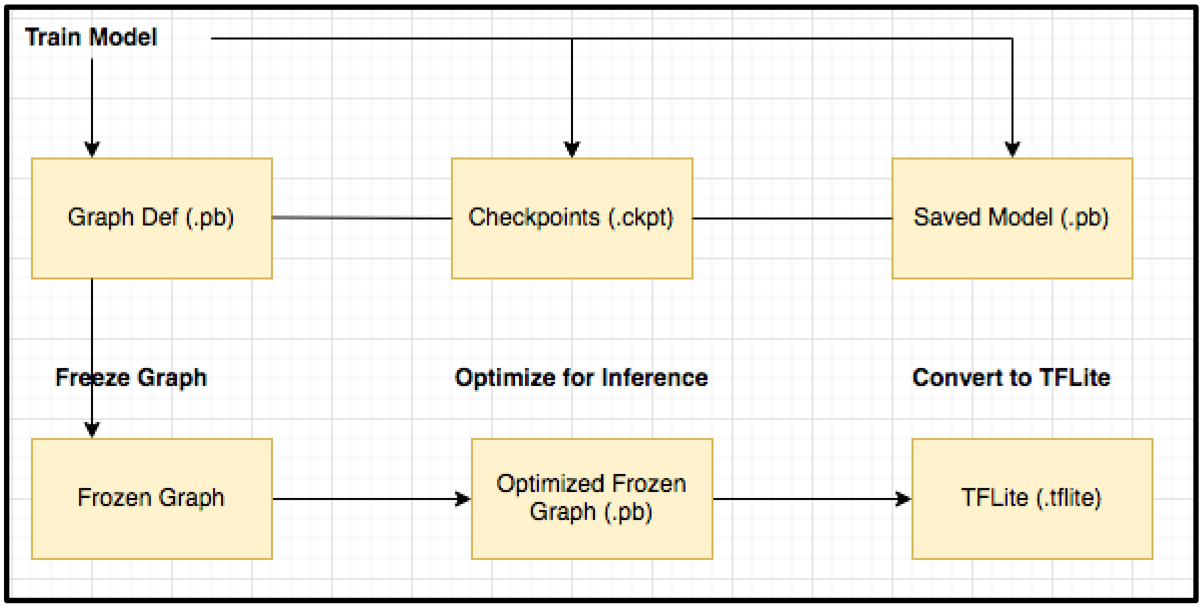


Figure 3.5 Model Conversion

3.9 Model Deployment

The research used post training quantization as an optimization method. The quantized model was then converted into a version that was lighter and could work in a constrained environment. In summary, the study took the trained model, froze the graph, optimized it for inference and then converted it into a .tflite file.

3.10 Evaluation of the Prototype

With the prototype complete, evaluation was done by testing its performance against the set research objectives. The prototype was evaluated in regards to the memory size. Respect to accuracy the predicted classes were compared against the predicted classes of the test data. Performance metrics were used for reporting the prototype prediction capability.

CHAPTER 4 RESULTS AND DISCUSSION

4.1 Introduction

This chapter discusses the results of the developed model in relation to the objectives as well as the research questions outlined in chapter one. The main objective of this research study was to develop a model based on deep learning for early detection of tomato leaf diseases that has reduced size and improved hardware accelerator latency. The research study was able to develop a TensorFlow model based on the residual networks (ResNet 50). This model was optimized through post-training quantization and a reduction in size was achieved. In particular, the model was reduced from an initial size of 245 MB to 61 MB. The other objective was to improve the generalizability of the existing deep learning models by use of a separate test data. And lastly to improve accuracy of existing local models through use of transfer learning.

4.2 Results of the study

The research study developed a deep learning residual network using the ResNet 50 architecture. Twenty epochs were utilized with the best model achieving an accuracy of 95.93%. With the test dataset, the quantized model was able to predict early blight class and health class with an accuracy of 93.75%.

