Mass plant Disease detection using A.I And Jetson Nano

By

Kelvin Garikayi Zawala

H190140y

HIT400 Capstone project Submitted in Partial Fulfilment of the

Requirements of the degree of

Bachelor of Technology

In

Software Engineering

In the

School of Information Sciences and Technology

Harare Institute of Technology

Zimbabwe

Figure 1.

Supervisor

……Ms Yolanda Chibaya……..

05/2023

Abstract

The Disease Detection System is a technology-based system designed to improve the detection and management of plant diseases in farms and greenhouses. Using advanced computer vision techniques and convolutional neural networks (CNNs), the system can accurately recognize and detect various plant diseases.

The system comprises two main components: the disease recognition component and the disease interpretation component. The recognition component uses a Jetson Nano to run advanced CNNs such as ResNet50, while the interpretation component provides information about the detected diseases and suggests possible treatment options.

With its ability to accurately detect plant diseases and provide valuable information to farmers and greenhouse managers, the Disease Detection System has the potential to revolutionize the way plant diseases are managed in agriculture.

Preface

his document aims to provide a detailed overview of the Disease Detection System, a cutting-edge technology designed to revolutionize the detection and management of plant diseases in farms and greenhouses. Using advanced computer vision techniques and convolutional neural networks (CNNs), the system can accurately recognize and detect various plant diseases, providing valuable information to farmers and greenhouse managers.

The Disease Detection System comprises two main components: the disease recognition component and the disease interpretation component. The recognition component uses a Jetson Nano as the computer of choice, utilizing its CUDA cores to run advanced CNNs such as ResNet50. The system is trained on a large dataset of plant images to enable it to accurately recognize and detect various plant diseases.

The interpretation component of the system uses advanced CNNs to provide information about the detected diseases and suggest possible treatment options. This component can be customized to provide information in different languages depending on the needs of the user.

In this document, I provide a comprehensive introduction to the Disease Detection System, including its history, how it works, and its potential applications in agriculture. I also discuss some of the challenges and limitations of the technology, such as the complexity of plant diseases and the need for continued improvement in accuracy and reliability.

Overall, the Disease Detection System is a promising technology that has the potential to greatly improve the detection and management of plant diseases in agriculture. I hope that this document serves as a valuable resource for those interested in learning more about this exciting technology.

Certificate of Declaration

This is to certify that work entitled "HIT400 Research Topic " *is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology (Hons) in Software Engineering ,Harare Institute of Technology .It is further certified that no part of research has been submitted to any university for the award of any other degree .*

Figure 2.

(Supervisor) Signature…….. Date…….

(Mentor) Signature…….. Date……

(Chairman) Signature…….. Date……..

Dedication

Dedicated to all those who work tirelessly to improve the health and well-being of our planet's flora.

This document is dedicated to the farmers, greenhouse managers, and agricultural scientists who have devoted their lives to understanding and managing plant diseases. To those who have faced the challenges of crop loss and reduced yields due to disease, but have continued to push forward in their pursuit of healthier and more productive plants.

It is for those who have embraced new technologies and techniques in their efforts to improve the detection and management of plant diseases. To those who have shown resilience in the face of adversity, and have used their experiences to inspire others to do the same.

This document is dedicated to those who have dedicated their lives to making our world a greener and more sustainable place. To those who have worked tirelessly to promote sustainable farming practices and protect our natural resources. To those who have used their skills, talents, and resources to help others and make a positive impact in their communities.

May this document serve as a reminder that no challenge is too great when we work together towards a common goal. With hard work, determination, and perseverance, we can improve the health and well-being of our planet's flora.

Declaration

All sentences or passages quoted in this dissertation from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree examination as a whole.

List Of Tables

List Of Images

CHAPTER 1 INTRODUCTION

1.1 Background

Protecting plant health is essential to all industries. Plants not only contribute to the global food system but also provide non-food crops for energy, fiber, fodder and horticulture. Plant diseases are defined as physiological functions of plants stimulated by existing plant pathogens (organisms or infectious agents). Global crop losses are estimated at $220 billion per year, accounting for 14.1% of losses caused by diseases. Culture failure can be caused not only by abiotic factors such as the environment but also by organisms such as oomycetes, fungi, viruses, bacteria, nematodes, and bacteria. Plant diseases reduce product quality and shelf life of crops, reduce the nutritional value of fruits and vegetables, reduce yield, damage aesthetics and make some crops unsaleable.

Today, agricultural planes are high-tech equipment. They've come a long way since the early days of the Tiger Moth and the shawls outside the window. Turbine aircraft today are highly productive and efficient and everything is computerized. They are very expensive but can cover a large area in a short time. An aircraft costs are broken into two areas:

- Fixed costs: These are costs incurred every year, regardless of the amount of work performed. This includes insurance, interest, pilot wages, and some maintenance. That means a plane has to do a lot of work every year to reduce those costs per hectare. One example is insurance which currently costs about $1,000 a week whether we work or not. If we do 1,000 hectares per week, it will be $1 per hectare. If we do 4000 hectares then $0.25c.
- The variable costs: This is the cost per hour to operate the aircraft. If the aircraft is stationary, these costs are not incurred. For example fuel and maintenance. These costs are quite high and range from $10 to $15 per minute. This means that when an aircraft is in flight, it should cover as many hectares per hour as possible.

A combination of the fixed and variable costs makes using an agricultural plane very expensive in mass crop assessment, that is checking crops for diseases. On top of that, it requires a lot of skill and experience to tell which plants are sick from an aerial view of the crop.

1.2 Problem Statement

Population and wealth growth over the next half-century will result in a massive increase in food consumption. To achieve this predicted growth without huge acreage expansion, agricultural yields must dramatically rise. Crop yields are significantly influenced by the insect and disease challenges farmers encounter, as well as the remedies available. Control strategies rely on the accurate identification of illnesses and causative agents. Illness control procedures can be a waste of time and money if the disease and the disease-causing agent are not properly identified. This can lead to additional plant losses. As a result, accurate illness diagnosis is critical. As a farmer scales up production assessment of crops for diseases becomes hard. The farmer may fly over fields but this requires a lot of skill and risks errors.

1.3 Objectives

- To provide a feasible solution that is mobile and can be attached on numerous vehicles.
- To give predictions of affected areas that over 70% close to the ground truth result
- To have a setup that requires 2 times less power than traditional scan setups
- To use 3 times less resources than the traditional `Basant to Air Tractor`
- To have a setup that requires at least 2 times less power than traditional scan setups

1.4 Hypothesis

The **Nvidia Jetson** is an integrated AI engine that consumes little power. It is a series of embedded computing boards from Nvidia that can be used for a variety of AI applications such as autonomous machines, robots, and drones.

