THE INFLUENCE OF PARALLEL COMPUTING ON BUILDING DEEP LEARNING MODEL FOR THE CLASSIFICATION OF BEAN DISEASES

NAME: GASHUGI JEAN BOSCO

REGISTRATION NUMBER: 2209000433

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE MASTER OF SCIENCE IN INFORMATION TECHNOLOGY OF THE UNIVERSITY OF KIGALI

October, 2023

DECLARATION

I, **GASHUGI Jean Bosco** do hereby declare that this dissertation entitled "***The Influence of Parallel Computing on Building Deep Learning Model for the Classification of Bean Diseases: Case Study Rwanda Agriculture Board in Rwanda***" is my original work and that it has not been submitted to any other institution of higher learning for any academic award.

…… ……

Gashugi Jean Bosco **Date**

APPROVAL

This dissertation has been submitted to the School of Postgraduate Studies for examination with the approval of my Supervisor.

…… ……

Dr Emmanuel Bugingo **Date**

DEDICATION

I dedicate this work to the Almighty God,

To my parents,

To my family,

To my wife,

To my kids,

To my workmates,

To my friends,

To my sister Jeanne MUKAMUZIMA.

ACKNOWLEDGEMENTS

First and foremost, I thank Almighty God for blessing and protecting me throughout this lengthy journey. This work is the result of several people's combined efforts. I want to offer my heartfelt gratitude to everyone who helped me finish this study project. I would like to convey my heartfelt gratitude to the personnel of the Department of Information Technology at the University of Kigali, as well as all professors, for their efforts to help me learn new skills that I used during this research work. Furthermore, my heartfelt gratitude goes to my supervisor, Dr. Emmanuel BUGINGO, for his dedication and sacrifice of his valuable time and efforts to guide me from the beginning to the finish of this work. More than that, I'd like to show my gratitude to everyone who has helped me in various ways throughout this project, including those whose names are not mentioned. I am grateful to Rwanda Agriculture Board management for granting me permission to do this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS AND ACRONYMS

AI: **A**rtificial **I**ntelligence

ANN: **A**rtificial **N**eural **N**etwork

CNN: **C**onvolutional **N**eural **N**etwork

CPU: **C**entral **P**rocessing **U**nit

CUDA: **C**ompute **U**nified **D**evice **A**rchitecture

cuDNN: **Cu**da **D**eep **N**eural **N**etwork

DL: **D**eep **L**earning

FN: **F**alse **N**egative

FP: **F**alse **P**ositive

GPGPU: **G**eneral-**P**urpose **GPU**

GPU: **G**raphics **P**rocessing **U**nit

ML: **M**achine **L**earning

MLP: **M**ulti-**L**ayer Perceptron

MPI: **M**essage **P**assaging **I**nterface

OPENMPI: **Open M**essage **P**assaging **I**nterface

RAB: **R**wanda **A**griculture **B**oard

RAM: **R**andom **A**ccess **M**emory

ReLU: **R**ectified **L**inear **U**nit

SLURM: **S**imple **L**inux **U**tility for **R**esource **M**anagement

TL: **T**ransfer **L**earning

TN: **T**rue **N**egative

TP: **T**rue **P**ositive

UOK: **U**niversity **O**f **K**igali

VGG: **V**isual **G**eometry **G**roup

ABSTRACT

In recent years, the utilization of deep learning techniques for image classification has made significant strides in the field of agriculture. One of the key areas of interest in agriculture is the early detection and classification of diseases in crops, as this can have an insightful impact on crop revenue and quality. This research has investigated the influence of parallel computing on the performance of a deep learning-based classification model for diagnosing bean diseases. Specifically, we have explored the use of parallel computing frameworks to accelerate model training and inference, thereby enhancing the efficiency and effectiveness of disease classification. Our findings demonstrated the potential for parallel computing to accelerate model training. When training a bean disease classification model, we achieved an accuracy of 0.93 using parallel computing, compared to 0.83 with serial computing. Moreover, parallel computing significantly reduced training time, taking only 3 minutes compared to 51 minutes with serial computing.

Keywords*:* Parallel Computing, Deep Learning, Ray, Central Processing Unit, Graphics Processing Unit, Image classification, and Model evaluation.

CHAPTER ONE: GENERAL INTRODUCTION

1.0 Introduction

This research explored the influence of parallel computing on bean disease classification model building using deep learning. It discusses the background, problem statement, and related work in detail. It then presents the proposed approach for parallel computing-accelerated bean disease classification, including data preprocessing, model architecture selection, parallel computing implementation, and performance evaluation. Finally, the research concluded with a discussion of the potential impact of parallel computing on bean disease detection and future research directions.

1.1 Background of the study

The agricultural industry plays a pivotal role in sustaining global food security and the economy (Kolodziejczak, 2020). As of the latest available statistics, a significant portion of Rwanda's population is engaged in agriculture. According to the National Institute of Statistics of Rwanda (NISR) and various reports from international organizations such as the World Bank and the Food and Agriculture Organization (FAO), revealed that approximately 70-80% of Rwandans rely on agriculture for their livelihood (NISR, 2023). Due to that the Rwandan government has been focusing on modernizing the agricultural sector, improving productivity, and promoting sustainable practices to enhance the livelihoods by sharing agricultural information and advisory services to farmers across the country. Additionally, a quarter of the population owning mobile phones has smartphones, and about 4 times more in urban areas (44%) compared to 12% in rural areas.

The beans are an essential food crop for all Rwandans and many millions of people worldwide, providing a rich source of protein, fiber, and vitamins. However, bean cultivation is challenged by various diseases that can significantly impact yield and quality (Hoang-Tu Vo, 2023). Accurate and timely disease detection is crucial for effective crop management and ensuring food security (Yi, 2015). This can lead to delay treatment and increased crop losses. To address these limitations, novel technologies are needed to improve the efficiency and accuracy of bean disease detection.

Deep learning, a subset of artificial intelligence, has emerged as a powerful tool for image classification tasks, including disease detection in agricultural crops. Deep learning models, particularly convolutional neural networks (CNNs), have demonstrated remarkable performance in analyzing image data and extracting relevant features for classification.

CNNs are a type of neural network that is particularly well-suited for image classification tasks. They are able to learn complex patterns and relationships within images, making them effective for identifying subtle visual differences that may indicate the presence of disease.

However, training deep learning models can be computationally expensive, especially when dealing with large datasets of images. This computational bottleneck can hinder the development and deployment of deep learning models for real-world applications, such as bean disease detection.

Parallel computing offers a viable solution to accelerate the training of deep learning models. By distributing the computational workload across multiple processing units, such as CPUs, GPUs, or specialized accelerators, parallel computing can significantly reduce training time and enable the processing of larger datasets. This enhanced computational efficiency can lead to the development of more accurate and robust disease classification models for bean crops.

1.2 Statement of the problem

Bean farmers face significant economic losses every year due to various diseases that can affect bean plants (Jean B. Ristainoa, 2021). There are two common diseases known as **angular leaf spot** and **bean rust** (Michelle M. Nay, 2018). **Angular leaf spot** is a bacterial disease caused by the Pseudomonas Syringae pv. Lachrymans, is considered as a serious disease of beans in many regions and **bean rust** is a fungal disease mostly attacking the crop in wet weather. If a farmer can detect these diseases early and apply appropriate treatment it can save lot of waste and prevent economic loss (Jean B. Ristainoa, 2021). However, the treatment for angular leaf spot and bean rust are little different so it's important that you accurately identify what kind of disease is there in that bean plant (P. Pamela, 2014).

Furthermore, accurate and timely disease detection is essential for effective crop management and ensuring food security (Seyed Hossein Nazer Kakhki, 2022). Traditional methods of disease identification, such as visual inspection by experts, are often time-consuming, labor-intensive, and prone to human error. However, Deep learning-based disease classification models have emerged as a promising solution. The Deep learning models, particularly convolutional neural networks (CNNs), have demonstrated remarkable performance in image classification tasks, including disease detection in agricultural crops. But, training these models can be computationally expensive, especially when dealing with large datasets of images. This computational bottleneck can hinder the development and deployment of deep learning models for real-world applications.

The Parallel computing offers a viable solution to accelerate the training of deep learning models. By distributing the computational workload across multiple processing units, such as CPUs, GPUs, or specialized accelerators, parallel computing can significantly reduce training

time and enable the processing of larger datasets (Xidong Wu, 2023). This enhanced computational efficiency can lead to the development of more accurate and robust disease classification models for bean crops.

SLURM (**S**imple **L**inux **U**tility for **R**esource **M**anagement) is a cluster management and job scheduling system that can be used to parallelize deep learning training. It allows users to submit jobs to a cluster of machines and specify the resources that each job needs, such as the number of CPUs, GPUs, and memory. SLURM then schedules jobs to run on the available resources in the cluster (Slurm & Deep Learning, n.d.).

In addition, the SLURM can be used to parallelize both data parallelism and model parallelism. To parallelize data parallelism, users can simply submit multiple jobs to train the model on different batches of data. To parallelize model parallelism, users can use a distributed deep learning framework, such as TensorFlow or PyTorch, which automatically distribute the model across the available GPUs or CPUs (Deep Learning on Supercomputers, n.d.).

This research addressed these aspects by utilizing effective parallel computing to build and deploy deep learning-based bean disease classification models that are both accurate and efficient. This model integrated into real-time disease detection systems to aid farmers in identifying and managing bean diseases promptly and effectively.

1.3 Research objectives

The main objective of this research is to contribute to the development of more efficient and accurate deep learning-based bean disease classification models trained on the top of parallel computing that can be effectively deployed in real-world agricultural applications.

1.3.1 General Objectives

The general objective of this study is to investigate the influence of parallel computing on the development and performance of deep learning models for the classification of bean diseases and their deployment in real-world agricultural contexts by using smartphones.

1.3.2 Specific objectives

i. To Distribute computation across multiple nodes using Ray Core and to develop an algorithm suitable for accelerating the model training.
ii. To Demonstrate the performance of the deep learning model with and without parallel computing, focusing on accuracy and training time.
iii. To Develop a mobile application and deploy a deep learning model on mobile devices, so that farmers can use them to identify bean diseases.

1.4 Research questions

The research questions that guided me during this study are:

i. What strategies and algorithms can be developed to distribute computation across multiple nodes using Ray Core, with the aim of accelerating the model training process for bean disease classification?
ii. What is the comparative performance of the deep learning model with parallel computing and the same model without parallel computing in terms of accuracy and training time?
iii. How can a mobile application integrated with a deep learning model be developed and deployed on mobile devices to enable farmers to identify bean diseases?

1.5 Significance of the study

The significance of the study lies in its potential to transform the field of agricultural disease classification by exploiting the power of parallel computing. The outcomes have implications for the efficiency, accuracy, and accessibility of deep learning models, ultimately benefiting farmers, researchers, and the agricultural industry as a whole.

1.5.1 Personal interest

From my perspective, this research provides an interesting exploration of cutting-edge technologies in the fields of deep learning and parallel computing. This research helped me to advance agricultural science and contributed to the solutions that improved global food security and sustainable farming methods.

1.5.2 Institution Interest

The UOK is interested in this study as it aligns with the university's commitment to cutting-edge research and technological advancements. The research contributes to the university's reputation for innovation, and the outcomes have practical applications in agricultural communities and showcasing the university's impact on societal challenges.

1.5.3 Interest to the Researcher

This study holds significant promise for advancing the field of beans disease classification using deep learning and parallel computing. Researchers can utilize its insights to develop more accurate, efficient, and cost-effective models, ultimately contributing to improved agricultural practices and food security.

1.5.4 Interest to the Famers

This research directly benefits farmers by providing to them more accurate and timely tools for disease detection in bean crops. This aligns with the farmers' interests in maximizing productivity, minimizing risks, and promoting sustainable agricultural practices.

1.5.5 Interest in Rwanda Agriculture Board

The Rwanda Agriculture Board (RAB) is interested in this study due to its direct relevance to the agricultural sector in Rwanda. The outcomes of the research could provide valuable insights and tools for the RAB to enhance disease management strategies, improve crop yields, and support the overall sustainability of bean farming practices in the whole country.

1.6 Scope of the study

The scope of this study delved into the various aspects of parallel computing, including hardware architectures, parallel programming models, and parallelization strategies, and assessed their impact on the efficiency, speed, and scalability of deep learning models specifically designed for the classification of bean diseases.

1.6.1 Content scope

The study was included the following:

i. **Deep Learning Techniques:** Examination of various deep learning architectures suitable for bean diseases classification.
ii. **Parallel Computing Techniques:** Exploration of parallel computing paradigms such as distributed computing, and multi-core processing in the context of model training.
iii. **Datasets and Preprocessing:** Analysis of relevant datasets related to bean diseases, and preprocessing methods to prepare data for model training.
iv. **Model Building:** The process of developing and optimizing deep learning models for bean diseases classification, considering both sequential and parallel implementations.
v. **Performance Metrics:** Evaluation of model performance metrics, including accuracy, precision, recall, and F1-score, to measure the effectiveness of parallel computing in comparison to traditional methods.

1.6.2 Geographical

This study was focused on the influence of parallel computing on bean disease classification models, considering various geographical regions where beans are significant crops. Rwanda is a country in East Africa with a significant agricultural sector. In addition, beans are a basic food crop in Rwanda, and bean diseases can cause significant yield losses. This location will be used in this research.

1.6.3 Time scope

This research mainly focused on the development of beans disease classification models using deep learning and parallel computing in the context of Rwanda's current agricultural practices. This research was planned to be carried out within a six-month timeline, encompassing research design, data collection, model development, evaluation, and reporting phases.

1.7 Limitations of the Study

While the study aims to provide valuable insights into the influence of parallel computing on bean disease classification models, there are certain limitations we have faced:

i. **Data Availability**: The development of accurate deep learning models requires large datasets of high-quality images of bean diseases. The limited availability of such datasets may hinder the training and generalization of models for certain diseases or regions.

ii. There are numerous bean plant diseases. However, the dataset used only comprised two classes of bean disease (bean rust and angular leaf spot) and a healthy class, limiting the analysis to these two diseases.

iii. The study didn't cover all possible parallel computing architectures and strategies but focused on prominent and widely used approaches.

1.8 Organization of the Study

Five chapters made up the research. The general introduction to the main topic of study was covered in the first chapter. The second chapter included the literature review in relation to the subject and goals of the study. It especially examined conceptual framework, empirical review, and theoretical review. The research's methodology is presented in the third chapter. The data visualization, image data augmentation, comparison between parallel vs serial computing, and discussion of findings are presented in the fourth chapter. Chapter 5 summarizes the findings, conclusions, recommendations, and future work.

CHAPTER TWO: LITERATURE REVIEW

2.0 Introduction

This chapter focuses on a literature review according to the fundamental theories, different traditional machine learning algorithms as well as deep learning methods employed in plant leaf diseases classification and tools concerned with this research, also it provides definitions and explanations needed to clarify what is going on in this research, conceptual framework and critical analysis of what other researchers have said on the subjects where my project fit and lastly, the contribution of this work that other researches did not address.

2.1. Concept of Study

This study aims to investigate the influence of parallel computing on the development of deep learning-based classification model for bean diseases. Deep learning algorithms, particularly convolutional neural networks (CNNs), have shown promising results in various image classification tasks, including the identification of plant diseases. By harnessing the parallel processing capabilities of modern computing architectures, such as distributed systems, this research seeks to enhance the performance of beans disease classification models in terms of both accuracy and training time.

2.2 Theoretical Review

2.2.1 Machine Learning (ML)

The machine Learning is a branch of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computer systems to improve their performance on a task through experience. In the context of computer vision, machine learning algorithms allow systems to automatically learn and improve their ability to interpret and analyze visual data.

a) Linear Regression

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression. The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s). The sample example for linear regression is shown in Figure 1.