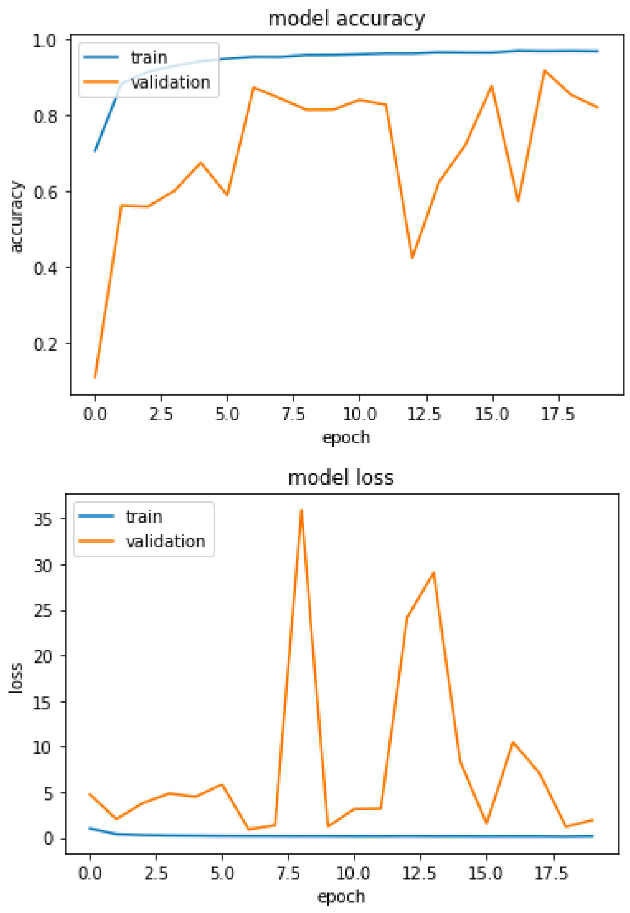


Figure 4.1 Model Evaluation

Model check point was used to save the best TensorFlow model with a size of 245MB. Float16 post training quantization was used to reduce the model size while converting it to a TensorFlow lite model. The resulting model had a size of 61 MB. The TensorFlow model was tested on an unseen data and the most appropriate class was predicted with an accuracy of above 0.9.

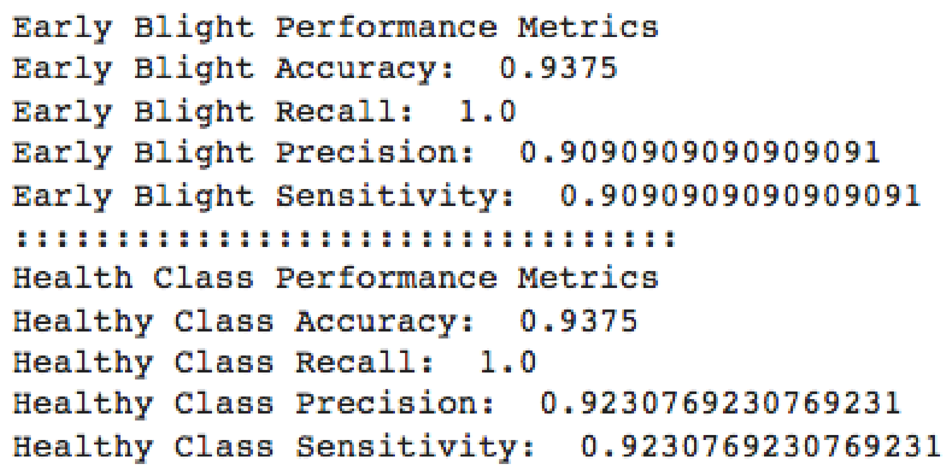
Model	**Size**
Non-Optimized Tensor Flow Model	245 MB
Non-Optimized Tensor Flow Lite Model	122 MB
Optimized Tensor Flow Lite Model	61 MB

Table 4.1 Model Size

The resulting model was evaluated using the loss and accuracy performance metrics. The resulting model has an accuracy of 95.93%. The optimized model was evaluated using a test data with 16 images. Since the problem was a multiclass, each label had its own performance metrics.