The **ZED camera** is a stereo camera that uses neural networks to reproduce human vision and brings stereo perception to a new level. It has a built-in IMU, barometer, and magnetometer for multi-sensor capture. The ZED camera can capture a large-scale 3D map of your environment and understand how objects move through space. It uses advanced sensing technology based on human stereo vision to add depth perception, motion tracking, and spatial understanding to your application.

The **DJI MG-1S** is an octocopter designed for precision variable rate application of liquid pesticides, fertilizers, and herbicides, bringing new levels of efficiency and manageability to agriculture. It has a powerful propulsion system that allows the aircraft to carry up to 10 kg liquid payloads. The combination of speed and power means that an area of 4,000-6,000 m² can be covered in just 10 minutes, or 40 to 60 times faster than manual spraying operations.

Agrobot is a company that builds smart farming machinery such as the Bug Vacuum, an insect vacuum robot that performs way more efficient, accurate and cost-effective than any other comparable solution. They also have the E-Series, the first pre-commercial robotic harvesters for gently harvesting strawberries. From an adaptable mobile platform, robotic manipulators work together to pick just the fruit that meets the farmer's quality standard .

The integration of Nvidia Jetson's low-power AI engine with the ZED camera's advanced sensing technology and spatial understanding capabilities will enable the detection and categorization of illnesses with high accuracy. By attaching this system to a flying drone like the DJI MG-1S or a crawling robot like the Agrobot crawling robot, it will be possible to efficiently map and monitor large areas for signs of illness. Further testing and experimentation will be necessary to validate this hypothesis and determine the effectiveness of the system in real-world scenarios.

1.5 Feasibility study:

- Technical

  The project was helped by technical feasibility in identifying operational challenges and long-term objectives, as well as how to handle the system's technological features. We also evaluated the tools and materials needed to perform the project from start to end, including hardware and software needs, availability or ability to acquire within the time period stated, and the cost of purchasing these tools. We looked into if there was any new technology that may help with the system's development.

  The availability of new emerging technologies, such as the Jetson chip and board, as well as software, the bulk of which is open source software, was an important factor in this study. Because of the internet's accessibility and the availability of free, open-source software, some portions of the project may now be accomplished with very few skills and in less time, saving money by delaying the process. Data is collected quickly, documented, and organised with minimum human mistake. The project is not extremely large, and if everything goes as planned, it should be done

within the required time limit. The project timeframe is likely to be influenced since all resources are available. Costs have been assessed to guarantee that they will not have a long-term impact on the project's technical resources.

- Economic

  The emphasis here is on determining if the cost of creating the entire system will not exceed the proposed budget. A cost-benefit analysis revealed that the application's development could proceed. Because the majority of the development software I will be utilizing is open source, it will be simple and inexpensive to obtain. The hazards anticipated during project development are substantially lower as compared to the advantages of creating it. The requisite processing power is already already available, therefore the cost is covered, the jetson was already available. A break down of the major hardware costs is noted below:

item	**Price**
Nvidia jetson Tx2	**986**
Zedd camera	**449**
Smart Drone with power to support jetson	**2400**
Total	*3838*

Table 1.

- Operational

  The operational feasibility of a proposed system is a measure of how well it addresses difficulties and capitalizes on possibilities identified during scope definition, as well as how well it fits the criteria established during the system development requirements analysis phase. It answers the question "would the system work?" as well as how well the system would fit into existing business structures and how useful it would be once deployed. The project's human resources determine operational feasibility, which comprises projecting whether the system will be used once it is created and deployed. After considering these operational requirements, we find that the proposed system is operationally feasible.

1.6 Project plan

The project will be divided into 4 main phases:

- Analysis and quick design
- Prototyping
- Testing
- deployment

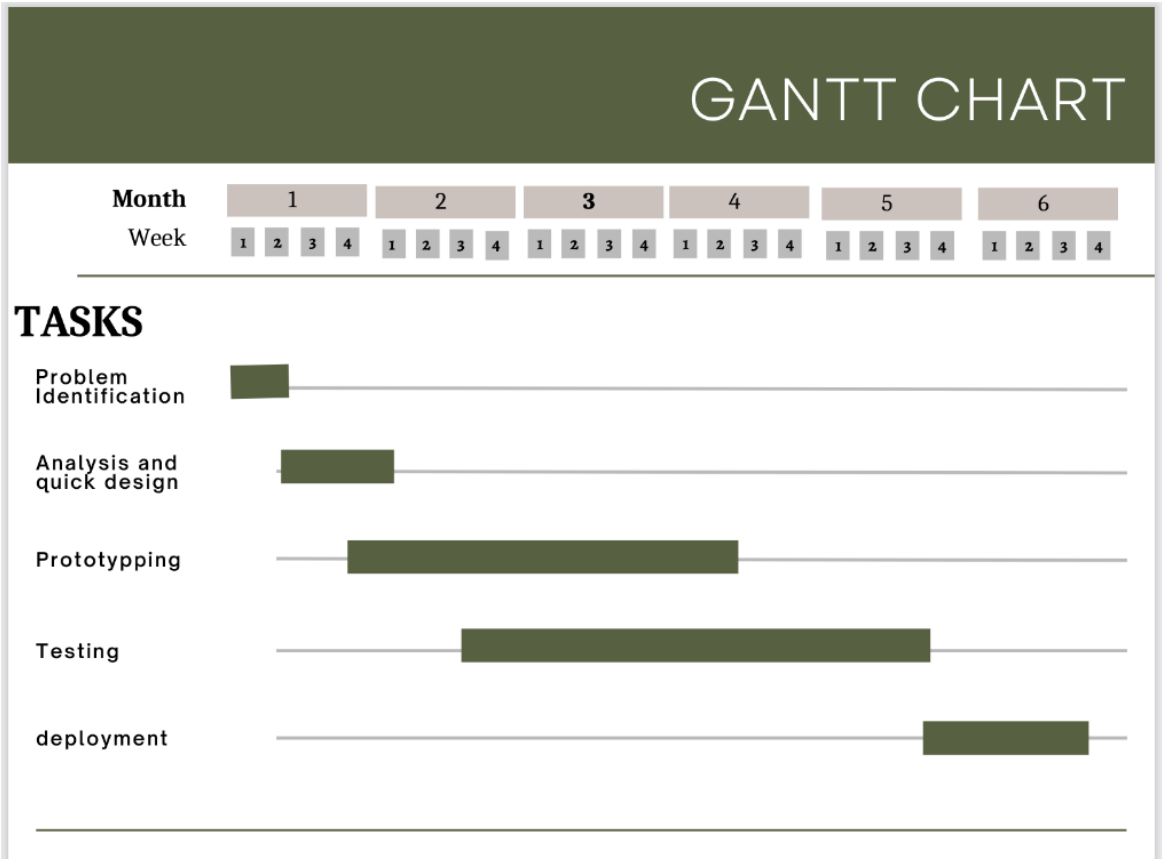


Figure 3.