The below formula is a presentation for linear regression.

$$y = \boldsymbol{w1} * \boldsymbol{x1} + \boldsymbol{w2} * \boldsymbol{x2} + bias$$

where **y**: Dependent variables; **w1, w2 :** Coefficient/Slope (or Gradient)

x1, x2 : Independent variables (Features); **bias**: Intercept

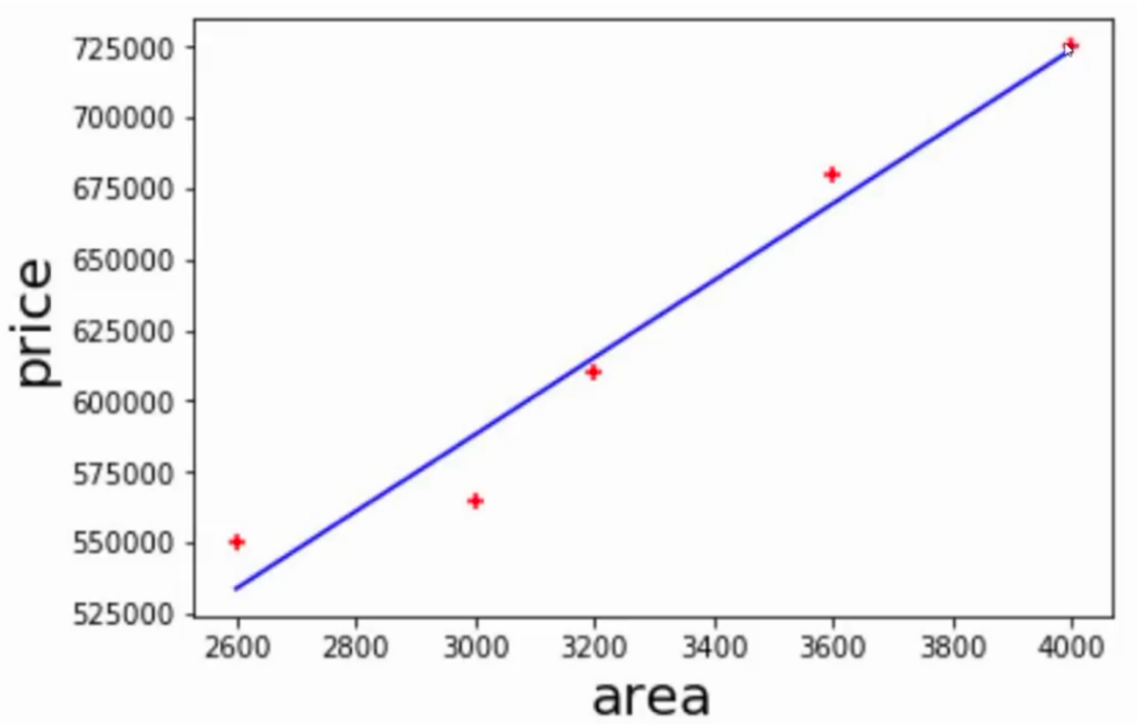


***Figure 1**: Linear Regression*

Source: https://github.com/codebasics/py/tree/master/ML/1_linear_reg

b) Logistic Regression

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class. It is used for classification algorithms its name is logistic regression. it's referred to as regression because it takes the output of the linear regression function as input and uses a **sigmoid/softmax** function to estimate the probability for the given class.

The below formula is a presentation for logistic regression.

$$f(x) = \frac{1}{1 + e^{-x}}$$

f(x): is the predicted output

x : is the input value

e : is the base of natural algorithms (**2.718**)

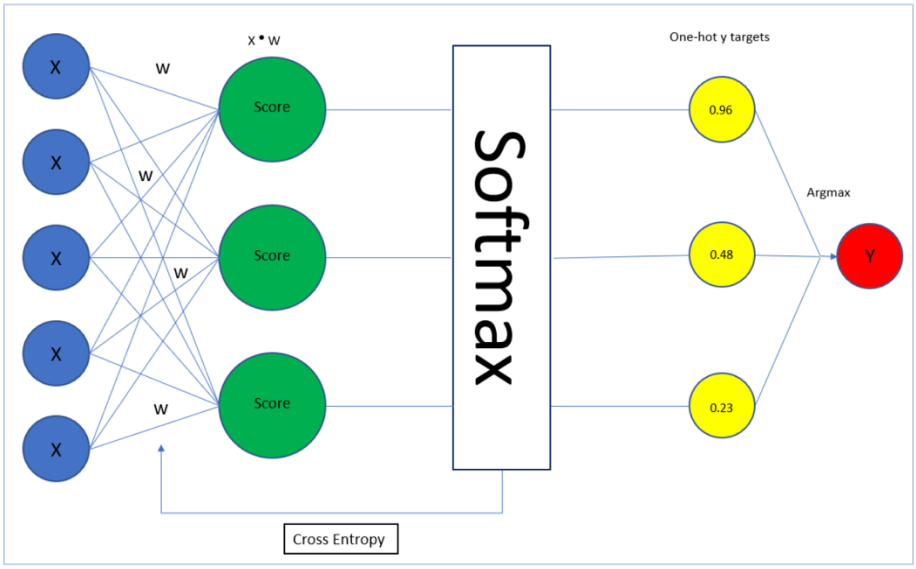


***Figure 2**: Softmax function*

source: https://medium.com/ds3ucsd/multinomial-logistic-regression-in-a-nutshell-53c94b30448f

2.2.2 Deep Learning

Deep learning has revolutionized the field of computer vision, enabling machines to perform complex tasks that were previously considered intractable. At its core, deep learning involves training artificial neural networks (ANNs) to learn from large amounts of data. These ANNs are inspired by the structure and function of the human brain, consisting of interconnected layers of neurons that process and extract information from data.

a) Neural Networks

This research implements deep learning which is simply many layers of hidden units in a neural network. Neural networks also artificial neural networks (ANN) an algorithm inspired by the structure and function of human brain hence has the goal of having machines mimic how the human brain works. Unlike other models, ANN is flexible/ complex functions built by composing simple functions. For this reason, they are fast to train, they do not take too much space, they are simple to manipulate and they can approximate complex non-linear trends in a numerically stable way.

The building blocks of neural networks are called perceptrons. It can be single if the network is made up of only an input and an output layer as in Figure 3. On the other hand, when data is huge having many inputs that a simple perceptron cannot handle, then a third layer is added

called hidden layer and the network becomes a multi-layer perceptron (MLP). An example of MLP with an input, output and two hidden layers is presented in Figure 4. The input layer takes in input data into the network while the output layer gives out the output based on the input data. The hidden layer can be one or many in a network and is responsible for learning the mapping between input and output through a series of matrix multiplications and mathematical transformations.

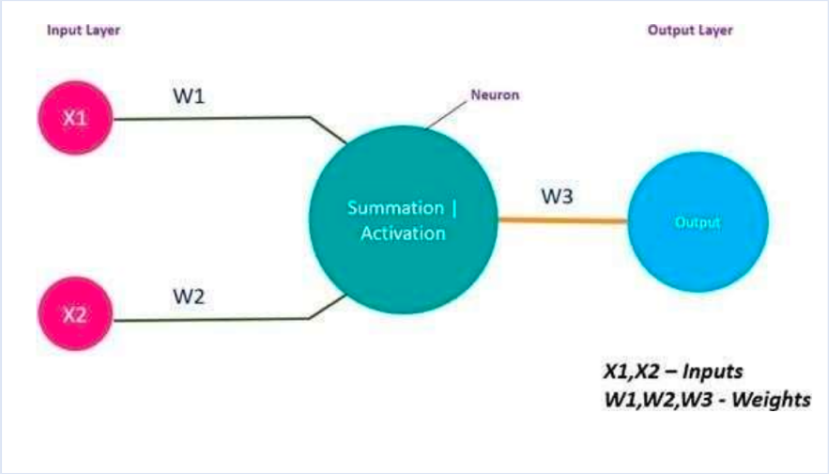


***Figure 3**: Single Perceptron*

Source: https://www.edureka.co/blog/neural-network-tutorial/

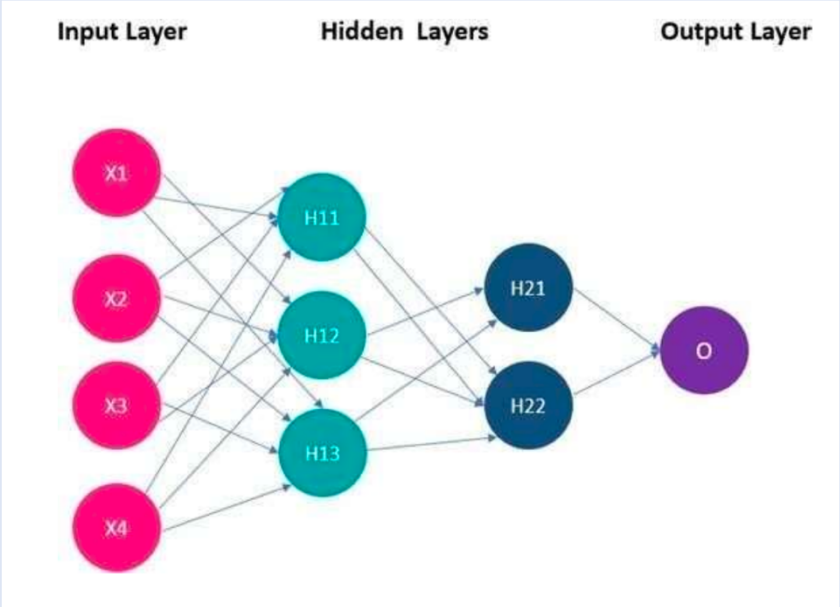


***Figure 4**: Multi-layer perceptron*

Source: https://www.edureka.co/blog/neural-network-tutorial/

i) Mathematical Expression

Above model can be written as:

$$out = f(bias + in_1 * w_1 + in_2 * w_2 + \cdots + in_n * w_n) = f\left(bias + \sum_{i=1}^{n} in_i * w_i\right)$$

where f () is the activation function.

b) Common Activation Functions

i) Linear function

This function is expressed as:

$f(x) = x$

A plot of the linear function is depicted as in Figure 5.

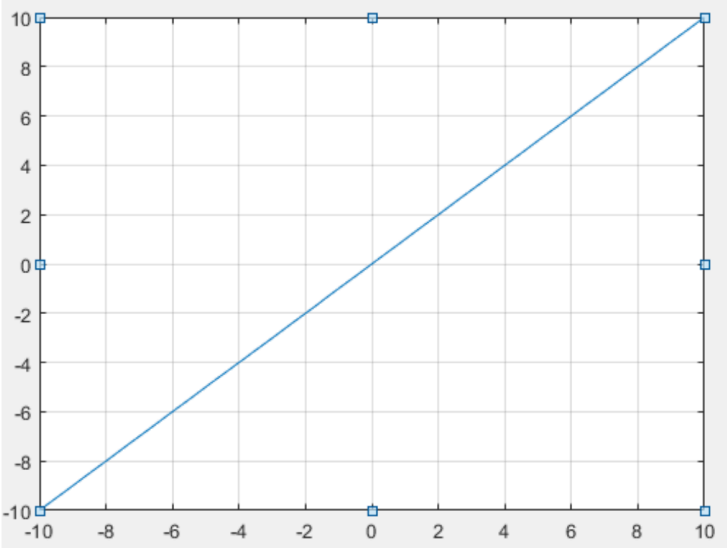


***Figure 5**: The linear function*

Source: http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

ii) Sigmoid function

The sigmoid function is given by:

$$f(x) = \frac{1}{1 + e^{-x}}$$

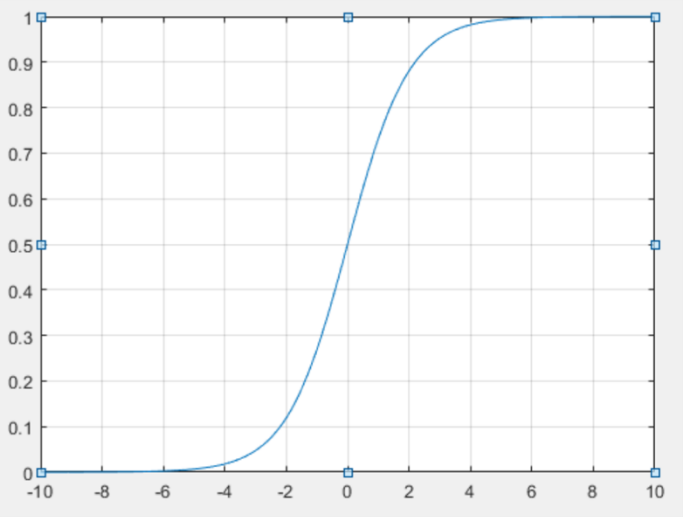


***Figure 6**: The Sigmoid function*

Source: http://www.stat.yale.edu/Courses/1997-98/101/sigfunc.htm

This function could explain the neuron's firing rate, the probability, between 0 and 1.
Tanh function is a rescaled sigmoid function which can be seen from the expression. Their shapes are similar, but with different scales. The Tanh function ranges from -1 to 1. Tanh is zero-centered function.

iii) Rectified linear unit (ReLU)

The rectified linear unit (ReLU) function is given by:

$$f = \begin{cases} 0 \; for \; x \leq 0 \\ x \; for \; x > 0 \end{cases}$$

The plot of this function is described in Figure 7

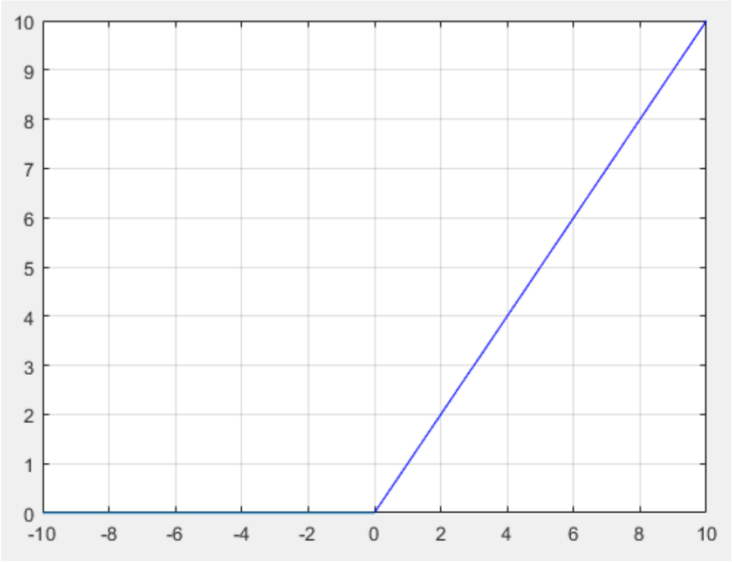


***Figure 7**: The ReLU function*

Source: http://www.stat.yale.edu/Courses/1997-98/101/reclinu.htm

ReLU is widely used in convolutional neural networks. The complexity of the ReLU calculation is smaller than the sigmoid or tanh function, and the ReLu leads to a faster convergence rate, because of without the exponential calculation. But it is still not zero-centered.

c) Mean Squared Error

The Mean Squared Error (**MSE**) is a metric in regression tasks, including the context of training Convolutional Neural Networks (CNNs) for regression problems. It calculates the average of the squares of the differences between the predicted values and the actual values. Mathematically, for a dataset with n samples, predicted values $\hat{y}_i$, and actual values y_i, MSE is computed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

***Figure 8**: Mean Squared Error*

In the context of CNNs, MSE is often used as the loss function during training. The goal of training is to minimize this loss function, which means minimizing the discrepancy between the predicted values and the actual values. During training, the parameters (weights and biases) of the CNN are adjusted iteratively using optimization algorithms (like SGD, Adam, etc.) to minimize the MSE.

d) Update Neurons

When we have a neural network model, we need to train and evaluate this model. In this case, the feedforward propagation and the backward propagation are required to be applied.

e) Forward propagation

In forward propagation, input data is fed into the network at the input layer and goes to the first hidden layer without any operations. The first hidden layer takes in the output of the input layer as its input and applies multiplication, addition and activation operation and passes this value to the next layer. The same procedure repeats for next hidden layers to the final layer where the output value is obtained.

The below formula is a presentation of forward propagation.

$$y = \sum_{i=0}^{n} w^i x^i + b$$

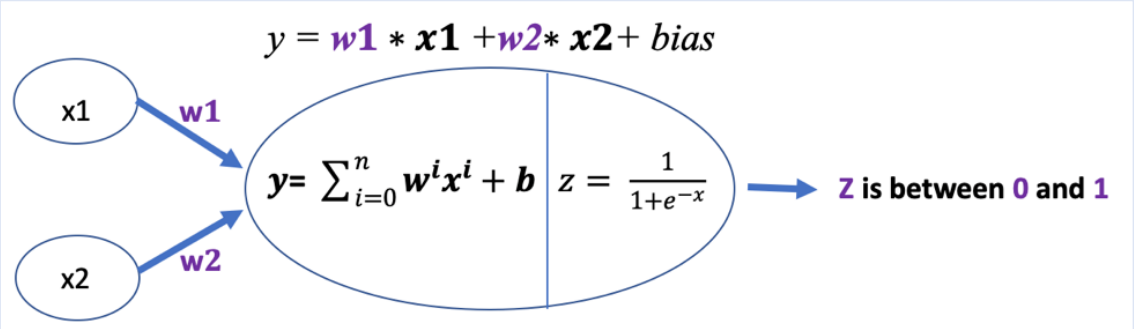


Figure 9: Forward propagation

The main goal is to find the most optimized weight and bias parameters. Since we don't know the optimal values when we start training, we initialize these parameters randomly and perform forward computations from left to right to calculate the output. This output becomes the model's prediction. Then, we need to assess how good this prediction is.

f) Backward propagation

Backward propagation process takes place after getting the predicted value from the forward propagation. The goal of a neural network is to make good predictions with minimal error. To compute the error made during forward propagation, predicted and actual values are compared and a loss function is used to calculate the error value. In back propagation, chain rule is used to compute the derivative/ gradient of error value with respect to weights starting from the last

layer going backwards until all the gradients for each weight in the entire network is obtained. The derivative/ gradient value is then subtracted from the weight value to reduce the error value or loss.

The below formula is a presentation of backward propagation on figure 10.

$$\partial/\partial w1 = \frac{1}{n}\sum_{i=1}^{n} x_i\,(\hat{y}_i - y_i)$$

Figure 10: Cost function

$$W1 = w1 - \text{learning rate} * \partial/\partial w1 \qquad \partial/\partial w1 = \frac{1}{n}\sum_{i=1}^{n} x_i\,(\hat{y}_i - y_i)$$

$$b = b - \text{learning rate} * \partial/\partial b \qquad \partial/\partial b = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)$$

Figure 11: Backward propagation (updating weights and reducing losses)

https://github.com/codebasics/py/blob/master/ML/3_gradient_descent/gradient_descent.py

g) Optimizer algorithms

Optimizer algorithms are essentially tools used to train various machine learning models, especially deep learning models. Their main function is to adjust the internal parameters of the model, like weights and biases, to achieve the best possible outcome.