Class	TP	FP	FN	TN
Early Blight	5	1	0	10
Healthy	3	1	0	12
Yellow Curl Virus	6	0	0	10

Table 4.2 Confusion Matrix Summary



```
Early Blight Performance Metrics
Early Blight Accuracy:  0.9375
Early Blight Recall:  1.0
Early Blight Precision:  0.9090909090909091
Early Blight Sensitivity:  0.9090909090909091
::::::::::::::::::::::::::::::::
Health Class Performance Metrics
Healthy Class Accuracy:  0.9375
Healthy Class Recall:  1.0
Healthy Class Precision:  0.9230769230769231
Healthy Class Sensitivity:  0.9230769230769231
```

Figure 4.2 Performance Metrics

Early Blight was used to measure the model performance as this was the images that the research was able to obtain for testing. The model had an accuracy of 93.75% for the Early blight class. The model had a recall of 1.0 and a precision of 0.9091. For the healthy class, the model also had an accuracy of 93.75%. The recall was equal to 1.0 while the precision for the healthy class was 0.923. The optimized model had an accuracy loss of 2.18% which is quite minimal as expected with float quantization.

4.3 Discussion of Results

One of the many pertinent issues where crop and pest disease prediction is concerned is the level of accuracy with which the model can predict the possible disease. The usefulness and reliability of a model is determined by its ability to accurately predict unseen data. This research study has demonstrated successfully the use of deep learning model based on transfer learning for prediction of pest as a more superior approach. The success is demonstrated by the high accuracy achieved by the model developed in this study. With the high accuracy achieved as compared to local models Toroitich (2017), this research concludes that the used of deep learning models coupled with transfer learning greatly improves the accuracy of the models.

Still on the issue of accuracy of the current prediction's models in Kenya, the issue of feature handling is crucial to the resulting accuracy of the final model. Local models' surfer low accuracy due to use of manual extraction Maina (2016). These research shows that the use of deep CNN, which has an automatic feature extractor, results into increased accuracy of the underlying model. Thus, this research conclude that the use of automatic feature extraction as opposed to manual features extractions results into more accurate models for crop disease and pest prediction.

Generalization, a key term in deep learning, refers to the ability of a model to react to new data. In practice, most researchers have used the same datasets both for training and testing of the deep learning models Mohanty et al. (2016). This research has however demonstrated that it's a best practice to always test the resulting model generalization ability using unseen data. This provides more confidence in the reliability of the model for use in production. It eliminated the fear of overfitting where the model performs very well on the training data but reacts poorly when new data is used. As a result, this research concludes that one of key best practices in ensuring generalization of a model is not lost, is the use of new and unseen data for testing.

While there has been an increase interest in use of deep learning models at the user level, one big challenge that impeded this move is the increased need of computational resources and time Brahimi et al. (2017). As a result, it is increasingly difficult to deploy the deep learning models in small devices that can easily be used by the users. To solve this challenge, this research has demonstrated that deep learning models can be optimized to have decreased memory size as well as low hardware requirement features. Using quantization, the research has been able to demonstrate that the deep learning models can be reduced in size by up to four times with minimal loss of accuracy. As a result, this research concludes that optimization of deep learning models should be incorporated when building the models for easy deployment as an application on small devices during the production stage.

4.4 Conclusion

4.4.1 Achievements

This section highlights the research achievements with respect to the main the specific study objectives. The main objective for conduction this research was to develop an optimized prediction model for tomato leaf diseases. This objective was advised by previous research that highlighted the lack of optimization in most mobile applications that make use of deep learning models Brahimi et al. (2017). Locally, there was no mention of optimization in any of the documented deep learning models used for early detection of pests and crop diseases. This research study was able to propose and implement a method that can help in the optimization of deep learning models. The research has demonstrated that quantization can be adopted when developing optimized deep learning models. The research has demonstrated that use of quantization reduces the initial deep learning models' size by over 75% and should be adopted when building the models.