Fig 1.1

CHAPTER 2 Literature Review

2.1 Introduction

In order to assess existing plant disease detection applications in agriculture, research was conducted. Because there are several detection methods used in agriculture for a wide range of crops. Further research was conducted to look at nutritional shortage detection tools, as they also fall into the same category.

2.2 Related work

- Airborne Sensors Accurately Monitor Crops in Real Time by Michelle Horton

  According to a study team from the University of Illinois, powerful airborne sensors might be crucial in supporting farmers in managing maize sustainably across the US Corn Belt. The study, which uses remote sensors in conjunction with newly developed deep learning algorithms, reliably and swiftly forecasts crop nitrogen, chlorophyll, and photosynthetic capacity.

  "When compared to traditional methods of leaf tissue analysis, remote sensing enables far quicker and more cost-effective methods of monitoring agricultural nutrients." "The immediate and high-resolution crop nitrogen information will be highly useful to producers in diagnosing crop development and guiding adaptive management," said main author Sheng Wang, an associate professor and research scientist at the University of Illinois Urbana-Champaign.

  "Precision agriculture that depends on sophisticated sensing technology and aerial satellite platforms to monitor crops might be the solution," said Kaiyu Guan, Blue Waters Associate Professor at the University of Illinois Urbana-Champaign.

  To overcome these constraints, the researchers presented a hybrid strategy based on hyperspectral imaging and machine-learning algorithms. Hyperspectral imaging, a growing field of remote sensing, use a spectrometer to divide a pixel into hundreds of pictures at various wavelengths, delivering additional information on the acquired image.

  The researchers constructed deep learning models based on aerial reflectance data using Radiative Transfer Modeling and a data-driven Partial-Least Squares

Regression (PLSR) technique. According to the research, PLSR requires a little amount of label data for model training.

The researchers used cuDNN and NVIDIA V100 GPUs to develop deep learning models to predict crop nitrogen, chlorophyll, and photosynthetic capacity at the leaf and canopy levels. The models were around 85% accurate when tested against ground-truth data. The method is quick, scanning fields in a matter of seconds per acre. According to Wang, such technology can be extremely beneficial in determining crop nitrogen condition and production potential.

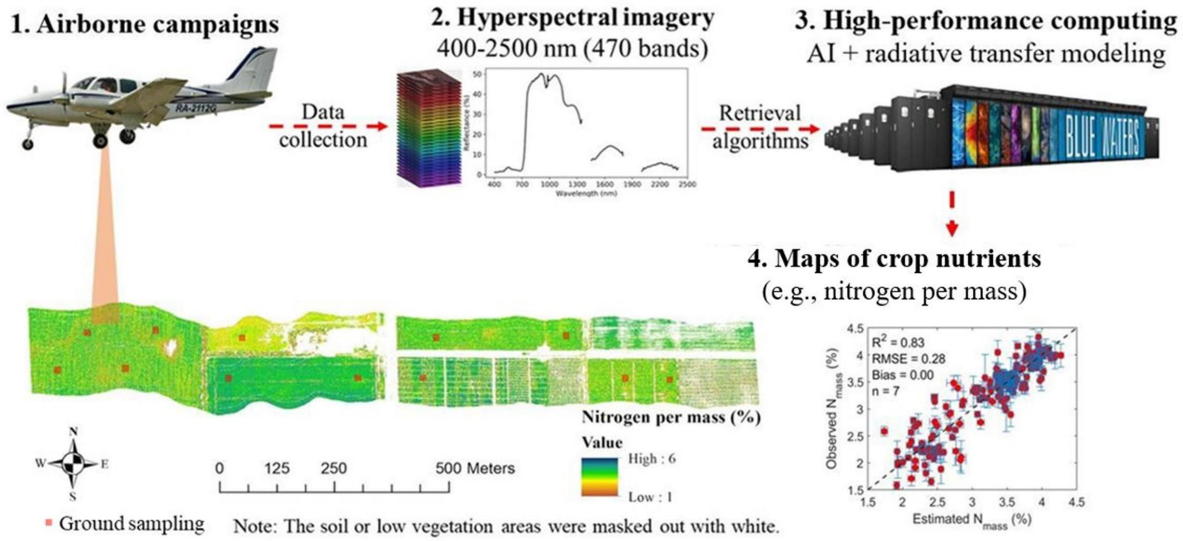


Figure 4. "Airborne hyperspectral imaging of nitrogen deficiency on crop traits and yield of maize by machine learning and radiative transfer modeling." Courtesy of Sheng Wang.

- An Application with Jetson Nano for Plant Stress Detection and On-field Spray Decision Marcos A. de Oliveira Jr.

  Because edge and fog devices have limited processing capability, applications in these settings are instantiated in a particular manner, dealing with a single problem. Precision agriculture has been one of the contexts in which research in edge and fog computing has increased in recent years, owing to the significant effect of technology on the agricultural sector. IoT devices have recently taken use of the Smart Farms idea. Its use provides farmers with near-real-time data regarding farm conditions.

  Artificial Intelligence (AI) tactics (Li et al., 2018) have been included into the solutions with the smart farm idea under construction.

Consider the Jetson Nano as a possibility (Nvidia, 2021). This is an NVidia embedded computer board featuring a low-power system and hardware resources like a GPU designed to speed up machine learning applications. As a result, this research investigates the feasibility of putting AI algorithms on the edge device Jetson Nano, with a focus on detecting stress levels in bean plants. The paper expands on the findings of (de Toledo, 2019), who employed neural networks and machine learning to effectively implement plant stress state categorization approaches.

The studies conducted allowed the validation of an application model using the Jetson Nano device for the detection of stress in bean plants, allowing this information to be used for real-time decision-making in the spraying of agricultural goods in the field.

- Real-time detection and identification of plant leaf diseases using convolutional neural networks on an embedded platform by Ruchi Gajjar

  Primary identification of agricultural disease is done by visual observation by the farmer in underdeveloped nations such as India, where the effective diagnosis of the illness largely rests on the farmer's expertise, experience, and aptitude. As it becomes a question of subjective analysis, there is always the potential for inaccurate illness identification. Plant pathologists or agronomists may even fail to detect a disease. Incorrectly diagnosing the illness, resulting in ineffective countermeasures This presents significant difficulty for accurate crop disease identification in order to give timely treatment and prevent crop loss. Machine learning-based algorithms have been used to detect illness in a crop. However, their breadth and range are restricted to identifying the type of illness in the crop. We employ machine learning to properly identify crop diseases in this research, but we also install it on an embedded platform and do real-time processing for leaf detection and disease diagnosis. The whole goal of this project is to develop a disease detection system on a portable device that farmers may use in the field to identify crop diseases in real time.