There are different types of optimizer algorithms, each with its own advantages and disadvantages. Here are some common ones:

- **Gradient descent:** This is a fundamental algorithm that calculates the direction of the steepest descent in the loss function and adjusts the parameters accordingly. It's widely used and relatively simple, but it can get stuck in local minima (not the absolute best solution).
- **Adam (Adaptive Moment Estimation):** This is an advanced algorithm that builds upon gradient descent by incorporating momentum and adaptive learning rates. It can be more efficient than gradient descent and often leads to faster convergence.
- **SGD (Stochastic Gradient Descent)**: This is a popular optimization algorithm used in machine learning, particularly in training large-scale models such as neural networks.

2.2.3 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) is a type of artificial neural network designed for processing and analyzing structured grid data, such as images or video. CNNs are particularly effective in tasks related to computer vision, pattern recognition, and image analysis. The architecture of CNNs is inspired by the visual processing that occurs in the human brain.

a) Convolutional Layer

The most important part in CNN is the convolutional layer. The name CNN is derived from the use of convolution operations. The purpose of convolution is mainly to extract the features of the picture. Convolution operations preserve the spatial relationship between pixels.

Suppose there is a 5x5 image, convolving with a 3x3 filter, and we want to get a 3x3 feature map, as shown below:

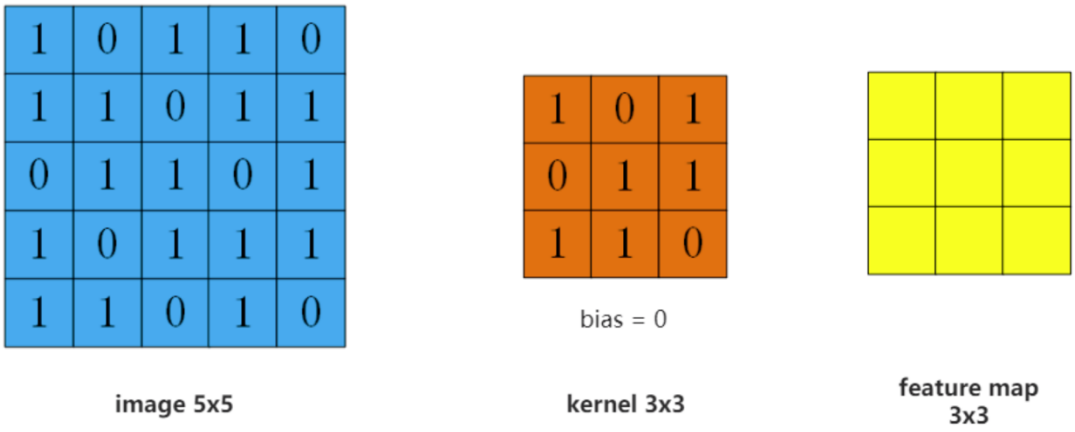


Figure 12: Convolution layer

Source: https://www.sciencedirect.com/topics/mathematics/convolutional-layer

b) Activation Layer

The activation layer mainly performs a nonlinear mapping on the output of the convolutional layer because the calculation of the convolutional layer is still a linear calculation. The activation function increases the ability of generalization. The ReLU function is generally used. Notice that the ReLU function only returns max (0, x), or simply removes the negative weights. The associated layer is also called ReLU layer. Finally, the convolutional layer and the activation function are combined together as a convolutional block.

c) Pooling Layer

When the input passes through the convolutional layer, if the receptive field is relatively small, the length of the stride is relatively small, and the obtained map is still relatively large. The

dimensioning operation can be performed on each feature map through the pooling layer, and the depth of output is still the same, and it is still the number of feature maps.

The pooling layer also has a “pooling” to scan the feature map matrix and calculate the matrix values in the pooling receptive field. There are generally two ways to calculate:

a) Max pooling: take the maximum value in the pooling receptive field matrix.

b) Average pooling: take the average value in the pooling receptive field matrix.

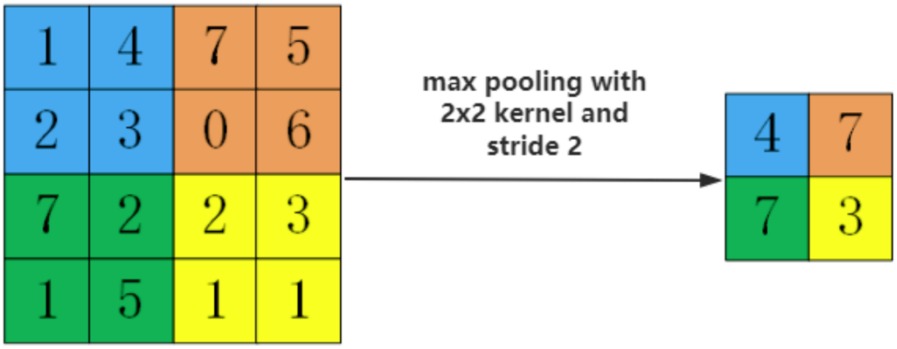


***Figure 13**: Pooling layer*

Source: https://paperswithcode.com/method/max-pooling

d) Fully Connected Layer

The fully connected layer mainly re-fitting features to reduce the loss of feature information. The output layer is mainly prepared for the output of the final target result.

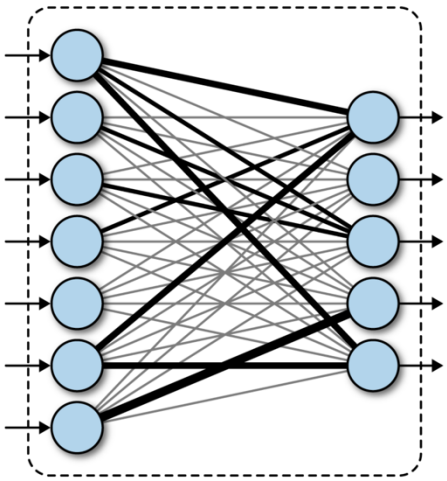


***Figure 14**: Fully Connected layer*

Source: https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

e) Flattening Layer

Before the fully connected layers, the output from the convolutional and pooling layers is typically flattened into a one-dimensional vector. This vector serves as the input to the fully connected layers.

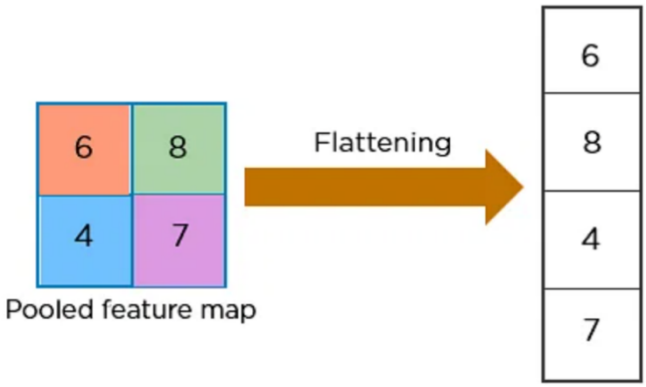


***Figure 15**: Flattening layer*

Source: https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

2.2.4 Image classification with CNN

Image classification using convolutional neural networks (CNNs) is a powerful technique for automatically identifying and categorizing images based on their content. CNNs have become the dominant approach for image classification tasks due to their ability to extract and learn complex patterns from image data.

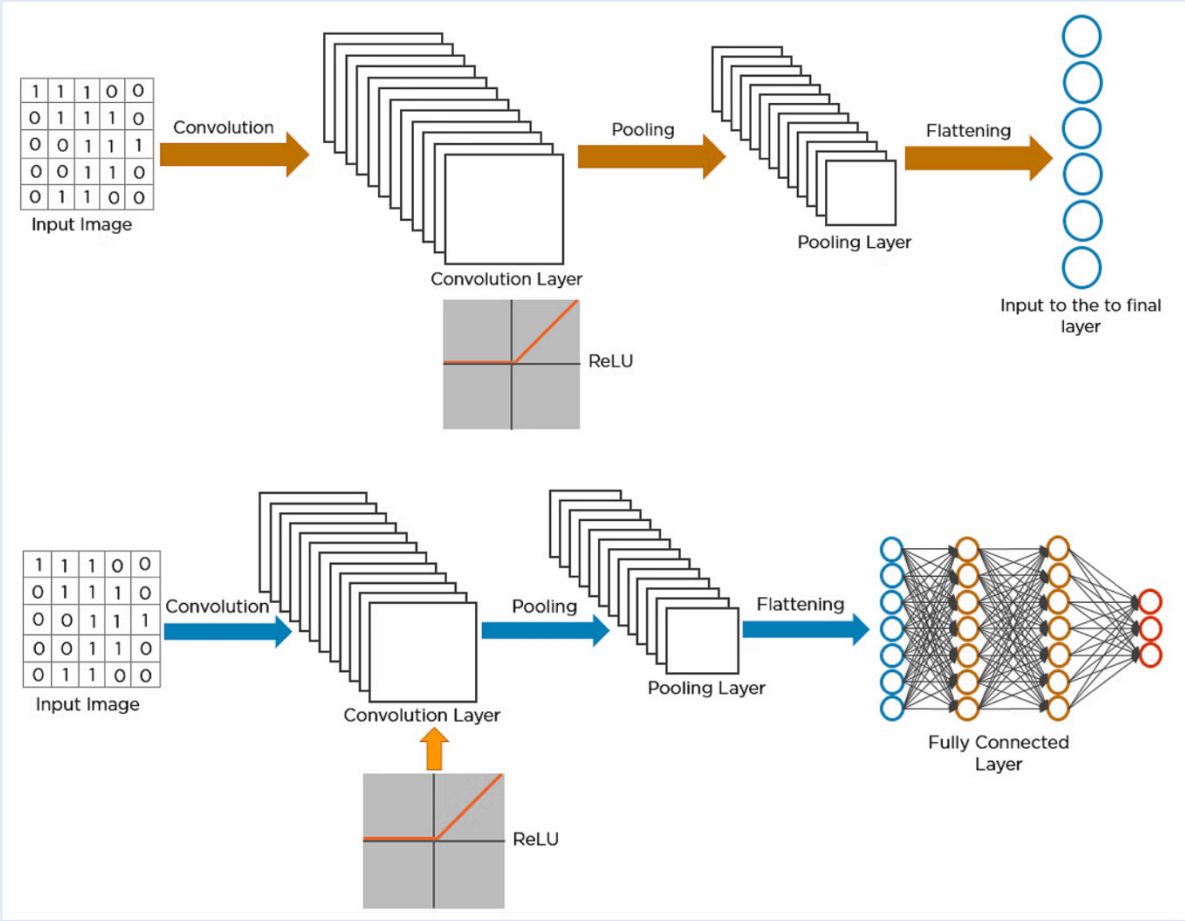


Figure 16: Image classification architecture

Source: https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

How exactly CNN recognizes a plant disease:

a) The pixels from the image are fed to the convolutional layer that performs the convolution operation
b) It results in a convolved map
c) The convolved map is applied to a ReLU function to generate a rectified feature map
d) The image is processed with multiple convolutions and ReLU layers for locating the features
e) Different pooling layers with various filters are used to identify specific parts of the image

f) The pooled feature map is flattened and fed to a fully connected layer to get the final output

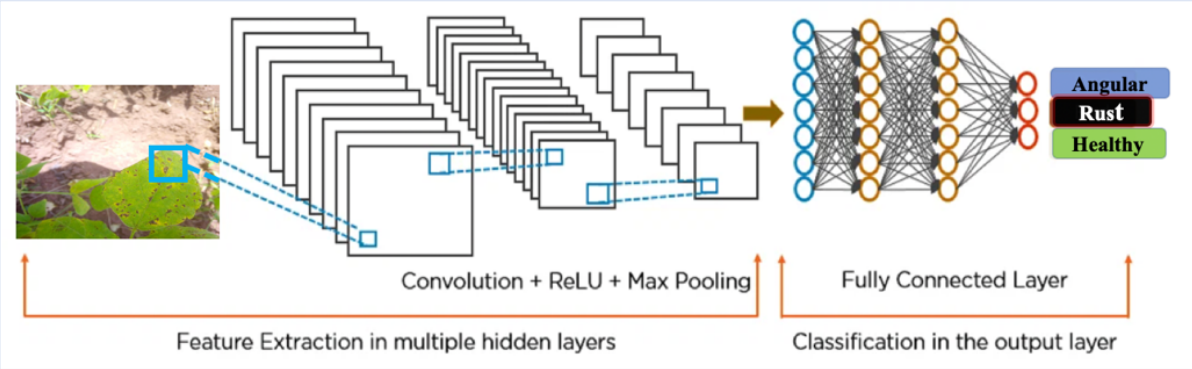


Figure 17: Bean plant disease classification

2.2.5 Parallel Computing

In recent years, parallel computing and soft computing has become a rapidly evolving field of study. The demand for parallel processing in increasing day by day. There are various software tools and libraries by which we can parallelize our programs. For example, we have OPENMP in c++ for parallel computing. OPENMP supports FORTRAN, C and C++. It is basically an Application Programming Interface for shared Memory Model programming. Python has its separate parallel processing module named Multiprocessing. Multiprocessing module enables to spawn multiple processes, allowing programmer to fully leverage the computing power of multiple processors. The main drawback of Python's Multiprocessing module is that it cannot be used for handling large numeric data. It cannot be used in Deep Learning Frameworks such as Keras as it decreases the accuracy of the models. Shared variables cannot be used in the Multiprocessing Module. Python also has a Parallel and Distributed computing framework called Ray. Ray can be used for developing emerging AI applications such as image classification, face recognition etc. Parallelizing multiple cores of CPU using Ray can also increase the speedup of the model significantly. The benchmark image classification algorithm used in this project is Convolutional Neural Network. The Dataset planned to be used in this project is the Beans Disease Image dataset contains around 2000 images. The system is configured with 250 GB RAM with 72 CPU Cores and Tesla P100 GPU for a single node.

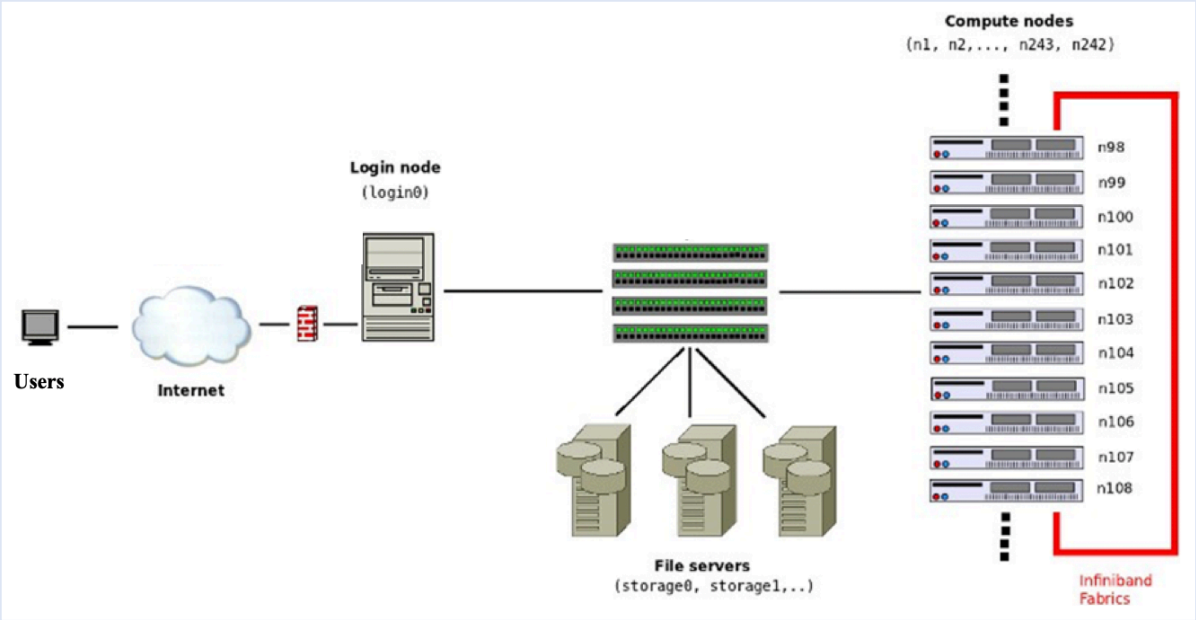


***Figure 18**: Cluster system diagram*

Source: https://scicomp.ethz.ch/wiki/Parallel_job_submission_with_SLURM

a) Message Passing Interface (MPI)

MPI, The Message Passing Interface, is a standard API for communicating data via messages between distributed processes that is commonly used in HPC to build applications that can scale to multi-node computer clusters. As such, MPI is fully compatible with CUDA, which is designed for parallel computing on a single computer or node.