The second objective for the research was to improve the generalization of deep learning through use of separate training and test data sets. This objective was advised by the common practice of using same training and test data sets Mengwei et al. (2019). This research was able to use different data sets albeit lack of a good sample size for the test images. This demonstrates a good best practice for deep learning models and also increases the confidence in usage of the model in local farm settings. As a result, this should be adopted for use during the development of deep learning models for pest and disease identification.

The third objective for the research was advising and demonstrating model enhancement methods that can be adopted to improve accuracy levels for local models. This objective was advised by past research which suffered from low accuracy Maina (2016). The research was successfully able to improve the accuracy of deep learning models through several methods. First, the research demonstrated the use of transfer learning as an approach for improving model accuracy. The resulting model had better accuracies as compared to previous local models that did not use transfer learning. Thus, use of transfer learning should be considered for better performance of the deep learning models.

Considering the challenges experienced in adopting use of deep learning within the farm, the model developed provided an opportunity for adopting the technology for use within the farm. With reduced memory size and computation requirements, the model could easily be deployed in a small device such as a smartphone and can be used within the farm without need for a third party or remote control by an expert. The model also provided a baseline for improving classification models for tomato leaf diseases as well as incorporating other crops such as maize and potatoes. The model also provided a chance for stakeholders to access an easy to use application that could incorporate advise on how to manage the detected disease.

4.4.2 Research Limitations

In achieving the study objectives, the research study was faced with several limitations. The study had limited sample size for the test data set with only 16 images. The limitation was as result of lack of diseased tomato leaf from the farms. In addition, the farmers were not able to identify the type of diseases the crop were ailing from making it to accurately label the test images. To overcome this challenge, the research used test images from online databases that were obtained in a different setting or period from the train images. Despite thorough search from different online databases, the research was only able to find a test set with 16 labeled images. With a small sample size there could be a possibility that the model was too optimistic. In addition, due to limited availability of labelled images only three classes (two diseases and a healthy class) were included in the test data.

Secondly, the research was limited in its experimentation capabilities of the different types of quantization methods. The post-training quantization methods included; dynamic-range quantization, full integer quantization and float 16 quantization. The research was limited to float 16 quantization as the other two methods demanded the use of a representative data. Due to lack of readily available labeled data for tomato leaf diseases, it was not possible for the research to obtain a representative dataset.

4.4.3 Further Work

In view of the study limitation this research proposes three areas for further work. Seeing that the testing of the model for the research was a bit limited respect to the test sample size, the research recommended further testing with locally obtained leave images that are accurately labelled and cover more disease classes. Secondly, the research study recommended experimentation with other quantization methods again pegged on availability of representative dataset. A comparative analysis of the different quantization methods is highly recommended with the aim of adopting the best approach with the most benefits.

Seeing that the model was developed and demonstrated from the server side, a fully functioning model in form of an application is recommended for future work. This will allow for experimentation and use by the farmer in a farm setting.