  A single shot detector (SSD) is employed to determine the location of the leaf in the field, and a convolutional neural network architecture is constructed and trained to categorize crop disease. To handle the data, the whole detection and identification architecture is moved to Nvidia's Jetson TX1. To perform real-time disease detection in the field, a portable device is built with a camera that captures real-time input from

the field and sends it to Nvidia's Jetson TX1, which recognizes the leaf and the kind of illness present in the crop. Because the gadget is a small, portable equipment, its embedded technology will help farmers diagnose crop diseases more quickly and accurately.The benefit of the entire system is real-time processing by a mobile device, which can function efficiently and accurately despite fluctuating lighting conditions, complicated backdrop and surrounding area such as other leaves, dirt, leaf orientation, camera angle, or picture quality.

- Airborne hyperspectral imaging of nitrogen deficiency on crop traits and yield of maize by machine learning and radiative transfer modeling by S,Wang

  Airborne hyperspectral imaging is a remote sensing technique that captures high-resolution images of the Earth's surface in hundreds of narrow and contiguous spectral bands. This allows for the identification and measurement of various surface materials and their properties. In the context of agriculture, hyperspectral imaging can be used to monitor crop health and nutrient levels, including nitrogen.

  Nitrogen is an essential nutrient for plant growth and development, and its deficiency can have a significant impact on crop yield. By using hyperspectral imaging to monitor nitrogen levels in crops, farmers can make informed decisions about fertilizer application and management to optimize crop yield and reduce environmental impact.

  Radiative Transfer Modeling (RTM) is a technique used to simulate the interaction of light with plant canopies. It can be used to model the reflectance of vegetation as a function of its biophysical properties, such as leaf area index (LAI), chlorophyll content, and canopy structure. By combining RTM with data-driven techniques such as Random Forest (RF) and Partial-Least Squares Regression (PLSR), it is possible to accurately estimate crop traits from hyperspectral imagery.

  In the study you mentioned, the researchers aimed to evaluate the combined use of RTM and data-driven techniques for accurately recovering leaf and canopy nitrogen-related photosynthetic variables using aerial imaging spectroscopy. The study also aimed to quantify the relationship between these variables and yield. By achieving these objectives, the researchers hoped to provide a valuable tool for sustainable nitrogen fertilizer management in precision agriculture.

2.3 Conclusion

From the related work, the proof is there that the solution is required and feasible, as components of the project have been used in other projects and proved viable. The solution will be powered by NVidia jetson tx2, an integrated ai engine, and will consume little power. The jetson's hybrid engine will be taught to identify ailments using datasets. We hope that the equipment will be able to categorize ailments, but this will have to be tested.

CHAPTER 3 ANALYSIS

3.1 Information Gathering Tools

Interviews: Conducting interviews with stakeholders, including farmers, greenhouse owners, and experts in the field of plant disease detection, can provide valuable insights into the requirements and design of the system.

Surveys: Surveys can be used to gather information from a large number of stakeholders, including farmers and experts in plant disease detection, and can help identify common themes and patterns in the requirements and design of the system.

Focus groups: Focus groups can be used to gather feedback and insights from a group of farmers or experts in plant disease detection, and can facilitate discussions and brainstorming sessions around the requirements and design of the system.

Observation: Observing farms and greenhouses in real-world settings can provide insights into their needs, preferences, and behaviors, and can inform the design of the system.

Literature review: Conducting a literature review can help identify best practices, existing solutions, and emerging trends in the field of plant disease detection in farms and greenhouses, and can inform the design of the system.

Prototyping: Developing prototypes and conducting user testing can provide valuable feedback on the usability, functionality, and performance of the system, and can inform the iterative design process.

Note that these tools can be used in combination with each other, and the specific tools and methods used will depend on the scope and requirements of the system, as well as the available resources and constraints.

3.2 Description of system

Disease Detection System is a technology-based system designed to recognize and detect diseases in plants using advanced computer vision techniques and convolutional neural networks (CNNs). This system is primarily developed to improve the detection and management of plant diseases in farms and greenhouses.

The Disease Detection System comprises two main components: the disease recognition component and the disease interpretation component. The recognition component is responsible for recognizing and detecting diseases in plants, while the interpretation

component is responsible for providing information about the detected diseases and suggesting possible treatment options.

The recognition component of the system uses a Jetson Nano as the computer of choice, utilizing its CUDA cores to run advanced CNNs such as ResNet50. The system is trained on a large dataset of plant images to enable it to accurately recognize and detect various plant diseases.

The interpretation component of the system uses advanced CNNs to provide information about the detected diseases and suggest possible treatment options. This component can be customized to provide information in different languages depending on the needs of the user.

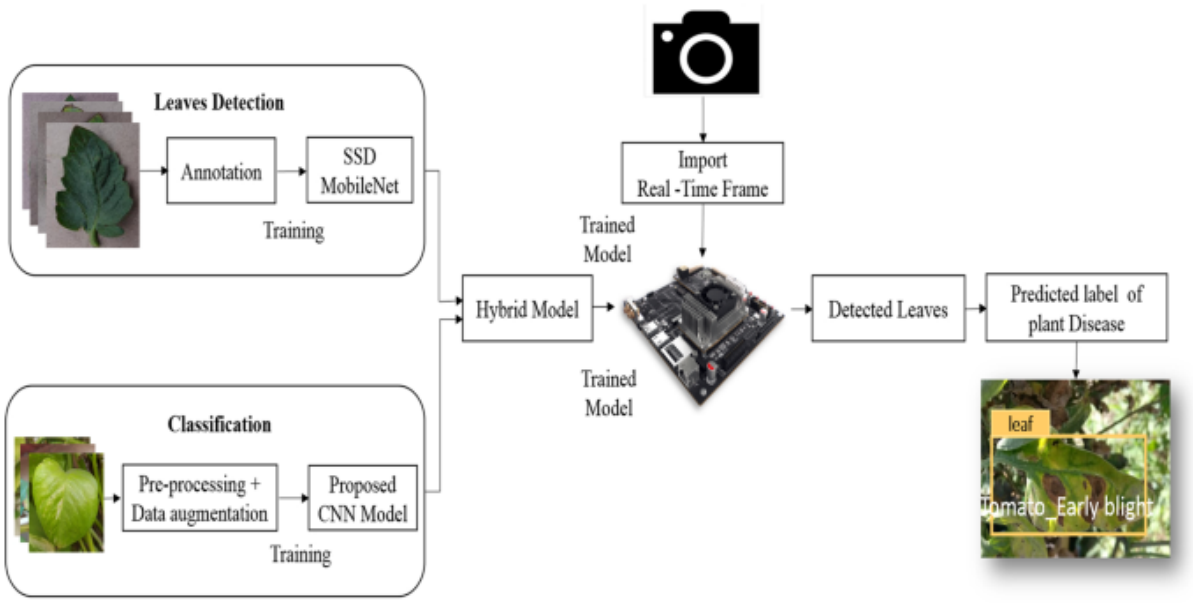


Figure 5.