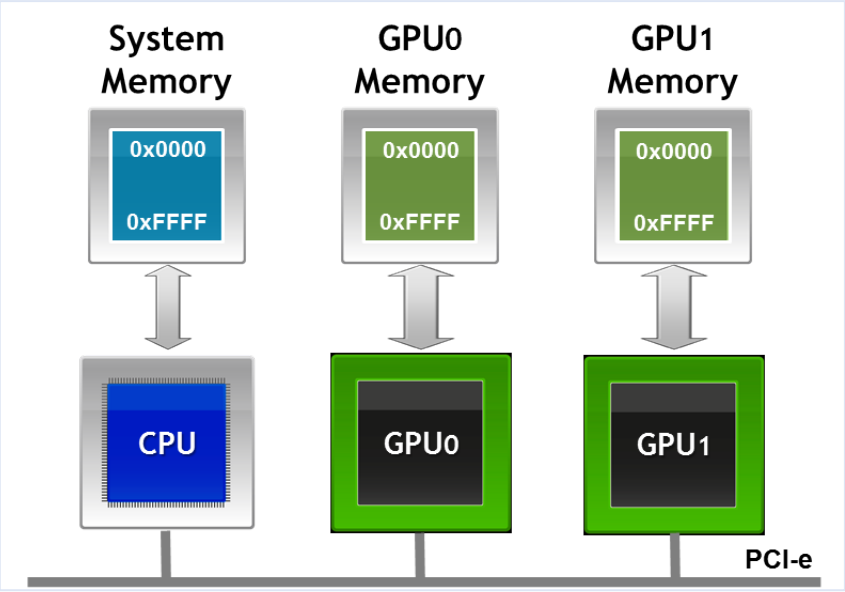


***Figure 19**: MPI*

Source: https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/

b) OpenMPI (Message Passing Interface)

OpenMPI (Message Passing Interface) is an open-source standard for parallel programming in distributed computing environments. It is designed to facilitate communication and coordination among processes running on different nodes of a parallel system, such as a cluster or a supercomputer. OpenMPI allows developers to create parallel applications that can efficiently utilize the computational power of multiple processors or cores.

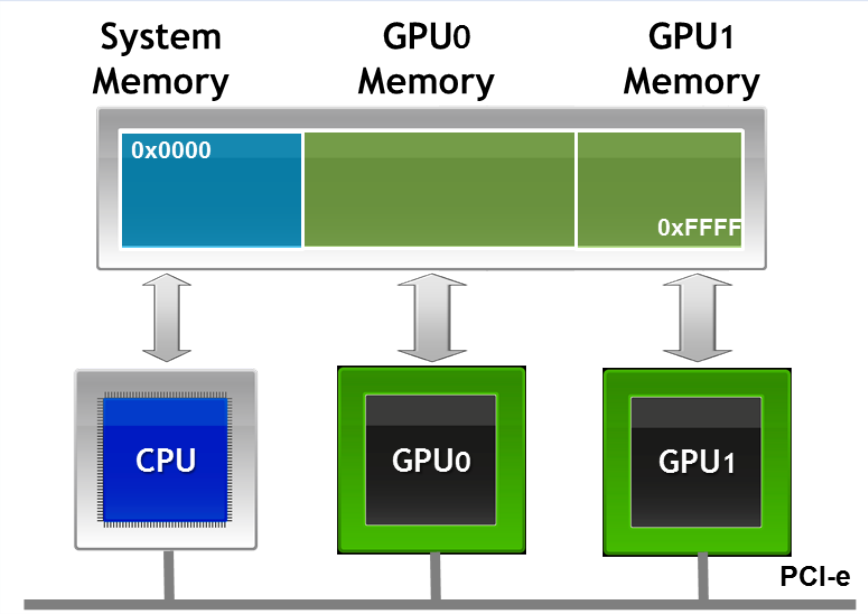


Figure 20: OpenMPI

Source: https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/

c) SLURM

SLURM, which stands for **S**imple **L**inux **U**tility for **R**esource **M**anagement, is an open-source job scheduler and resource management system commonly used in high-performance computing (HPC) environments. SLURM is designed to efficiently manage and allocate computing resources, such as CPU cores and GPUs, in a cluster or supercomputing environment. It provides a framework for users to submit, monitor, and control jobs, making it particularly useful for parallel and distributed computing tasks, including deep learning neural network training.

In the context of deep learning neural network training, SLURM can be used to efficiently distribute and manage the computational workload across multiple nodes of a cluster. Finally, Deep learning frameworks, such as TensorFlow or PyTorch, can be configured to distribute the training workload across multiple nodes and GPUs. SLURM facilitates this by providing the necessary infrastructure for parallel execution.

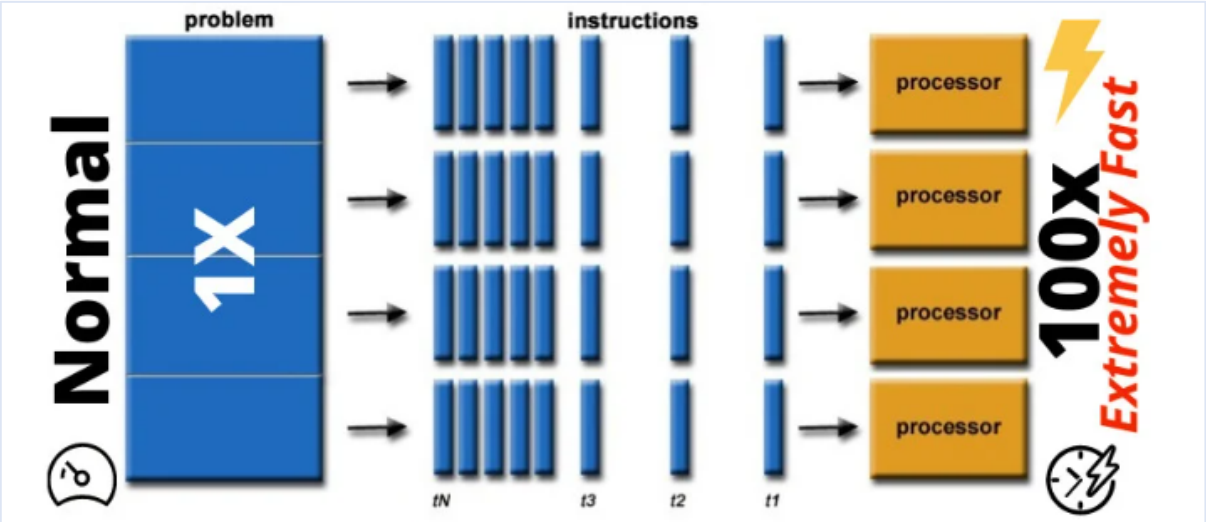


***Figure 21**: Training Data distribution with SLURM*

Source: https://researchcomputing.princeton.edu/support/knowledge-base/parallel-code

2.3 Empirical Review

The use of deep learning for bean diseases classification has been an area of active research in recent years. Several studies have investigated the application of deep learning for beans disease classification. These studies have demonstrated the potential of deep learning to accurately identify beans diseases from images of infected plants. Moreover, Transfer learning (TL) technique have been used to improve the learning process by leveraging a pretrained model for a new task. As the TL models are neural networks that have undergone training on a large dataset, typically used for tasks such as image recognition or natural language processing. These models are then adjusted and fine-tuned for a new task using a smaller dataset. The core principle behind transfer learning is based on the idea that a model capable of recognizing specific features in one domain can be repurposed to identify similar features in another domain (Yang, 2010). However, deep learning models can be computationally expensive to train, especially when dealing with large datasets.

Parallel computing has emerged as a promising solution to address the computational challenges of training deep learning models. By utilizing multiple processors or computers, parallel computing can significantly reduce the training time and enable the development of more complex and accurate deep learning models.

For instance, a study by Geng et alham. (Geng, 2020) employed parallel computing to train a deep learning model for beans disease classification. The authors reported that using parallel computing reduced the training time by up to 80% compared to using a single processor.

Another study by Wu et al investigated the use of parallel computing to train a deep learning model for classifying multiple plant diseases, including beans diseases. The authors found that parallel computing significantly improved the accuracy of the deep learning model (Wu, 2021). These studies highlight the potential of parallel computing to enhance the development and performance of deep learning models for beans disease classification. For example, study by Zheng (Zhang, 2018) achieved an accuracy of 98.2% on a dataset of bean images using a convolutional neural network (CNN). Besides, work by Cheng (Cheng, 2019) achieved an accuracy of 99.5% on a dataset of bean images using a deep residual network (DRN). Parallel computing has also been used to accelerate the training of deep learning models for bean diseases classification. Likewise, they used a GPU to accelerate the training of a CNN for bean diseases classification, achieving a 10x speedup (Wang, 2019). This paper states that the CUDA supported GPU gives much faster and better results as compared to the CPU. The paper compares the GPU and CPU computations of multiple algorithms by parallelizing them into independent threads. Moreover, this paper gives us a set of guidelines to help us determine when and in which way should the GPU be used for the parallelization (Prathamesh Borhade, 2020).

Generally, the use of computers to solve complex scientific problems and simulations led to high-performance computing systems to provide more computing power through distributed computing. To achieve this, large problems are broken down into smaller problems, and each is computed in a separate computing unit. Information is exchanged between the compute units through a network and specific software such as Message Passing Interface (MPI) (Kahira, 2021).

This study tested the model on three different bean leaf image datasets with varying difficulty and found that the proposed approach achieved remarkable accuracy, with over 92% accuracy on all three datasets. This demonstrates the potential of deep learning techniques in detecting bean leaf diseases and highlights the value of using mobile deep learning models for this purpose (Elhoucine Elfatimi, 2023).

Dispute, this research implemented a data augmentation approach to train a robust deep-learning model, to accurately classify a given bean leaf image (unseen) into different infected or healthy category. In addition, this study addressed the effectiveness of the classifier in challenging real-life situations such as angle rotation and light alteration of a leaf. Lastly, the study introduces a novel approach to accelerate the training process of deep learning models for bean disease classification by leveraging parallel computing. By distributing computational

tasks across multiple processors, in result to that the model training time is significantly reduced, allowing for quicker experimentation and optimization.

2.4 Conceptual Framework

This conceptual framework provides a structured approach to studying the influence of parallel computing on bean disease classification model building using deep learning. The framework emphasizes the importance of data collection, data cleaning & preprocessing, data augmentation, data training with parallel computing, test model, model evaluation, model quantization, and usability of the model in real-world applications.

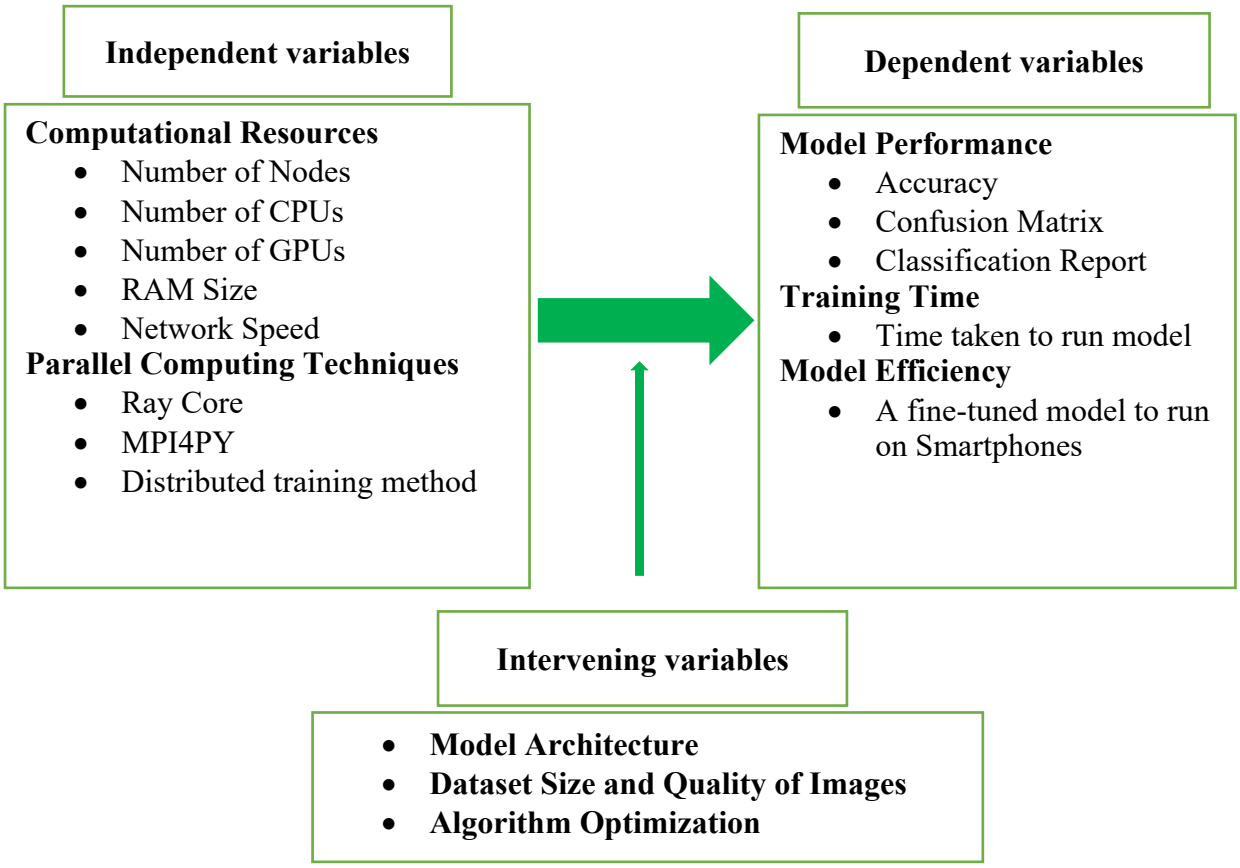


***Figure 22**: Conceptual Framework*

Source: Own Conceptualization

2.5 Block Diagram

a) **Parallel Computing**

b) **Serial Computing**

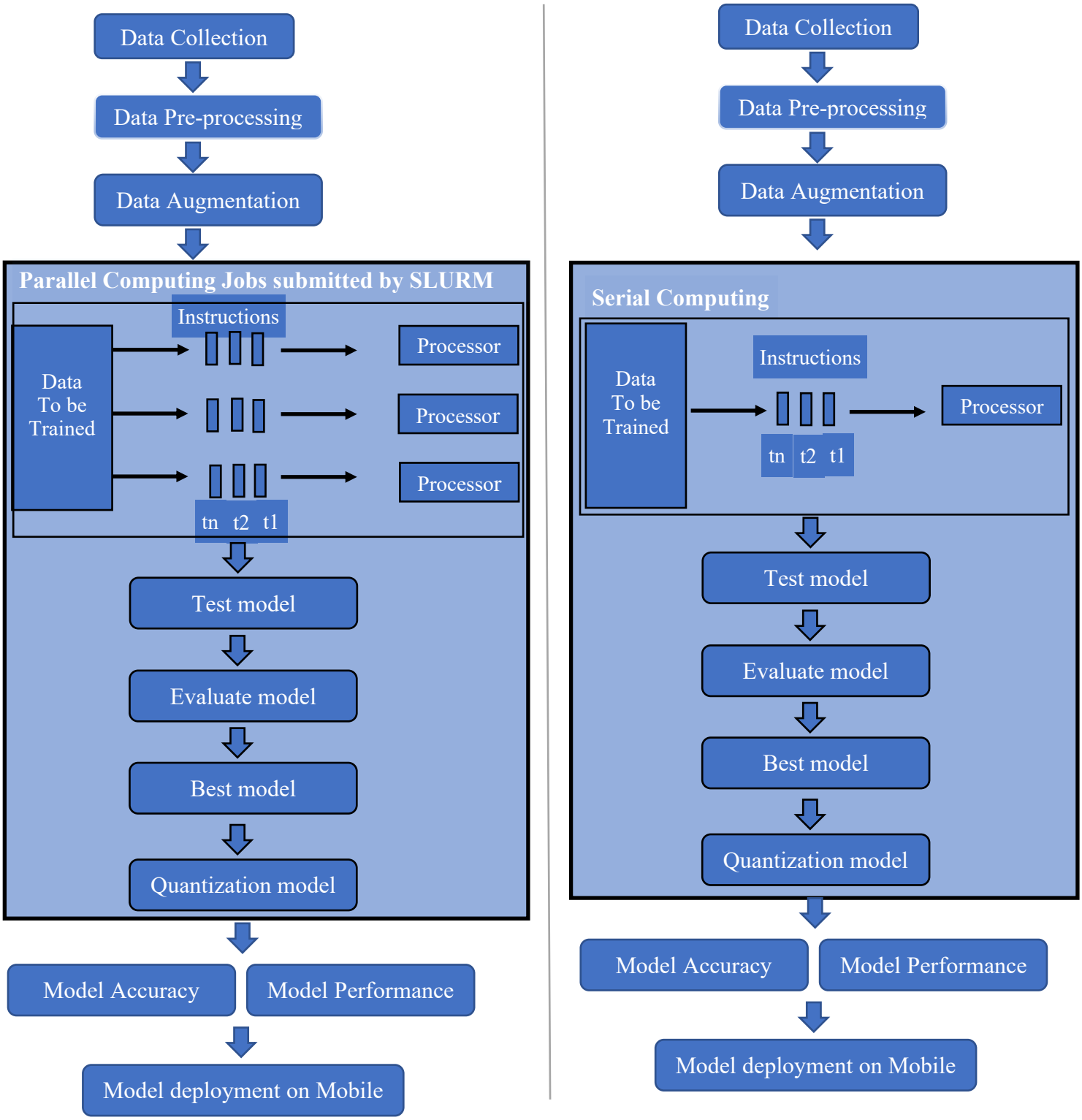


***Figure 23**: Parallel/Serial Computing Block Diagram*

2.6 Research Gaps

Despite the growing interest in utilizing deep learning for plant disease classification, the limited exploration of Parallel Computing still exists. Even though deep learning has shown promise in disease classification, there's a lack of research investigating the specific benefits of parallel computing techniques in this domain. This study aims to bridge this gap by exploring the influence of parallelism on the accuracy and training time for the bean disease classification model (Haohan Ding, 2023).