5. References

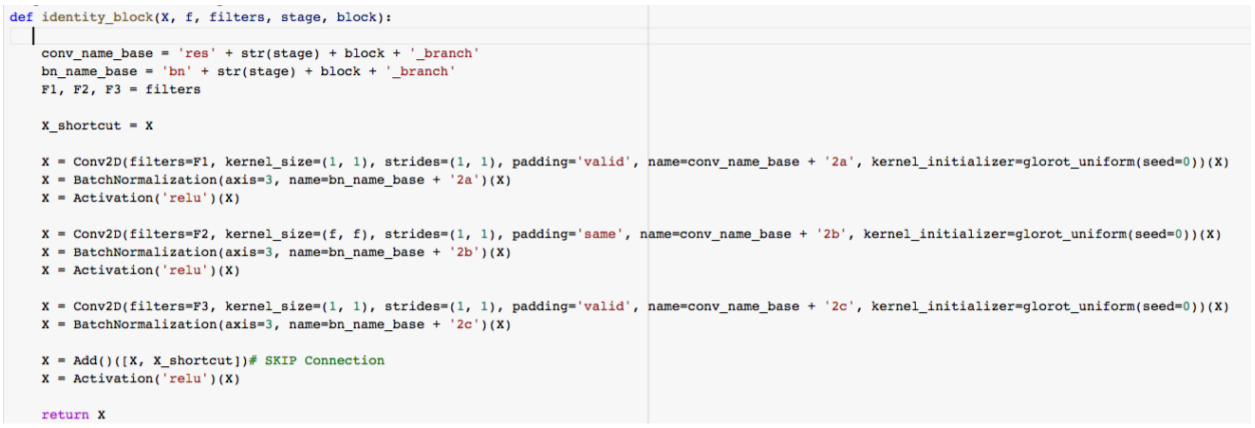
1. OECD, F. a. (2016). *OECD-FAO Agricultural Outlook 2016-2025*. Paris: OECD Publishing.
2. FAO (2017). The future of food and agriculture – Trends and challenges. Rome.
3. FAO (2019). *Integrated Pest Management in tomato in Eritrea - A facilitator's field guide*. Rome
4. MUST (2016). Electronic Device for Real Time Monitoring of Crop Diseases and Pests. *MUST*. https://www.must.ac.ke/electronic-device-real-time-monitoring-crop-disease-pests/
5. Xu, Mengwei & Liu, Jiawei & Liu, Yuanqiang & Lin, Felix & Liu, Yunxin & Liu, Xuanzhe. (2019). A First Look at Deep Learning Apps on Smartphones. WWW '19: The World Wide Web Conference. 2125-2136. 10.1145/3308558.3313591
6. Pallagani, Vishal & Khandelwal, Vedant & Chandra, Bharath & Udutalapally, Venkanna & Das, Debanjan & Mohanty, Saraju. (2019). dCrop: A Deep-Learning Based Framework for Accurate Prediction of Diseases of Crops in Smart Agriculture. 29-33. 10.1109/iSES47678.2019.00020.
7. Brahimi, M., Boukhalfa, K., and Moussaoui, A. (2017). Deep learning for tomato diseases: classification and symptoms visualization. *Appl. Artif. Intell.* 31, 299– 315. doi: 10.1080/08839514.2017.1315516
8. Toroitich, P. K. (2017). *A Model for early detection of potato late blight disease: a case Study in Nakuru County* (Thesis). Strathmore University. Retrieved from http://su-plus.strathmore.edu/handle/11071/5685
9. Maina, C. N. (2016). Vision-based model for maize leaf disease identification: a case study in Nyeri County (Thesis). Strathmore University. Retrieved from http://suplus.strathmore.edu/handle/11071/4820
10. Justin Boulent, S. F.-L.-C. (2019, July 23). Convolutional Neural Networks for the Automatic Identification of Plant Diseases. *Frontiers in Plant Science, 10*(941)
11. Duarte-Carvajalino, Julio & Alzate, Diego & Ramirez, Andrés & Santa, Juan & Fajardo-Rojas, Alexandra & Soto-Suárez, Mauricio. (2018). Evaluating Late Blight Severity in

Potato Crops Using Unmanned Aerial Vehicles and Machine Learning Algorithms. Remote Sensing. 10. 1513. 10.3390/rs10101513.

12. Mohanty SP, Hughes DP and Salathe M (2016) Using Deep Learning for Image-Based Plant Disease Detection. Front. Palnt Sci. 7:1419.
13. Krishnaswamy Rangarajan, Aravind & Raja, P. & Ramesh, Aniirudh. (2018). Tomato crop disease classification using pre-trained deep learning algorithm. Procedia Computer Science. 133. 1040-1047. 10.1016/j.procs.2018.07.070.
14. Kabir, S. M. (2016). *Basic Guidelines for Research: An Introductory Approach for All Disciplines.* Chittagong, Bangladesh: Book Zone Publication
15. Ji, Qingge & Huang, Jie & He, Wenjie & Sun, Yankui. (2019). Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images. Algorithms. 12. 51. 10.3390/a12030051.
16. Anirudh Koul, S. G. (2019). *Practical Deep Learning for Cloud, Mobile, and Edge.* Massachusetts: O'Reilly Media, Inc.