The Disease Detection System has several applications in various fields, including agriculture and horticulture. In agriculture, this system can be used to improve crop yield by providing early detection and treatment of plant diseases. In horticulture, it can be used to improve the health and quality of plants in greenhouses and gardens.

3.3 Data analysis

UML context diagrams can be used to represent the various components and interactions in the Disease Detection System.

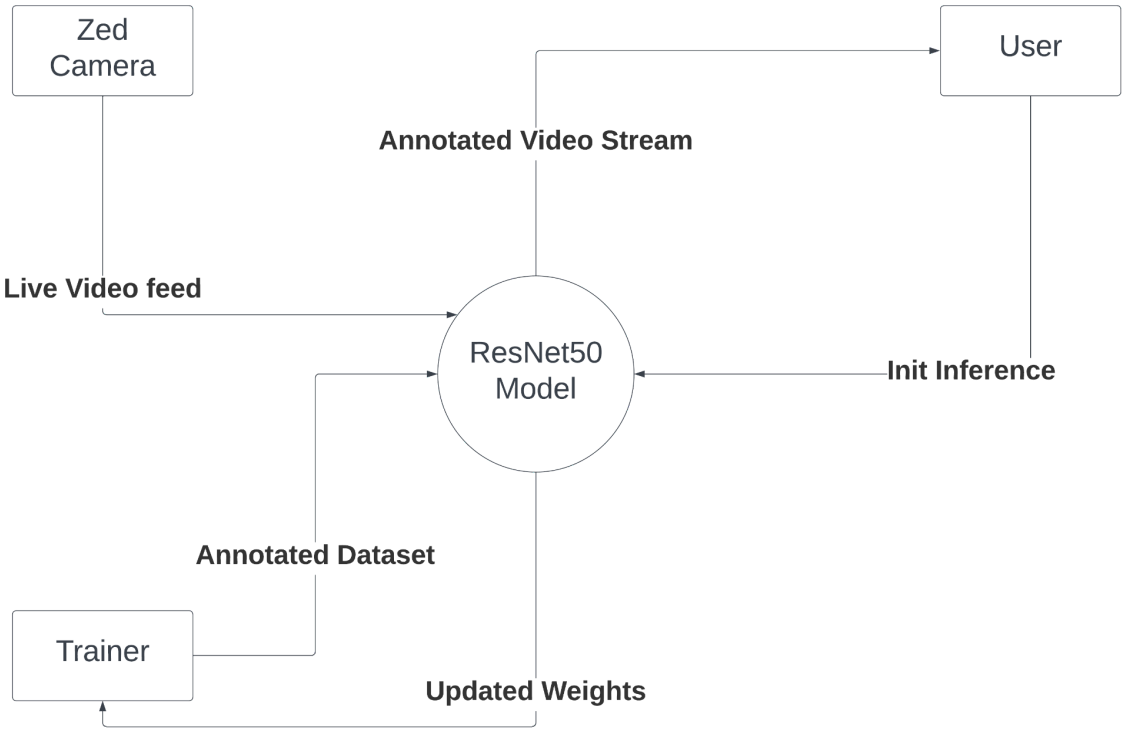


Figure 6.

The context diagram depicts a system that uses a ResNet50 model to analyze and annotate a live video feed of leaves to identify and highlight diseases. The system consists of four main entities: a trainer, a ResNet50 model, a ZED camera, and a user.

The trainer is responsible for providing the ResNet50 model with an annotated dataset of leaves and their diseases. This dataset is used to train the model and update its weights, allowing it to accurately identify and classify diseases in leaves. The training process involves feeding the annotated dataset into the model and adjusting its weights to minimize the error between the predicted and actual classifications.

The ZED camera captures a live video feed of leaves and sends it to the ResNet50 model for analysis. The camera is equipped with advanced sensors that allow it to capture high-quality

video in a variety of lighting conditions. The video feed is sent to the model in real-time, allowing for rapid analysis and annotation.

The ResNet50 model processes the video feed in real-time, using its updated weights to accurately identify and classify diseases in the leaves. The model uses advanced machine learning algorithms to analyze the video feed and make predictions about the presence of diseases in the leaves. These predictions are then used to annotate the video stream, highlighting leaves and their diseases.

The user initiates an interaction with the system and receives an annotated video stream in return. This video stream highlights leaves and their diseases, providing the user with valuable information about the health of the plants. The user can use this information to take appropriate action, such as treating the plants or adjusting their care regimen.

In summary, the context diagram shows how the trainer, ResNet50 model, ZED camera, and user interact with each other to create a system that can accurately identify and classify diseases in leaves using a live video feed. The system leverages advanced machine learning algorithms and high-quality video capture technology to provide users with valuable information about the health of their plants.

3.4 DFD of existing system

The system is one in which a farmer manually identifies diseases in a field. Here's a description of a DFD that shows how this process might work:

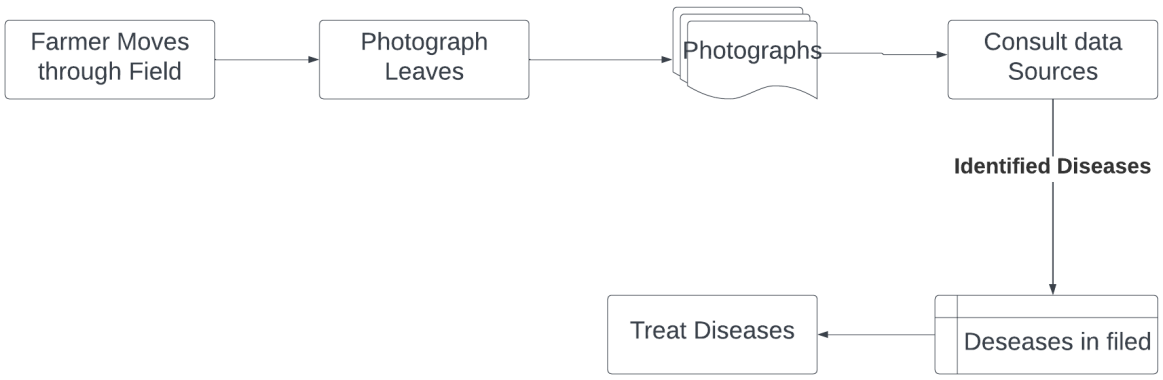


Figure 7.

The DFD would consist of several processes and external entities. The main external entity in this system is the farmer, who initiates the process of identifying diseases in the field.

The first process in the DFD might be the farmer visually inspecting the plants in the field to look for signs of disease. This could involve looking for changes in the colour or texture of the leaves, or the presence of spots or other abnormalities.