CHAPTER THREE: RESEARCH METHODOLOGY

3.0 Introduction

This chapter describes procedures and strategies employed in the implementation of this study. It covered the Research Design, Study Population, Sampling, Data Collection Methods, Data Processing, Data Analysis, Model Evaluation, System Architecture, Preparation of Deep Learning Environment and Tools, Limitations, and Ethical considerations. This research approach was suitable and fit for our study because the researcher collected data based on our target objectives and criteria.

3.1 Research Design

For this study, due to stratified sampling is a technique in which the population is divided into smaller, more homogeneous subgroups called strata. Furthermore, it ensures that the sample is representative of the entire population, and it is especially important when the population is not homogeneous. It is in this regard that we used this research method to collect the target population which corresponds to the research questions we intended to address. Therefore, we developed a model to be used by farmers and others to detect diseases in bean plants in real-time.

3.2 Study Population

The study population for this research was built on a dataset obtained from **AI-Lab Makerere** (AI-Lab-Makerere, 2020). The images were captured in bean fields by the Makerere AI lab in collaboration with the National Crops Resources Research Institute (NaCRRI), Uganda's national body in charge of agricultural research. The dataset presented a diverse range of bean cultivars and disease conditions with high-resolution and processed images. Moreover, the bean leaf images were broadly divided into two categories, the dataset has **990** images to produce a good model.

a) Healthy bean plants: These plants do not exhibit any signs of disease. Their leaves are typically green and free of spots, lesions, or other abnormalities.

***Figure 24**: Healthy bean plant*

Source: https://www.thespruce.com/how-to-grow-green-beans-1403459

b) Bean plants with diseases: These plants exhibit various symptoms of disease, depending on the specific pathogen involved. Some common symptoms include:

i) Angular leaf spot: This disease is caused by the bacterium Pseudomonas syringae pv. phaseolicola. It causes small, angular lesions on the leaves, which can eventually coalesce and cause the leaves to drop off.

***Figure 25**: Angular leaf spot*

Source: https://barmac.com.au/problem/angular-leaf-spot-2/

ii) Bean rust: This disease is caused by the fungus Uromyces appendiculatus. It causes small, orange-brown pustules on the underside of the leaves. The pustules can eventually rupture and release spores, which can infect other plants.

***Figure 26**: Bean plant with rust disease*

Source: https://plantwiseplusknowledgebank.org/doi/10.1079/PWKB.20127802224

3.3 Sampling

The stratified sampling approach was employed to ensure a balanced representation of different disease classes in the dataset (Stratified Random Sampling, n.d.). The sampling strategy aimed to capture the natural variability in disease symptoms and leaf characteristics. This technique improves the accuracy of our deep-learning model.

This study employed the stratified formula as follows:

Stratified sampling Formula: $\mathbf{n\_i} = \frac{(N\_i * n)}{N}$

where:

$\mathbf{n\_i}$ is the sample size for stratum **i**;

$\mathbf{N\_i}$ is the population size for stratum **i**;

n is the total sample size;

N is the total population size.

There were 3 folders inside the dataset. One for the train set containing **792** images **(80%)** which were used to fit the models, another for the validation set containing **99** images **(10%)** used to estimate prediction error for model selection, and the last one for the test containing **99** images **(10%)** to be used for assessment of the generalization error of the final chosen model. All the sets contained the three categories healthy leaf images, leaf images affected by angular leaf spot, and leaf images affected by bean rust as shown in Table 1. The dataset was balanced hence a model built on it has minimal chances of being biased towards a specific class.

Population	Categories	Samples		
		Train	Validation	Test
990	Healthy	264	33	33
	Angular Leaf Spot	264	33	33
	Bean Rust	264	33	33
		792	99	99

Table 1: *The summary statistics of the 3 classes in train, validation, and test sets*

The size for test and validation sets $\mathbf{n\_i} = \frac{(N\_i * n)}{N}$

Where $\mathbf{N\_i}$:330; **n**: 99; **N**:990

Thus $\mathbf{n\_i} = \frac{(330 * 99)}{990} = \mathbf{33}$

Finally, the test is equal to **(33)** and validation is equal to **(33)** for each class respectively.

3.4 Data Collection Methods and Instruments/Tools

While conducting Data collection for this research, We have chosen to use a dataset provided by **AI-Lab Makerere**, due to the leaf images had high resolution and were annotated, standardized size, format, and color balance, which enhanced the quality and consistency of the dataset. Additionally, these leaf images are totally similar with the local bean leaf here in Rwanda.

3.5 Data Processing

The data processing included image augmentation, image cleaning, and other preprocessing steps to ensure the model's robustness. Parallel processing techniques were applied to distribute the computational workload across multiple processors or GPUs, significantly reducing training time.

3.6 Data Analysis

To enhance deep learning model performance in distinguishing between healthy and diseased bean plants, robust datasets comprising images of both conditions are essential. Implementing data preprocessing techniques, including image resizing, normalization, and augmentation, is crucial to refine the data quality and augment model accuracy. Furthermore, leveraging parallel computing facilitates swift and efficient execution of tasks like parallel image resizing and data augmentation across multiple processors, thereby expediting the preprocessing phase and overall model development process.

3.7 Model Evaluation

Model evaluation is a very crucial part of building an effective deep learning model since it gives an explanation on the model performance. To evaluate a model, evaluation metrics are employed to discriminate among the model results. This research adopted 3 model evaluation metrics that are classification accuracy, confusion matrix and classification report.

3.7.1 Classification Accuracy

Accuracy measures how close measurements are to the specific value. This metric works by comparing the predictions of a classification model with the known labels. It is then calculated as the number of correct predictions divided by the total number of predictions.

It is given by:

Accuracy = Correct Predictions / Total Predictions

This research employed these metrics to understand the overall proportion of correct predictions to total predictions for the trained model.

3.7.2 Confusion Matrix

Evaluation of a classification model should not depend on classification accuracy alone since this metrics can be misleading in these cases:

a) When the number of classes in a dataset exceeds two like in this study we have three classes and if we get a high accuracy, we would not know whether it is because all the three classes are being predicted equally well or if only one or two classes are being neglected by the classifier.

b) Though not the case in this study, when the number of observations in each class is unequal because the classifier can be highly biased towards the majority class. This research thereby

calculated the classification matrix to give a better understanding of what the classifier was getting right as well as what types of errors it was making during classification. A confusion matrix refers to a table used to summarize by each class, the number of correct and incorrect predictions of a classifier with count values.

		Actual Values		
		Positive	Negative	Total
Predicted Values	Positive	*TP*	*FP*	*TP + FN*
	Negative	*FN*	*TN*	*FN + TN*
	Total	*TP + FN*	*FP + TN*	*N*

***Table 2**: Confusion matrix for binary classification*

From Table 2, true positive (TP) are the actual positives that the classifier correctly predicted as positive. True negative (TN) is the actual negative that the classifier correctly predicted as negative. TP and TN are both termed as correct predictions for the positive and the negative classes respectively. False positive (FP) are the actual negative but the classifier wrongly predicted it as positive. We also call this type I error. Lastly, false negative (FN) is the actual positive but the classier is wrongly predicted as negative. This is also known as a type II error. FP and FN are both called the incorrect/ misclassified predictions for the positive and the negative classes respectively.

3.7.3 Classification Report

Like confusion matrix, this classification report also uses true positives, false positives, true negatives and false negatives to measure how many predictions are true and how many are false. It measures the quality of predictions from a classifier. First from the classification report, this study used precision = TP / (TP + FP) to determine the reliability of the model. It tells the proportion of the correctly predicted cases that actually turned out to be positive. Secondly, this study considered recall = TP / (TP + FN) which tells the proportion of the positive cases that were successfully predicted by the classifier. Lastly, we considered F1 Score = 2*(Recall * Precision) / (Recall + Precision) which is the weighted harmonic mean of precision and recall. F1 score value of 1.0 indicates best score while a value of 0.0 signifies worst performance.

3.8 System Architecture

The hardware configuration used while conducting this research is a CentOS 7 as an operating system, The Graphics processing unit (GPU) with NVIDIA Tesla p100-pcie-16GB model and 250 GB Random Access Memory (RAM) for each. The software used is python programming language which is an interpreted high-level and general-purpose programming language for all data manipulation and analysis. Notebook and Terminal were used as the computational environment and TensorFlow as the deep learning framework.

3.9 Preparation of Deep Learning Environment

3.9.1 Operating System

CentOS 7 is a popular operating system for deep learning, due to its stability, flexibility, and wide range of supported deep learning frameworks and libraries. Moreover, it is a mature and stable operating system that is well-suited for production environments. It is also relatively lightweight and can be installed on a variety of hardware platforms, including desktops, laptops, and servers. That's the reason, we chose to use it in this study.

3.9.2 NVIDIA Support

NVIDIA provides a comprehensive suite of hardware and software for deep learning. Their GPUs are the most powerful accelerators available for deep learning, and their CUDA and cuDNN libraries provide highly optimized primitives for deep learning operations. GPUs are specialized processors that are designed to handle the massive parallel computations that are required for deep learning. NVIDIA's GPUs are the most powerful GPUs available, and they are used by many of the world's leading deep-learning researchers and practitioners (NVIDIA Tesla P100 PCIe 16 GB, n.d.).

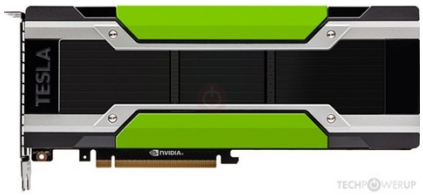


Figure 27: GPU

a) CUDA

CUDA is a parallel computing platform and application programming interface (API) model developed by NVIDIA for general-purpose GPU (GPGPU) computing. CUDA is a software layer that gives developers direct access to the GPU's hardware, allowing them to write code that can be run on the GPU in parallel (What Is CUDA?, n.d.).

b) CUDNN

cuDNN is a GPU-accelerated library of primitives for deep neural networks (DNNs). cuDNN provides highly optimized implementations for standard DNN routines such as forward and backward convolution, attention, matmul, pooling, and normalization (NVIDIA cuDNN, n.d.).

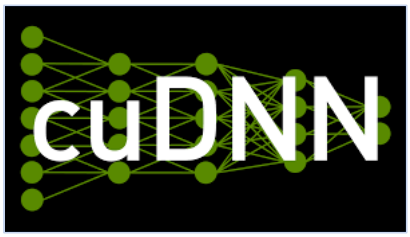


Figure 28: CuDNN

3.9.3 Tools

There are so many deep-learning tools. Because deep learning is the most popular field in recent years, deep learning frameworks are getting more diverse and powerful. The Python programming language was used as the base of this framework. The following tools were used on the top of python programming language during this research.

a) Python Language

Python language serves as a powerful and flexible tool for implementing algorithms and building models. It enables developers to easily express complex mathematical computations and build intricate neural networks, its simplicity, readability, and robust ecosystem.

b) TensorFlow

TensorFlow is a popular open-source deep learning framework developed by the Google Brain team. It provides a comprehensive set of tools and libraries for building and deploying machine learning models, particularly deep neural networks.

c) Keras

Keras is a high-level neural networks API written in Python. It is designed to be easy to use and efficient, and it can be used to build a wide range of deep learning models, including convolutional neural networks (CNNs). Keras is built on top of TensorFlow, a popular deep learning framework, and it provides a number of features that make it a popular choice for deep learning practitioners.

d) Jupyter Notebook

During this research, we have used Jupyter Notebook as an interactive computing environment that is specifically designed for developing, training, and evaluating deep learning models. It is integrated with popular deep learning frameworks called TensorFlow, etc.

3.10 Limitations

Some potential limitations of this study include the data availability and all possible parallel computing strategies have not been covered.

3.11 Ethical Considerations

The ethical considerations in the study included:

Ensuring the privacy and confidentiality of data sources. In addition, establishing clear guidelines for data sharing and collaboration with other researchers, ensuring responsible data usage and attribution. The Protection for intellectual property rights associated with the developed models and algorithms, adhering to relevant copyright and licensing regulations.

CHAPTER FOUR: PRESENTATION, ANALYSIS AND INTERPRETATION OF FINDINGS

4.1 Introduction

This chapter presents the findings from the study on the influence of parallel computing on building deep learning models for the classification of bean diseases. It includes a detailed analysis of the data and the results obtained from various experiments. The objective is to provide a comprehensive demonstration of how parallel computing influences the efficiency and accuracy of deep learning models in this specific context. The chapter is structured as follows: Section 4.2 focused on data visualization to illustrate the dataset characteristics and preliminary results. Section 4.3 discussed the role of image data augmentation in enhancing model performance. Section 4.4 explored the use of transfer learning with MobileNetV2 to improve model accuracy. Section 4.5 compared the performance of parallel and serial computing approaches. Section 4.6 tackled the deployment of the model on mobile devices.

4.2 Data visualization

Data visualization showed the dataset used while training and evaluating the deep learning models. It helps in identifying patterns, distributions, and potential anomalies in the data. In this section, we presented visualizations of the dataset, including the distribution of different classes of bean diseases, sample images, and other relevant preprocessing.

4.2.1 Distribution of Bean Disease Classes

To begin, we analyzed the distribution of different bean disease classes in the dataset. This helps in understanding the balance of the dataset and the representation of each disease class.

Figure 29: *Distribution of images in the train, validation, and test set*

As shown by the length of the bars, the dataset is well-balanced across the sets with their respective classes.

4.2.2 Sample Images of Bean Diseases

Visualizing sample images from each class provides insight into the characteristics of the diseases and the challenges involved in classification.

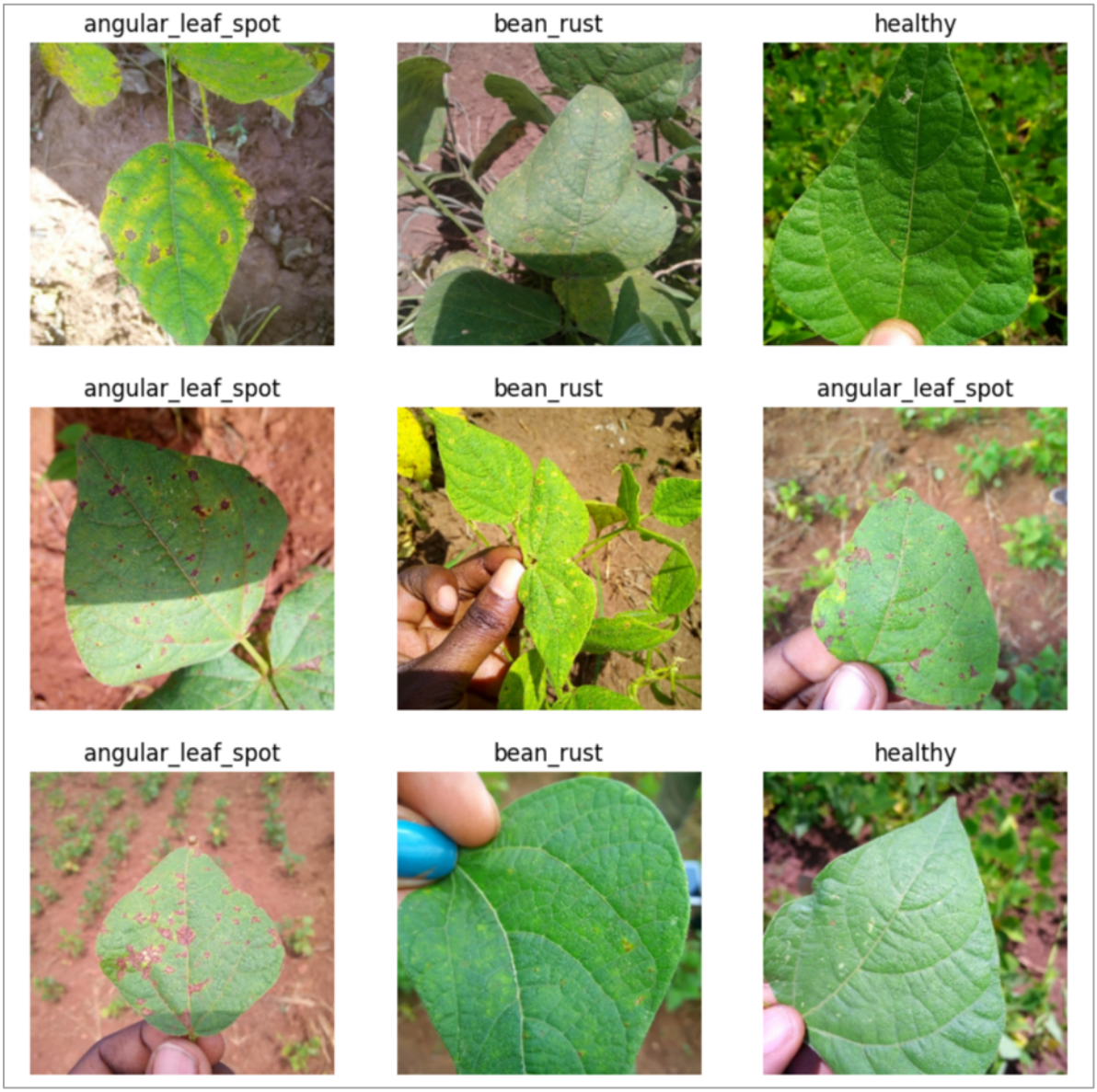


Figure 30: Random images from each class

The sample bean leaf images used in this study are shown in the figure above. Visually, distinguishing bean rust from angular leaf spot is not easy due to the presence of spots in both leaves. However, determining a healthy leaf from the two leaves is easy since it has no spot.