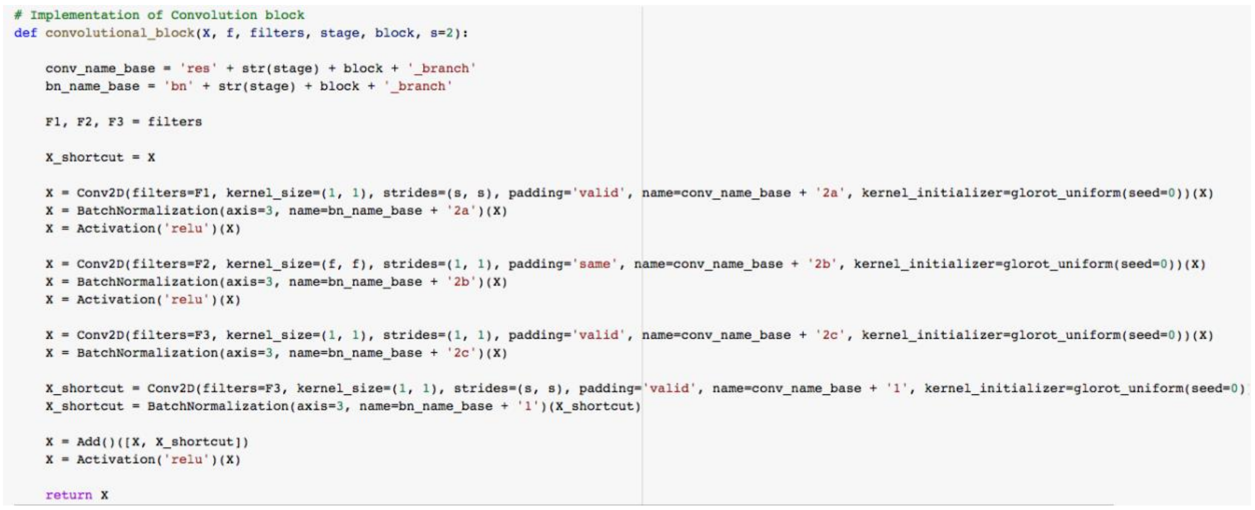
Appendix: Source Code

The Identity Block



```
def identity_block(X, f, filters, stage, block):

    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
    F1, F2, F3 = filters

    X_shortcut = X

    X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(1, 1), padding='valid', name=conv_name_base + '2a', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1), padding='same', name=conv_name_base + '2b', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding='valid', name=conv_name_base + '2c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    X = Add()([X, X_shortcut])# SKIP Connection
    X = Activation('relu')(X)

    return X
```

The Convolution Block



```
# Implementation of Convolution block
def convolutional_block(X, f, filters, stage, block, s=2):

    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    F1, F2, F3 = filters

    X_shortcut = X

    X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(s, s), padding='valid', name=conv_name_base + '2a', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1), padding='same', name=conv_name_base + '2b', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding='valid', name=conv_name_base + '2c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    X_shortcut = Conv2D(filters=F3, kernel_size=(1, 1), strides=(s, s), padding='valid', name=conv_name_base + '1', kernel_initializer=glorot_uniform(seed=0)
    X_shortcut = BatchNormalization(axis=3, name=bn_name_base + '1')(X_shortcut)

    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X
```

The ResNet 50

```
def ResNet50(input_shape=(224, 224, 3)):

    X_input = Input(input_shape)

    X = ZeroPadding2D((3, 3))(X_input)

    X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')


    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    X = X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    X = AveragePooling2D(pool_size=(2, 2), padding='same')(X)

    model = Model(inputs=X_input, outputs=X, name='ResNet50')

    return model
```