If the farmer identifies any potential signs of disease, they would then move on to the second process, which involves diagnosing the disease. This could involve consulting reference materials, such as books or online resources, to determine the type of disease affecting the plants.

Once the farmer has identified the disease, they would move on to the third process, which involves taking appropriate action to treat or prevent the spread of the disease. This could involve applying pesticides or other treatments to the affected plants, or adjusting their care regimen to improve their health.

In summary, the DFD shows how data flows between different processes and entities in a system designed to help a farmer manually identify diseases in a field. The system relies on the farmer's knowledge and expertise to visually inspect the plants and diagnose any diseases present.

3.5 Evaluation of Alternatives Systems

A plant disease detection system that captures images using a ZED camera and processes them using a Jetson Nano is a sophisticated and effective way to identify diseases in plants. However, there are several alternatives to this approach that could also be effective in detecting plant diseases. Some of these alternatives include:

Manual inspection: One alternative to using a camera and computer-based system is for a farmer or trained professional to manually inspect the plants for signs of disease. This approach involves visually examining the plants for changes in color, texture, or shape that may indicate the presence of disease. The individual conducting the inspection would use their knowledge and expertise to identify any potential issues and take appropriate action. While this approach can be effective, it can also be time-consuming and labor-intensive, particularly for large fields.

Drones: Another alternative is to use drones equipped with cameras to capture aerial images of the plants. These images can then be analyzed using computer vision algorithms to identify signs of disease. This approach can cover large areas quickly and provide a bird's-eye view of the plants, making it easier to spot potential issues. However, it requires specialized equipment and expertise to operate the drones and analyze the images.

Satellite imagery: Satellite imagery can also be used to detect signs of disease in plants. This approach involves analyzing images captured by satellites to identify changes in the color or texture of the plants that may indicate the presence of disease. Satellite imagery can cover large areas and provide high-resolution images, making it a powerful tool for detecting plant diseases. However, it can be expensive and may not be available in all areas.

Sensors: Another alternative is to use sensors placed in the field to monitor the health of the plants. These sensors can measure various parameters, such as soil moisture or temperature, that can provide an early warning of potential disease outbreaks. Sensors can provide real-time data and can be configured to send alerts when certain thresholds are exceeded. However, they require regular maintenance and calibration to ensure accurate readings.

In summary, there are several alternatives to using a ZED camera and Jetson Nano-based system for detecting plant diseases. Each approach has its own strengths and weaknesses, and the best approach will depend on factors such as the size of the field, the type of plants being grown, and the resources available to the farmer.

3.6 Functional Analysis of Proposed System-Functional

A plant disease detection system that captures images using a ZED camera and processes them using a Jetson Nano is designed to accurately and efficiently identify diseases in plants. Here's a functional analysis of how such a system might work:

Image capture: The first step in the process is to capture images of the plants using a ZED camera. This camera is equipped with advanced sensors that allow it to capture high-quality images in a variety of lighting conditions. The camera can be mounted on a fixed or mobile platform to capture images of the plants from different angles.

Image processing: Once the images have been captured, they are sent to a Jetson Nano for processing. The Jetson Nano is a small, powerful computer that is capable of running advanced machine learning algorithms. These algorithms are used to analyze the captured images and identify any signs of disease in the plants.

Disease identification: The machine learning algorithms used by the Jetson Nano are trained on a large dataset of images of plants with and without diseases. This allows the algorithms to accurately identify and classify diseases in the captured images. The results of this analysis are then sent back to the user for review.

Action: Once the user has reviewed the results of the analysis, they can take appropriate action to treat or prevent the spread of any identified diseases. This could involve applying pesticides or other treatments to the affected plants, or adjusting their care regimen to improve their health.

In summary, a plant disease detection system that captures images using a ZED camera and processes them using a Jetson Nano is designed to accurately and efficiently identify diseases in plants. The system leverages advanced imaging and machine learning technologies to provide users with valuable information about the health of their crops.

3.7 Non-functional Requirements

A non-functional analysis of a plant disease detection system that captures images using a ZED camera and processes them using a Jetson Nano would focus on the system's non-functional requirements, such as performance, reliability, and usability. Here are some key non-functional aspects to consider when evaluating such a system:

Performance: The performance of the system is a critical factor to consider. The system should be able to capture and process images quickly and accurately, allowing users to rapidly identify and respond to any diseases present in their crops. Factors that can affect performance include the quality of the camera and the processing power of the Jetson Nano.

Reliability: The reliability of the system is also important. The system should be able to operate consistently and accurately over time, providing users with reliable information about the health of their crops. Factors that can affect reliability include the quality of the components used in the system and the robustness of the algorithms used to analyze the images.

Usability: The usability of the system is another key factor to consider. The system should be easy to use and understand, allowing users to quickly and easily capture and analyze images of their crops. Factors that can affect usability include the design of the user interface and the availability of documentation and support.

Scalability: The scalability of the system is also important, particularly for larger farms or agricultural operations. The system should be able to scale up to handle larger numbers of images or more complex analyses as needed. Factors that can affect scalability include the processing power of the Jetson Nano and the design of the system architecture.

In summary, a non-functional analysis of a plant disease detection system that captures images using a ZED camera and processes them using a Jetson Nano would focus on key non-functional requirements such as performance, reliability, usability, and scalability. These factors are important to consider when evaluating the overall effectiveness and suitability of such a system.

3.8 User Case Diagram

The main actor in this system is the user, who interacts with the system to identify diseases in their crops.

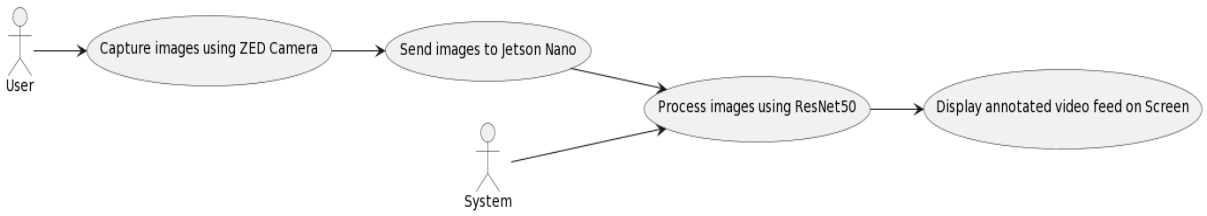


Figure 8.

One use case in the diagram might be “Capture Images.” This use case represents the user capturing images of their crops using the ZED camera. The user would initiate this use case by activating the camera and capturing images of their plants.

Another use case in the diagram might be “Analyze Images.” This use case represents the system analyzing the captured images using the Jetson Nano. The Jetson Nano would use its machine learning algorithms to identify any signs of disease in the captured images.