4.3 Image Data Augmentation

Data augmentation techniques were used to increase the diversity of the training data without actually collecting new data. It involves applying various transformations to the existing images, such as rotations, translations, and flips, to create new, modified versions. This helps in improving the robustness and generalization ability of the deep learning models.

4.3.1 Augmentation Techniques

The following augmentation techniques were applied to increase the bean disease dataset:

i. **Rotation**: Random rotations within a specified range (-20 to 20 degrees).
ii. **Translation**: Random horizontal and vertical shifts (20% of the image width/height).
iii. **Shear**: Random shearing transformations.
iv. **Zoom**: Random zoom in and out.
v. **Horizontal Flip**: Random horizontal flipping of images.
vi. **Normalization**: Scaling pixel values to the range [0, 1].

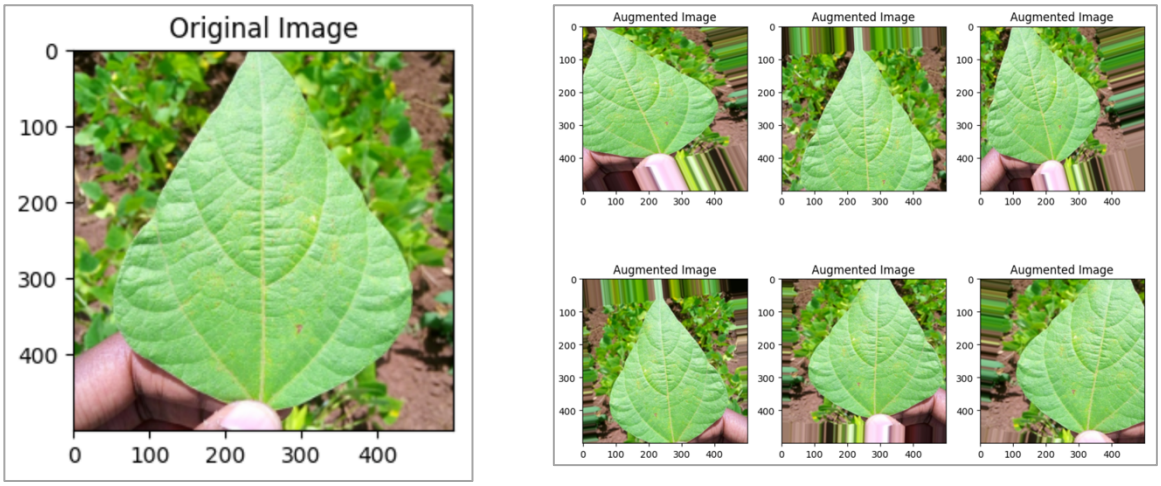


Figure 31: Original vs. Generated images from image augmentation techniques

This visualization of the augmented images was important as we were able to see the different versions of the leaf images that we were to train our model with.

4.4 Transfer Learning with MobileNetV2

Transfer learning was involved by using a pre-trained model on a large dataset and fine-tuning the bean disease classification model. In this study, we used the MobileNetV2 model, pre-trained on the ImageNet dataset, to improve our model.

4.4.1 Model Architecture

The MobileNetV2 model was used as the base model, with additional layers added for fine-tuning the bean disease dataset.

```
# Load MobileNetV2 base model pre-trained on ImageNet
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(img_width, img_height, num_classes))
```

Figure 32: Base model creation from transfer learning

We have chosen the MobileNetV2 base model for transfer learning due to two main reasons, such as a strong performance and well-designed Architecture.

4.5 Parallel vs Serial Computing

Parallel computing leverages multiple processors to perform computations simultaneously, significantly reducing training times for models. We have compared the performance of parallel and serial computing approaches in the context of training deep learning models for the classification of bean diseases, then the following findings were observed.

a) Parallel Computing

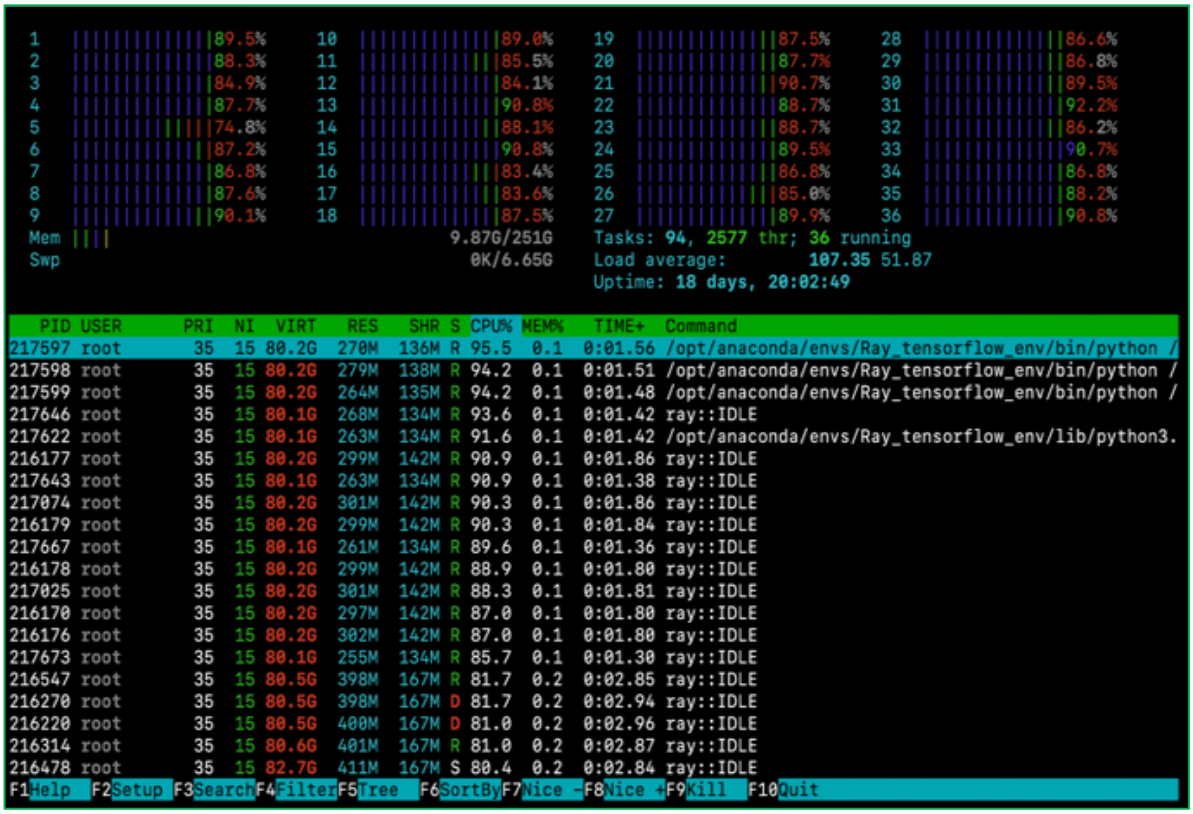


***Figure 33**: Parallel Computing*

We have used Parallel Computing to execute multiple instructions simultaneously by dividing the task into smaller sub-tasks that run concurrent on **36 cores** per node. We Utilized multiple CPU cores on multiple nodes which are called "**Distributed Processing** " to perform computations simultaneously. It reduced training time significantly for this large dataset. Additionally, it was capable of handling larger batches due to distributed memory across multiple nodes as shown on the figure above.

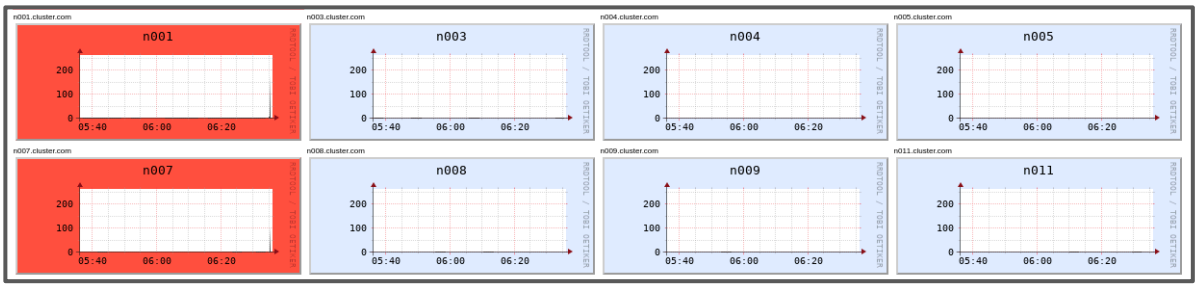


Figure 34: Dashboard of Parallel processing while is ongoing

We have used a dashboard provided by the ganglia tool to monitor deep learning workloads across multiple nodes (**n001** and **n007**) for parallel computing with distributed systems such as clusters and grids.

b) Serial Computing

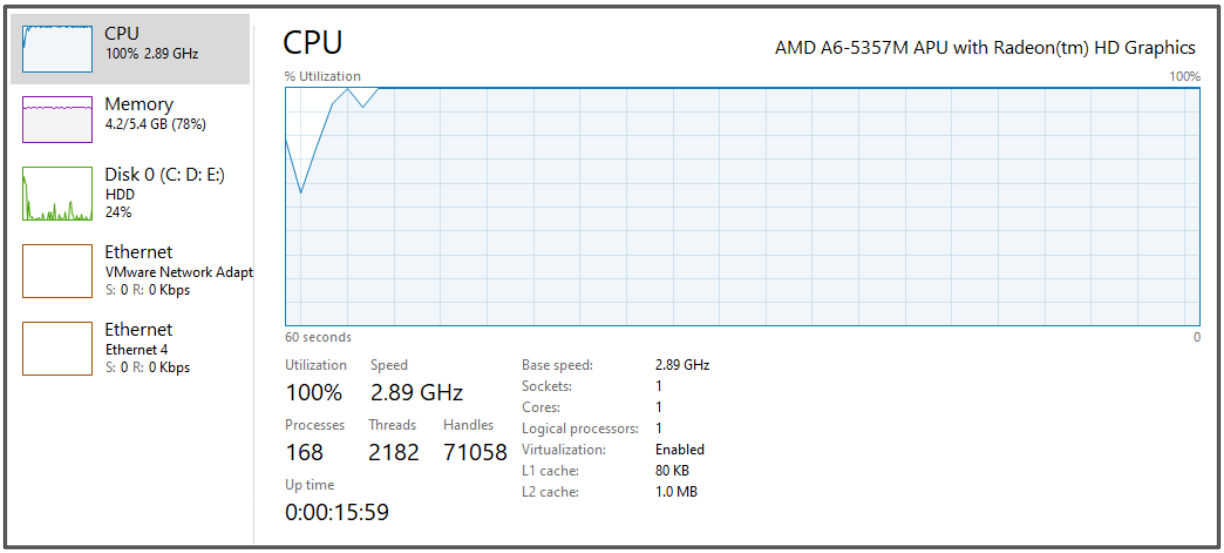


Figure 35: Serial Processing

The Serial Computing executed one instruction at a time in a sequential manner. The process data were in a step-by-step fashion and the performance was generally slow due to the sequential of tasks. The speed was limited by a single core and cannot exploit modern multi-core architectures effectively. It was taking a long training time for large datasets and complex models, because the serial computing was limited to the capabilities of a single core.

4.5.1 Training Time Findings

To evaluate the impact of parallel computing on training time, we trained the same deep learning model using both serial and parallel computing approaches. The training times were recorded and compared as follows:

```
# Train model
start_time = time.time()
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples/train_generator.batch_size,
    epochs=64,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples/validation_generator.batch_size)
training_time = time.time() - start_time
```

***Figure 36**: Training Time on Parallel vs Serial Computing*

*With Parallel Computing **Training Time: 201.33 (s) ≈ 3 mins**

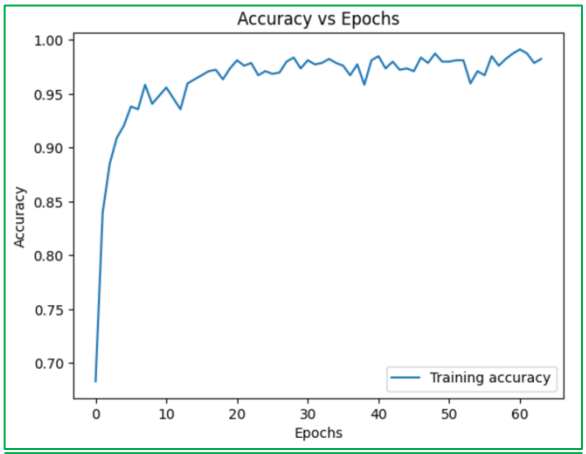
*With Serial Computing **Training Time: 3049.52 (s) ≈ 51mins**

We observed that, when employing parallel computing, the training time of the deep learning model was taking approximately **3 minutes**. This efficiency was achieved by distributing computations across multiple processors, allowing simultaneous processing of data and tasks. In contrast, serial computing, which processes tasks sequentially, took extremely longtime around **51 minutes** to complete the same training. This stark difference highlights the advantage of parallel computing in handling the intensive computations typical in deep learning tasks, thereby accelerating the training process and improving overall efficiency.

4.5.2 Model Performance Findings

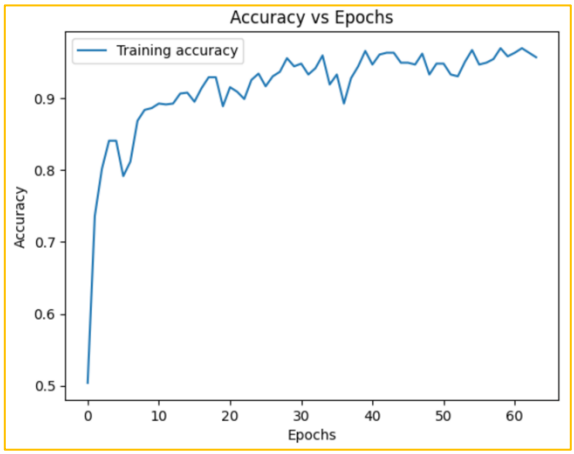
Our model was trained using parallel computing approach and achieved higher accuracy due to the ability to process larger datasets and reduce the risk of overfitting by mitigating the use of larger datasets and regularization techniques as shown on below figures.

a) Accuracy



*Accuracy on Parallel Computing: **0.93***

VS



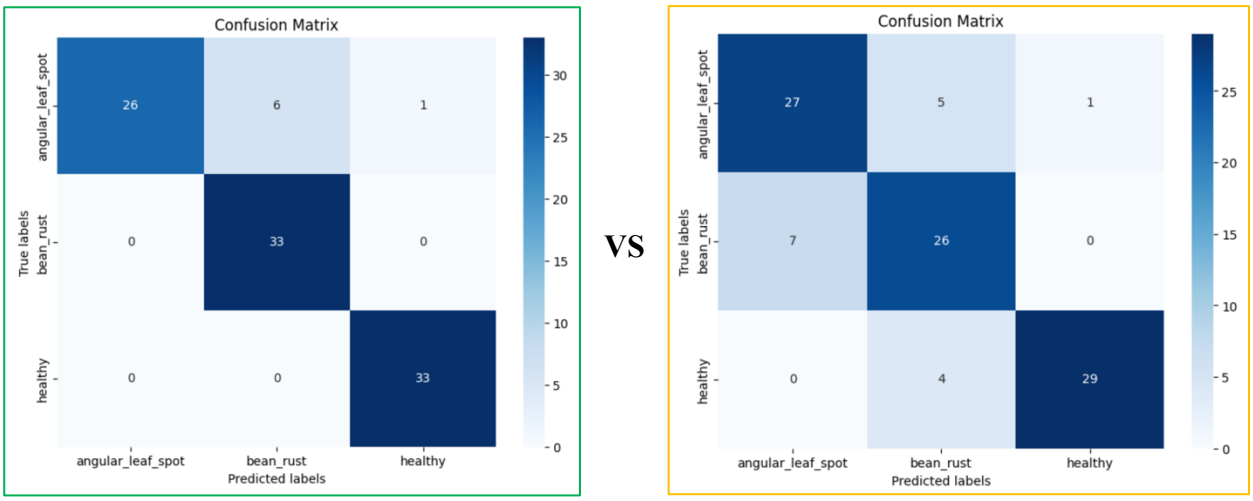
*Accuracy on Serial Computing: **0.83***

***Figure 37**: Accuracy obtained on Parallel vs Serial Computing*

Using parallel computing, the deep learning model achieved an accuracy of approximately **0.93**. This is due to the ability to process larger batches of data simultaneously and utilize more complex architectures efficiently. In contrast, with serial computing, the model reached an accuracy of around **0.83**, likely due to limitations in processing power and the inability to

handle larger datasets effectively. This demonstrates the significant benefits of parallel computing in both speed and accuracy for deep learning tasks.

b) Confusion Matrix



(i) Parallel Computing **(ii) Serial Computing**

Figure 38: Confusion Matrix on Parallel vs Serial Computing

i) With Parallel Computing, the deep learning confusion matrix summarizes the model's performance across different classes as follows:

a. "Angular_leaf_spot" (Predicted vs Actual): The model predicted **26** instances correctly as Angular_leaf_spot, misclassified **6** instances of bean_rust as Angular_leaf_spot, and misclassified **1** instance of healthy as Angular_leaf_spot.