Transfer Learning

```
#Transfer Learning
headModel = base_model.output
headModel = Flatten()(headModel)
headModel=Dense(256, activation='relu', name='fc1',kernel_initializer=glorot_uniform(seed=0))(headModel)
headModel=Dense(128, activation='relu', name='fc2',kernel_initializer=glorot_uniform(seed=0))(headModel)
headModel = Dense( 10,activation='softmax', name='fc3',kernel_initializer=glorot_uniform(seed=0))(headModel)
```

Model Quantization

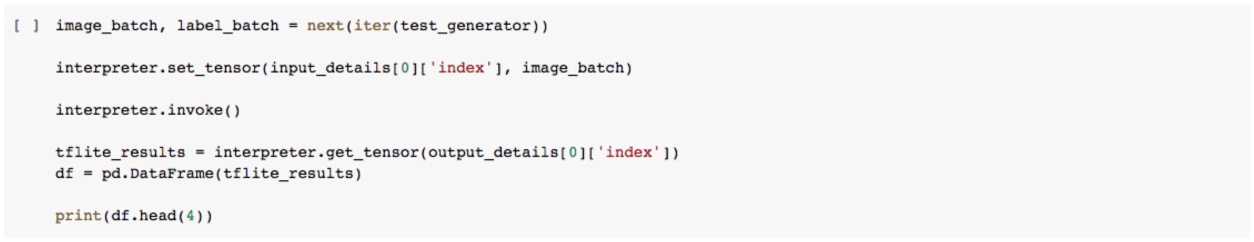
```
[ ]  converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

```
[ ]  converter.target_spec.supported_types = [tf.float16]
```

```
[ ]  tflite_quant_model = converter.convert()
```

Model Prediction

```
[ ]  interpreter = tf.lite.Interpreter (model_content =tflite_quant_model)

     interpreter.allocate_tensors()

     input_details = interpreter.get_input_details()
     output_details = interpreter.get_output_details()

     interpreter.resize_tensor_input(input_details[0]['index'], (16,224,224,3))
     interpreter.resize_tensor_input(output_details[0]['index'],(15,224,224,3))

     interpreter.allocate_tensors()
```



```
[ ] image_batch, label_batch = next(iter(test_generator))

    interpreter.set_tensor(input_details[0]['index'], image_batch)

    interpreter.invoke()

    tflite_results = interpreter.get_tensor(output_details[0]['index'])
    df = pd.DataFrame(tflite_results)

    print(df.head(4))
```

Performance Metrics (Confusion Matrix)

```
print("Early Blight Performance Metrics")
EB_acc = (r0[0][0] + r0[-1][-1]) / numpy.sum(r0)
print("Early Blight Accuracy: ", EB_acc)
#Recall = TP / (TP + FN)
EB_recall = r0[1,1] / (r0[1,1]+r0[1,0])
print("Early Blight Recall: ", EB_recall)
#Precision = TP / (TP +FP)
EB_precision = r0[1,1] / (r0[1,1]+r0[0,1])
print("Early Blight Precision: ", EB_precision)
#Sensitivity = TN / (TN + FP)
EB_sensitivity = r0[1,1]/(r0[1,1]+r0[0,1])
print("Early Blight Sensitivity: ",EB_sensitivity)
print(":::::::::::::::::::::::::::::::::")

print("Health Class Performance Metrics")
H_acc = (r1[0][0] + r1[-1][-1]) / numpy.sum(r1)
print("Healthy Class Accuracy: ", H_acc)
H_recall = r1[1,1] / (r1[1,1]+r1[1,0])
print("Healthy Class Recall: ", H_recall)
H_precision = r1[1,1] / (r1[1,1]+r1[0,1])
print("Healthy Class Precision: ", H_precision)
H_sensitivity = r1[1,1]/(r1[1,1]+r1[0,1])
print("Healthy Class Sensitivity: ",H_sensitivity)
```