A third use case in the diagram might be “View Results.” This use case represents the user viewing the results of the analysis to identify any diseases present in their crops. The user would initiate this use case by accessing the results of the analysis through a user interface.

In summary, a use case diagram for a plant disease detection system that captures images using a ZED camera and processes them using a Jetson Nano would show how users interact with the system to capture and analyze images of their crops and view the results of the analysis.

CHAPTER 4 Design

4.1 Systems Diagrams

A data flow diagram (DFD) is a graphical representation of the flow of data in a system. It shows how data is input, processed, and output within the system. Here is a data flow diagram for the system:

The first step in the process is the capture of images by the ZED camera. These images are then sent to the Jetson Nano for processing.
The Jetson Nano receives the images and uses its CUDA cores to run the ResNet50 algorithm. This algorithm processes the images and identifies leaves in them. The processed images are then annotated with information about the identified leaves.
The annotated images are then sent to an attached screen for display. The user can view the annotated video feed on the screen in real-time. This allows them to monitor the health of plants in their environment.

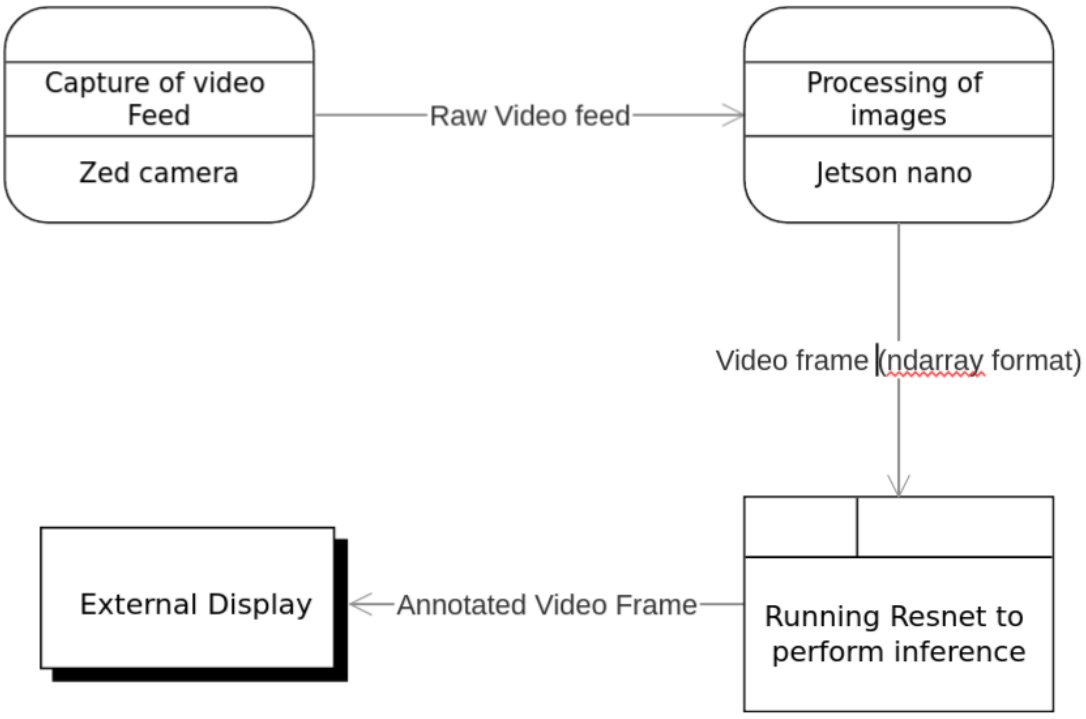


Figure 9.

The activity diagram below describes the flow of actions in the system . The ZED camera captures images of the environment and sends them to the Jetson Nano for processing. The Jetson Nano uses its CUDA cores to run the ResNet50 algorithm and identify leaves in the images. The annotated video feed is then returned and displayed on an attached screen for the user to view. This system can be used in various applications, such as in agriculture or horticulture, to identify and monitor the health of plants in real-time.

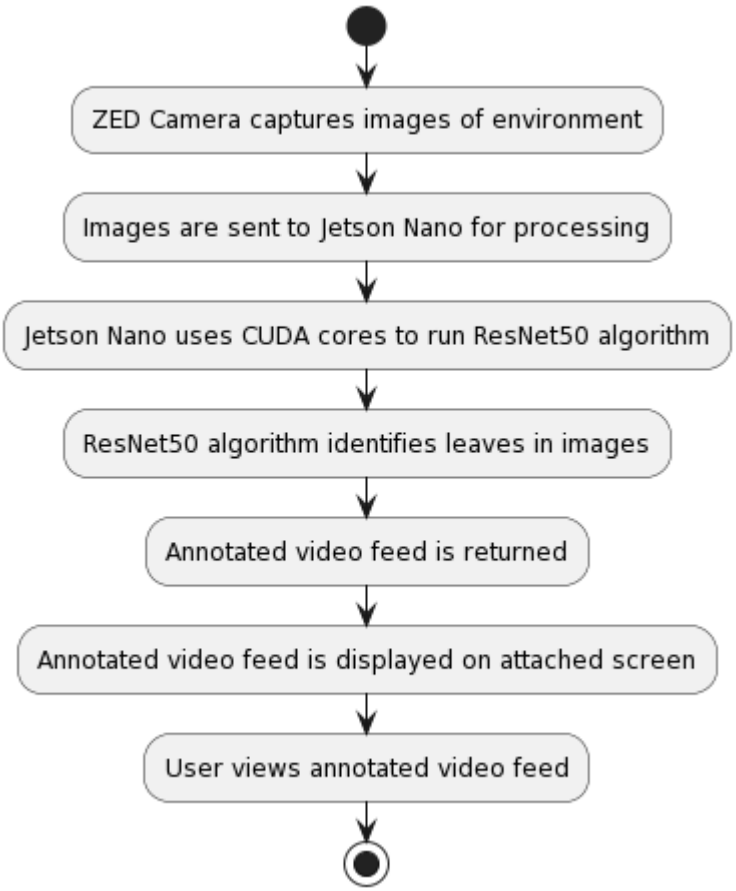


Figure 10.