b. "Bean_rust" (Predicted vs Actual): The model accurately predicted **33** instances as bean_rust, with no misclassifications.

c. "Healthy" (Predicted vs Actual): The model correctly predicted **33** instances as Class healthy, with no misclassifications. This confusion matrix demonstrates strong performance for Bean_rust and Health, with perfect classification. As a result, this model is effective in distinguishing between the different classes.

ii) With Serial Computing, the deep learning confusion matrix reveals the model's classification performance across multiple classes as follows:

a. "Angular_leaf_spot" (Predicted vs Actual): The model correctly predicted **27** instances as Angular_leaf_spot, misclassified **5** instances of bean_rust as Angular_leaf_spot, and misclassified **1** instance of healthy as Angular_leaf_spot.

b. "Bean_rust" (Predicted vs Actual): The model correctly predicted **26** instance as bean_rust, misclassified **7** instances of Angular_leaf_spot as bean_rust, and no misclassification of healthy as bean_rust.

c. "Healthy" (Predicted vs Actual): The model correctly predicted **29** instances as healthy, misclassified **4** instances of bean_rust as healthy, and no misclassification of Angular_leaf_spot as healthy.

c) Classification Report

Classification Report:

	precision	recall	f1-score	support
angular_leaf_spot	1.00	0.79	0.88	33
bean_rust	0.85	1.00	0.92	33
healthy	0.97	1.00	0.99	33
accuracy			0.93	99
macro avg	0.94	0.93	0.93	99
weighted avg	0.94	0.93	0.93	99

(i) Parallel Computing

Classification Report:

	precision	recall	f1-score	support
angular_leaf_spot	0.79	0.82	0.81	33
bean_rust	0.74	0.79	0.76	33
healthy	0.97	0.88	0.92	33
accuracy			0.83	99
macro avg	0.83	0.83	0.83	99
weighted avg	0.83	0.83	0.83	99

(ii) Serial Computing

***Figure 39**: Classification Report on Parallel vs Serial Computing*

i) With Parallel Computing, the classification report for the trained deep learning model on distributed systems "**Parallel Computing**" shows strong performance across three categories: **angular leaf spot, bean rust**, and **healthy**. The model achieved an overall accuracy of **0.93** based on the support of **99** samples. **Precision**, **recall**, and **F1-score metrics** for each class indicate high reliability, with the "healthy" class performing the best (**F1-score** of **0.99**) and the "bean rust" and the "healthy" classes showing the highest recall at **1.00**. The macro and weighted averages for precision, recall, and F1-score are all consistently at **0.93**, demonstrating balanced performance across all categories.

ii) With Serial Computing, the classification report shows a moderate performance of the model to classify bean diseases by identifying three different plant conditions: angular leaf spot, bean rust, and healthy leaves. The model achieved an average precision, recall, and F1-score of **0.83** across all classes. Additionally, the moderate accuracy of **0.83** indicates the model correctly classified most of the **99** leaves it analyzed. However, it's important to note that the macro and weighted averages are also both **0.83**, which shows that the model needs to be improved for better accuracy and performance across all three classes.

4.5.3 Model Performance Testing Findings

During the testing phase for our model performance on unseen images, we observed that the model's predictions against the actual labels of the leaf samples had a high performance as shown in Figure 40. Out of the six tests conducted, the model correctly predicted the disease classification in five instances. This indicates that the model has a high level of accuracy, successfully identifying the correct disease in most cases.

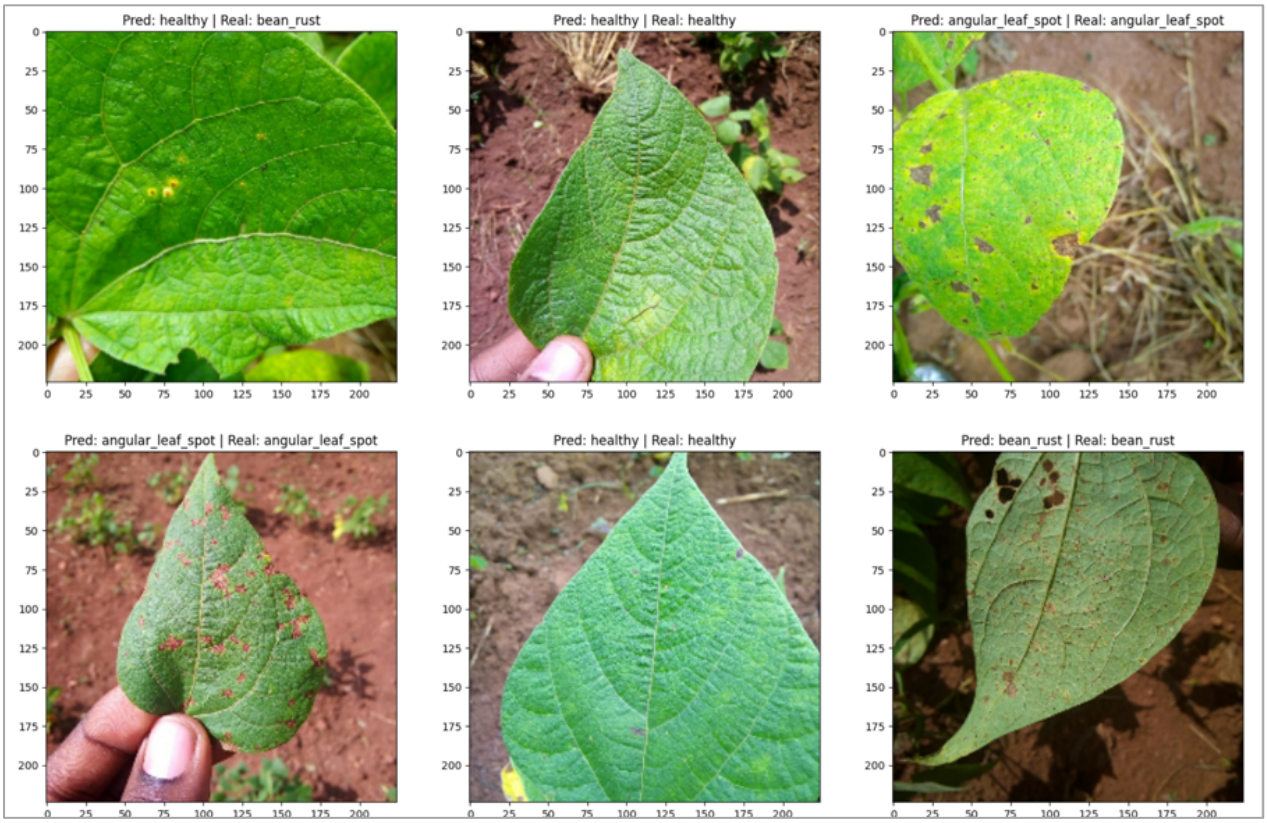


***Figure 40**: Testing model performance*

4.6 Model deployment on mobile devices Findings

Model deployment on mobile devices offers significant advantages in terms of accessibility and usability in field conditions, especially for applications like disease detection in crops.

4.6.1 Model Optimization

To deploy the model on mobile devices, it is essential to optimize it for size and speed. Techniques such as model quantization, pruning, and converting to TensorFlow Lite (TFLite) format are employed.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open("model.tflite", 'wb') as f:
  f.write(tflite_model)
```

***Figure 41**: Model conversion in TensorFlow Lite format*

4.6.2 Inference on Mobile Devices

The optimized model has been deployed on mobile devices by using TensorFlow Lite, which provides tools for running inference efficiently on edge devices like smartphones.

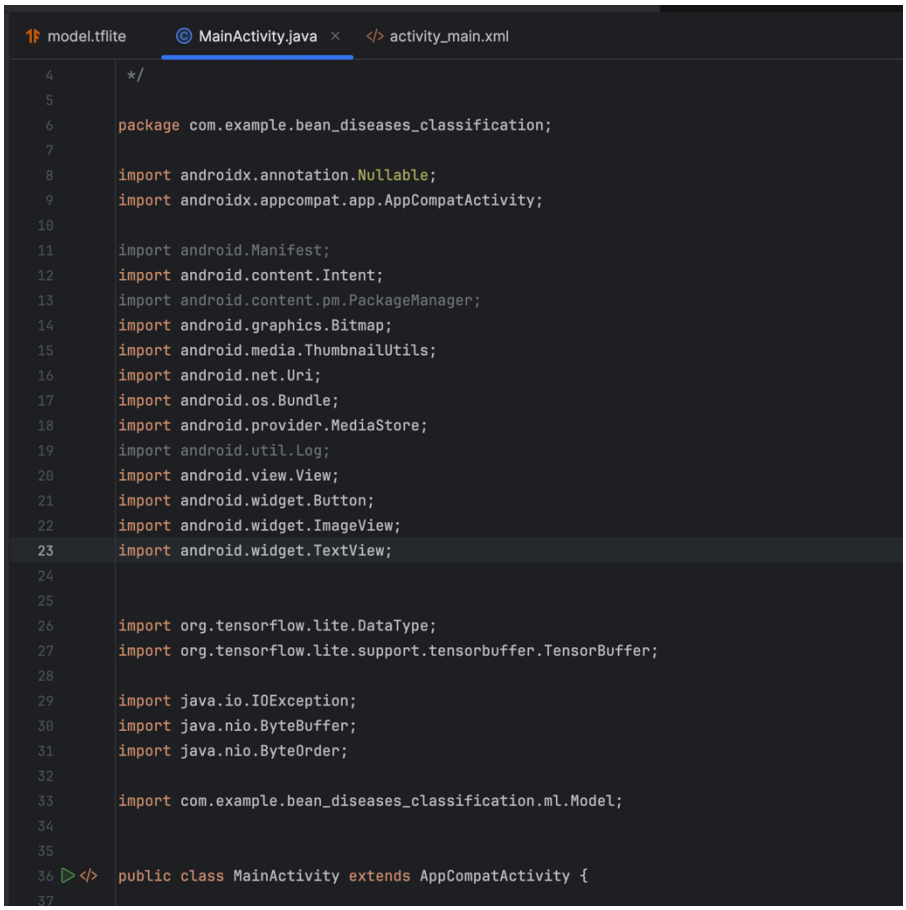


***Figure 42**: Model deployment on Android smartphones*

4.6.3 Real-World Application

Deploying the model on mobile devices allows farmers and agricultural workers to use their smartphones to quickly and accurately diagnose bean diseases in the field. This has led to timely interventions and better crop management practices in bean disease detection.

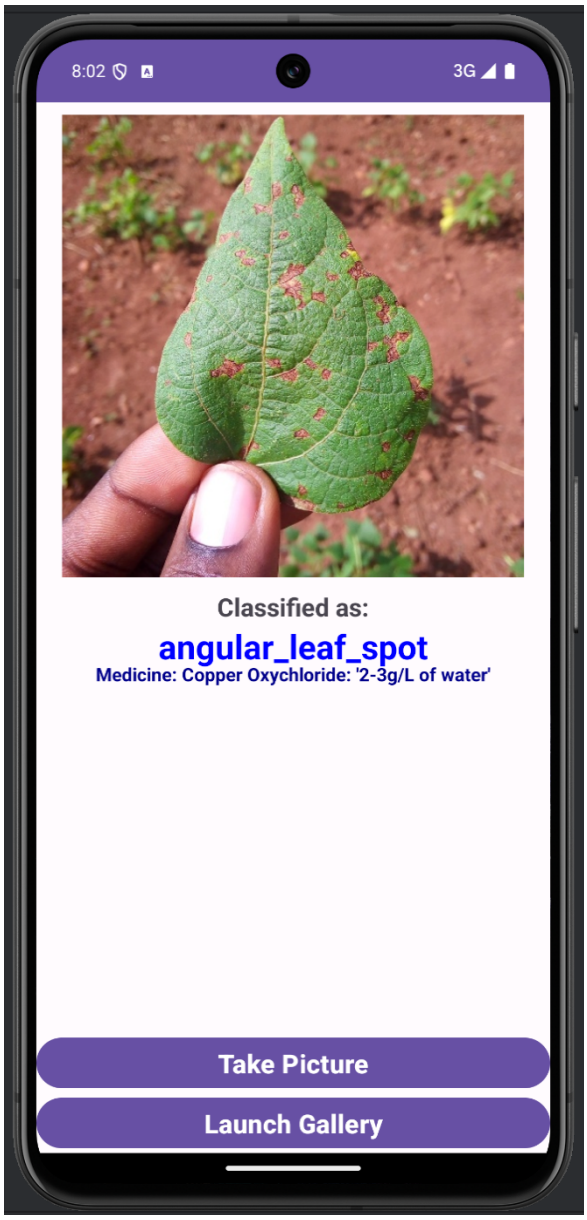


Figure 43: Model in Real-World Application

This application running on mobile devices will significantly benefit farmers and agricultural workers. The farmers will diagnose bean diseases directly in the field by capturing images with their smartphones. This eliminates the need for sending samples to labs for analysis, saving valuable time, etc. Moreover, early detection is crucial for containing outbreaks and preventing significant yield loss.

CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This chapter discussed the findings from the study on the influence of parallel computing on building deep learning models for the classification of bean diseases. This chapter includes a comprehensive summary of the experimental results, conclusions drawn from the study, recommendations based on the findings, and potential areas for future research.

5.2 Summary of Study Findings

This study investigated the role of parallel computing in enhancing the efficiency and performance of deep learning models for classifying bean diseases, yielding several key findings. Firstly, parallel computing significantly reduced the training time of these models; models that previously took several hours to train on a single CPU were trained in a fraction of the time using multiple nodes with multiple processors. Specifically, the bean disease classification model achieved an accuracy of 0.93 with parallel computing, compared to 0.83 with serial computing. Additionally, the training time was reduced to 3 minutes with parallel computing, whereas it took 51 minutes with serial computing. This increase in computational efficiency not only expedited the training process but also improved model performance by allowing for more extensive experimentation and optimization of model architectures and hyperparameters. Additionally, the use of image data augmentation techniques, which improved the robustness and generalization of the models, was efficiently managed through parallel computing, ensuring that augmentation processes did not become a training bottleneck. Moreover, the application of transfer learning with the MobileNetV2 model pre-trained on ImageNet proved highly effective for bean disease classification. Fine-tuning this pre-trained model on the bean disease dataset resulted in high accuracy and reduced training time compared to training from scratch. Finally, the study demonstrated the feasibility of deploying optimized deep-learning models on mobile devices. Techniques such as model quantization and conversion to TensorFlow Lite ensured efficient model operation on resource-constrained devices, making real-time disease diagnosis in the field possible.

5.3 Conclusion

This study conclusively showed that parallel computing demonstrably accelerated the development and deployment of a deep-learning model for bean disease classification. Leveraging parallel computing yielded a significant improvement in model accuracy, achieving a classification accuracy of 0.93 compared to 0.83 obtained with serial computing. Furthermore, parallel computing drastically reduced training time, taking only 3 minutes

compared to 51 minutes with serial processing. This enhanced efficiency facilitates faster model iterations, exploration of a broader range of hyperparameters, and ultimately, the development of more robust and accurate models. Additionally, the ability to deploy these models on mobile devices paves the way for real-time disease diagnosis in agricultural fields, empowering farmers and agricultural workers.

5.4 Recommendations

Based on the study findings, the following recommendations are made: First, we encourage individuals to create their own high-performance computing clusters using Raspberry Pi nodes, which offers a cost-effective solution for setting up HPC clusters and utilizing parallel computing for model training and heavy computational tasks. Second, researchers and practitioners should adopt parallel computing frameworks to accelerate the training process and enhance model performance. Third, leveraging transfer learning with pre-trained models can significantly reduce training time and improve accuracy, particularly when dealing with limited datasets. Fourth, implementing comprehensive data augmentation techniques consistently can enhance model robustness and generalization capabilities. Finally, optimizing models for mobile deployment is crucial, as it enables real-time, on-the-field diagnostics, greatly benefiting agricultural practices by providing valuable tools for farmers and agricultural workers.

5.5 Future Work

Future research can build on this study by exploring several key areas. Firstly, transfer learning from models pre-trained on agriculture-specific datasets should be investigated to further enhance performance in agricultural applications. Secondly, integrating deep learning models with edge computing and IoT devices can create comprehensive, scalable solutions for precision agriculture, improving real-time data processing and decision-making capabilities. Lastly, extending the study to include a wider range of crop diseases and different types of crops will help validate the generalizability of the approach and provide more comprehensive diagnostic tools for farmers, ultimately supporting broader agricultural practices.

REFERENCES

Cheng, Y. e. (2019). *Bean leaf disease detection and classification based on deep residual learning.* Computer and Electronics in Agriculture.

Deep Learning on Supercomputers. (n.d.). Retrieved from https://towardsdatascience.com/: https://towardsdatascience.com/deep-learning-on-supercomputers-96319056c61f

Elhoucine Elfatimi, R. E. (2023, November). *Impact of datasets on the effectiveness of MobileNet for beans leaf disease detection*. Retrieved from SpringerLink: https://link.springer.com/article/10.1007/s00521-023-09187-4

Geng. (2020). *Parallel computing for training deep learning models for beans disease classification.* IEEE international Conference.

Hoang-Tu Vo, L.-D. Q. (2023). *Ensemble of Deep Learning Models for Multi-plant Disease Classification in Smart Farming.* Cantho City, Vietnam: Software Engineering Department, FPTUniversity.

Jean B. Ristainoa, P. K. (2021). *The persistent threat of emerging plant diseasepandemics to global food security.* Manhattan: Barbara Valent, Kansas State University, Manhattan, KS.

Kahira, A. N. (2021). *Convergence of Deep Learning and High Performance Computing: Challenges and Solutions.* Barcelona: Universitat Politecnica de Catalunya.

Kolodziejczak, K. P. (2020). *The Role of Agriculture in Ensuring Food Security in Developing Countries. Considerations in the Context of the Problem of Sustainable Food Production.* Poznan, Poland: Department of Economics and Economic Policy in Agribusinesses, Faculty of Economics and Social Sciences, Poznan University of Life Sciences, Wojska Polskiego 28, 60-637 Poznan, Poland.

Michelle M. Nay, T. L.-V.-C. (2018). *A Review of Angular Leaf Spot Resistance in Common Bean.* Parana: Dep. Agronomia, Univ. Estadual de Maringá, Maringá, Paraná, Brazil.

NVIDIA cuDNN. (n.d.). Retrieved from https://docs.nvidia.com/cudnn/index.html: https://docs.nvidia.com/cudnn/index.html

NVIDIA Tesla P100 PCIe 16 GB. (n.d.). Retrieved from techpowerup: https://www.techpowerup.com/gpu-specs/tesla-p100-pcie-16-gb.c2888

P. Pamela, D. M. (2014). *Severity of angular leaf spot and rust diseases on common beans in Central Uganda.* Kampala: National Crops Resources Research Institute, Namulonge.

Paymode, A. S. (2021). *Transfer Learning for Multi-Crop Leaf Disease Image Classification using Convolutional Neural Network VGG.* MGM's Jawaharlal Nehru Engineering College, Aurangabad 431001, Maharashtra, Inda.

Prathamesh Borhade, R. D. (2020). *Image Classification using Parallel CPU and GPU Computing.* International Journal of Engineering and Advanced Technology(IJEAT), ISSN: 2249 - 8958, Volume-9 Issue-4.

Seyed Hossein Nazer Kakhki, M. V. (2022). *Predict bean production according to bean growth, root rots, fly and weed development under different planting dates and weed control treatments.* Kermanshah: Plant Protection Research Department, Kermanshah Agricultural & Natural Resources Research & Education Center, AREEO, Kermanshah, Iran.

Shouan Zhang, N. D. (2019). *Disease Control for Snap Beans in Florida.* IFAS Extension, University of Florida.

Slurm & Deep Learning. (n.d.). Retrieved from run.ai: https://www.run.ai/guides/slurm/slurm-deep-learning

Stratified Random Sampling. (n.d.). Retrieved from Questionpro: https://www.questionpro.com/blog/stratified-random-sampling/

Wang, X. e. (2019). *A GPU-accelerated deep learning framework for bean disease classification.* Computers and Electronics in Agriculture.

What Is CUDA? (n.d.). Retrieved from https://blogs.nvidia.com: https://blogs.nvidia.com/blog/what-is-cuda-2/

Wu, W. e. (2021). *Parallel computing for training models for multi-plant disease classification.* In Proceedings of the 2021 International Conference on Artificial Intelligence and Machine Learning(ICAIML)).

Xidong Wu, P. B. (2023). *Performance and Energy Consumption of Parallel Machine Learning Algorithms.* ECE 2166.

Yang, S. J. (2010). *A survey on transfer learning. .* IEEE Transactions on knowledge and data engineering, 22(10):1345-1359.

Yi, F. (2015). Current and Prospective Methods for Plant Disease Detection.

Zhang, X. e. (2018). *A convolutional neural network approach for bean disease classification using image analysis.* Sensors.

APPENDICES

Appendix A. WORK PLAN

Activities	Months						
	1	2	3	4	5	6	7
Acceptance of research topic							
Development of chapter one of research proposal							
Writing of literature review of proposal							
Writing of research design and methodology							
Defense of research proposal and corrections							
Data collection							
Analysis and interpretation of findings							
Submission of final report and defense							

***Table 3**: Working plan*

Appendix B. BUDGET ESTIMATION

Number	Item	Cost (Rwf)
1	Thesis fee	350,000
2	Computer device	400,000
3	Research proposal printing	15,000
4	Research proposal binding	5,000
5	Editing and printing of corrections	10,000
6	Internet research	150,000
7	Data Collection	50,000
8	Binding and printing of final report	40,000
Total amount		**1,020,000**

***Table 4**: Budget estimation*

Appendix C. MODEL DEPLOYMENT IN ANDROID SMARTPHONES

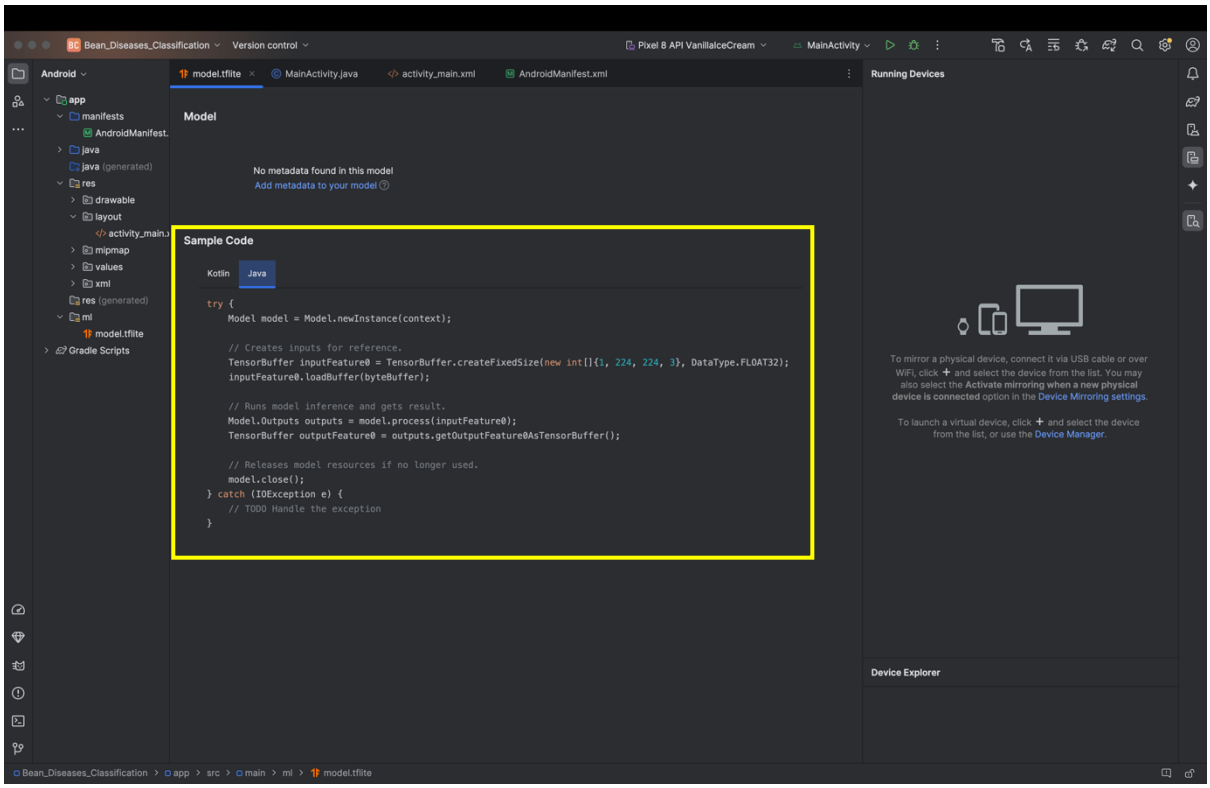


***Figure 44**: Importing TensorFlow Lite Model in Android Studio*

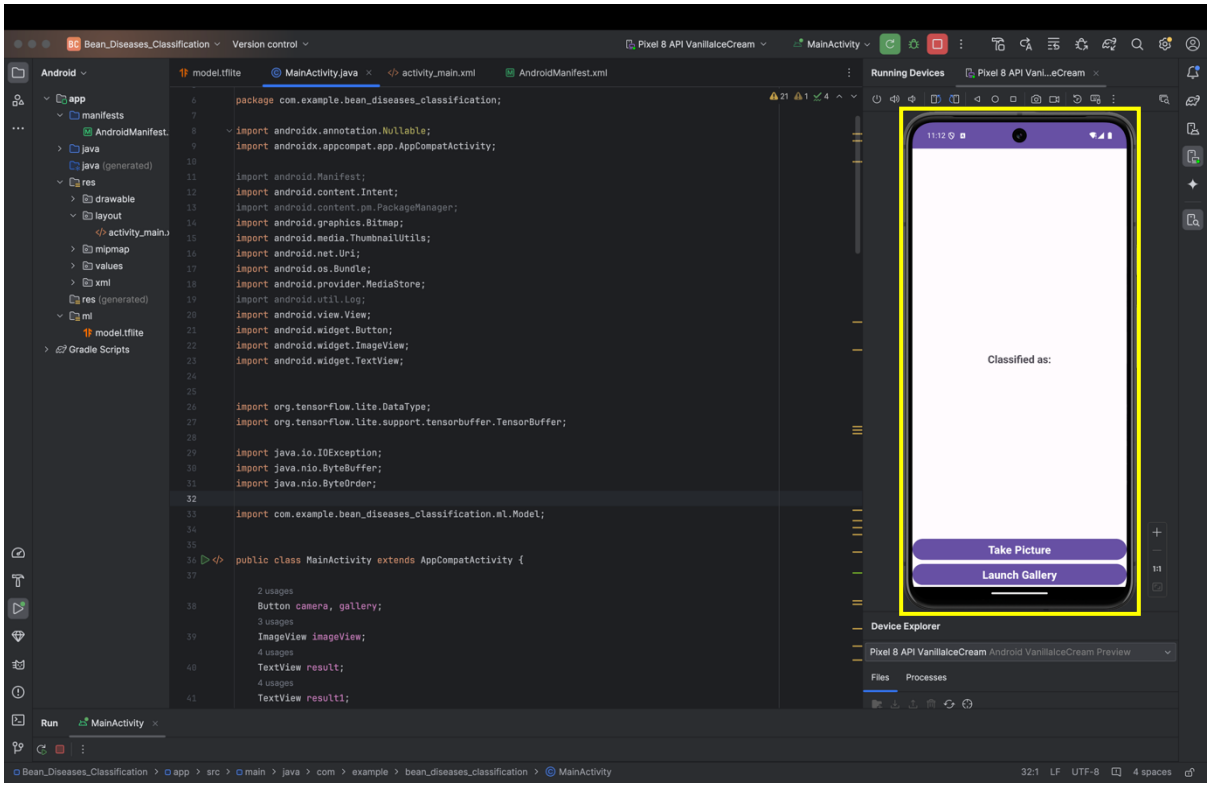


***Figure 45**: User Interface Application for Bean Disease Classification*

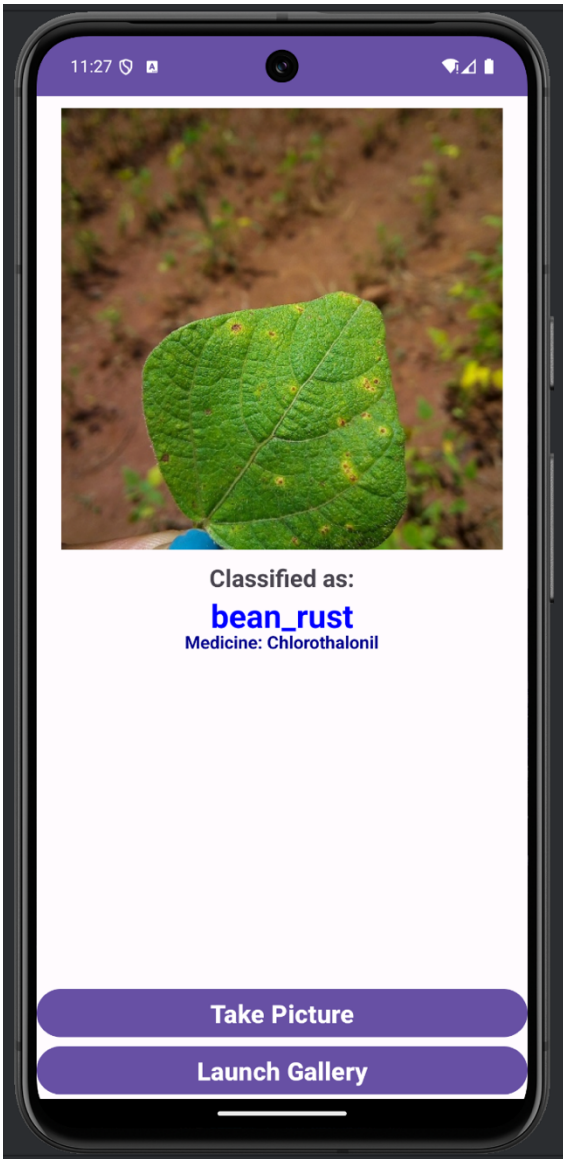


***Figure 46**: Farmers predict Bean disease using their Smartphones*

Appendix D. CODES USED FOR MODEL DEPLOYMENT IN ANDROID SMARTPHONES

```
/*
 * Created by GASHUGI Jean Bosco
 */

package com.example.bean_diseases_classification;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.media.ThumbnailUtils;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;


import org.tensorflow.lite.DataType;
import org.tensorflow.lite.support.tensorbuffer.TensorBuffer;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;

import com.example.bean_diseases_classification.ml.Model;


public class MainActivity extends AppCompatActivity {

    Button camera, gallery;
    ImageView imageView;
    TextView result;
    TextView result1;
    int imageSize = 224;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        camera = findViewById(R.id.button);
```

```
        gallery = findViewById(R.id.button2);

        result = findViewById(R.id.result);
        result1 = findViewById(R.id.result1);
        imageView = findViewById(R.id.imageView);

        camera.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(cameraIntent, 3);

            }
        });
        gallery.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent cameraIntent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
                startActivityForResult(cameraIntent, 1);
            }
        });
    }

    public void classifyImage(Bitmap image){
        try {
            Model model = Model.newInstance(getApplicationContext());

            // Creates inputs for reference.
            TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1, 224, 224, 3},
DataType.FLOAT32);
            ByteBuffer byteBuffer = ByteBuffer.allocateDirect(4 * imageSize * imageSize * 3);
            byteBuffer.order(ByteOrder.nativeOrder());

            int[] intValues = new int[imageSize * imageSize];
            image.getPixels(intValues, 0, image.getWidth(), 0, 0, image.getWidth(),
image.getHeight());
            int pixel = 0;
            //iterate over each pixel and extract R, G, and B values. Add those values individually
to the byte buffer.
            for(int i = 0; i < imageSize; i ++){
                for(int j = 0; j < imageSize; j++){
                    int val = intValues[pixel++]; // RGB
                    byteBuffer.putFloat(((val >> 16) & 0xFF) * (1.f / 1));
                    byteBuffer.putFloat(((val >> 8) & 0xFF) * (1.f / 1));
                    byteBuffer.putFloat((val & 0xFF) * (1.f / 1));
                }
            }

            inputFeature0.loadBuffer(byteBuffer);

            // Runs model inference and gets result.
```

```
        Model.Outputs outputs = model.process(inputFeature0);
        TensorBuffer outputFeature0 = outputs.getOutputFeature0AsTensorBuffer();

        float[] confidences = outputFeature0.getFloatArray();
        // find the index of the class with the biggest confidence.
        int maxPos = 0;
        float maxConfidence = 0;
        for (int i = 0; i < confidences.length; i++) {
            if (confidences[i] > maxConfidence) {
                maxConfidence = confidences[i];
                maxPos = i;
            }
        }
        String[] classes = {"angular_leaf_spot", "bean_rust", "healthy"};
       // result.setText(classes[maxPos]);
        if (classes[maxPos] == "angular_leaf_spot")
        {
            result.setText(classes[maxPos]);
            result1.setText("Medicine: Copper Oxychloride: '2-3g/L of water'");
        }
        else if (classes[maxPos] == "bean_rust") {
            result.setText(classes[maxPos]);
            result1.setText("Medicine: Chlorothalonil");
        }
        else
        {
            result.setText(classes[maxPos]);
            result1.setText("Medicine: Not required.");
        }

        // Releases model resources if no longer used.
        model.close();
    } catch (IOException e) {
        // TODO Handle the exception
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    if(resultCode == RESULT_OK){
        if(requestCode == 3){
            Bitmap image = (Bitmap) data.getExtras().get("data");
            int dimension = Math.min(image.getWidth(), image.getHeight());
            image = ThumbnailUtils.extractThumbnail(image, dimension, dimension);
            imageView.setImageBitmap(image);

            image = Bitmap.createScaledBitmap(image, imageSize, imageSize, false);
            classifyImage(image);
        }else{
            Uri dat = data.getData();
            Bitmap image = null;
            try {
```

```
                image = MediaStore.Images.Media.getBitmap(this.getContentResolver(), dat);
            } catch (IOException e) {
                e.printStackTrace();
            }
            imageView.setImageBitmap(image);

            image = Bitmap.createScaledBitmap(image, imageSize, imageSize, false);
            classifyImage(image);
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
  }
}
```