4.2 Architectural Design

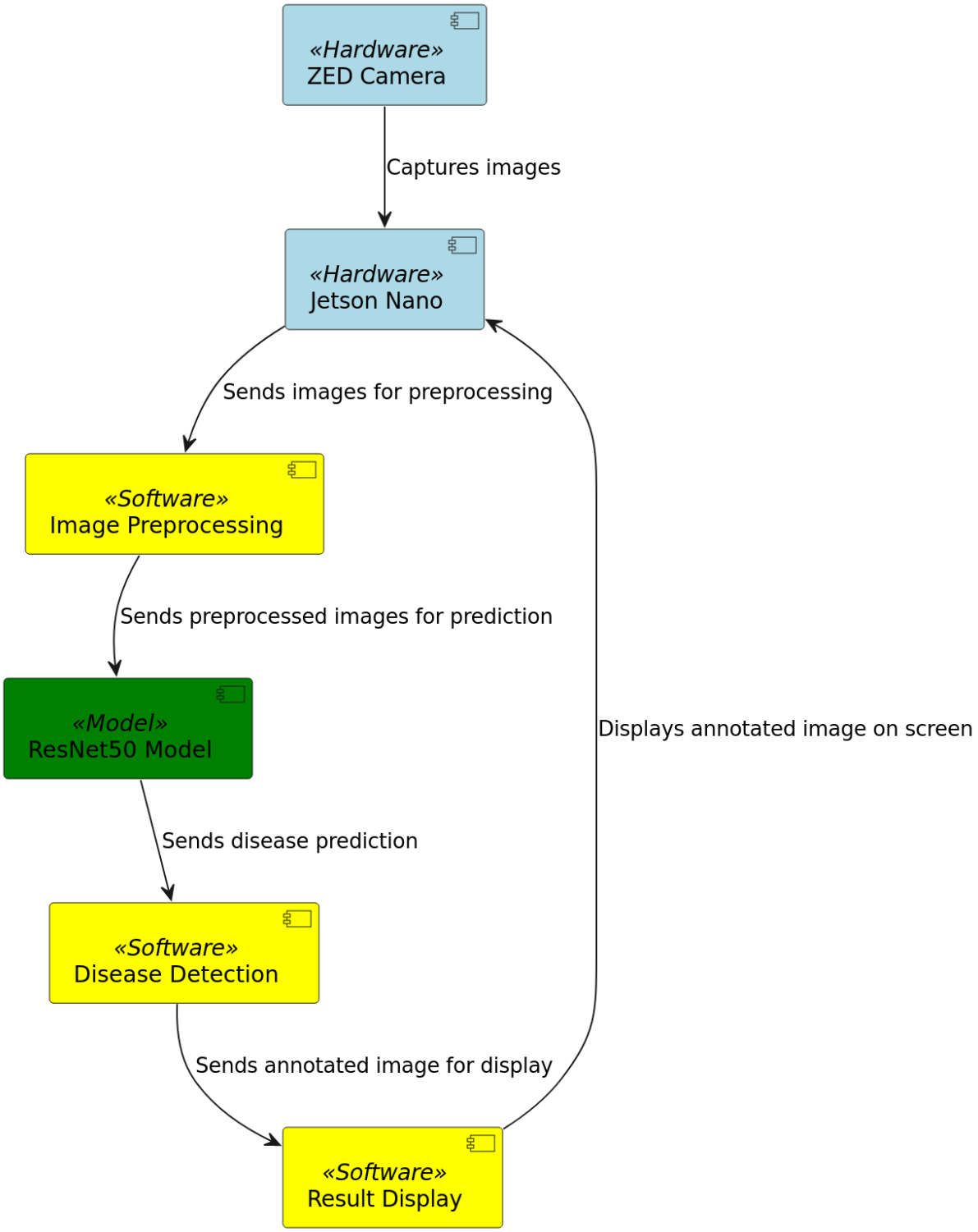


Figure 11.

Data Acquisition: The first step in the system is to acquire data in the form of images of plants. This is done using a ZED camera, which is a stereo camera capable of capturing high-resolution images and depth information. The camera is connected to a Jetson Nano, which is a small and powerful computer designed for embedded systems. The Jetson Nano receives the images from the camera and stores them in memory for processing.

Image Preprocessing: Once the images have been acquired, they need to be preprocessed before being input into the ResNet50 model. This step involves several image processing techniques such as resizing the images to the required dimensions, normalizing the pixel values, and applying data augmentation techniques such as rotation and flipping. These preprocessing steps help to improve the performance of the ResNet50 model by providing it with consistent and varied input data.

Disease Detection: After preprocessing, the images are input into the ResNet50 model. ResNet50 is a deep learning model that has been trained to recognize plant diseases based on their visual characteristics. The model processes the images and outputs a prediction of the disease present in each image. This prediction is based on the probability of each disease being present in the image.

Result Display: Once the disease prediction has been made, the Jetson Nano uses the OpenCV library to write the predicted disease and any other relevant information onto the captured image. This annotated image is then displayed on a screen for the user to see. The user can use this information to take appropriate action to treat the detected disease.

This system architecture allows for real-time detection of plant diseases using a combination of powerful hardware (Jetson Nano and ZED camera) and advanced machine learning techniques (ResNet50 model). By using this system, users can quickly and accurately detect plant diseases and take appropriate action to prevent their spread.

4.3 Program Design

A sequence diagram shows the interactions between objects in the system in the order in which they occur. It is used to model the flow of messages and events between objects in a system over time:

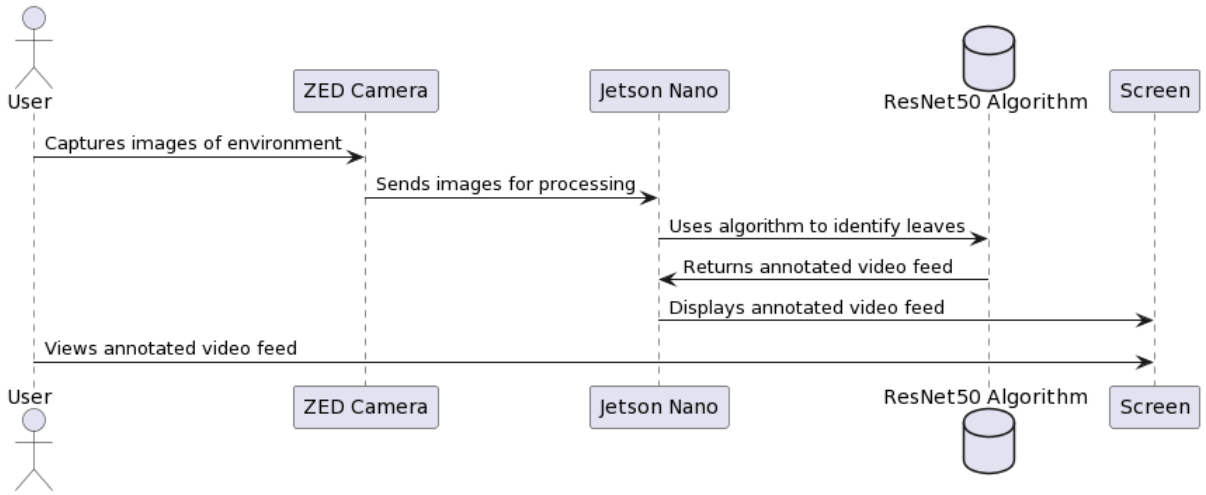


Figure 12.

The class diagram below describes the LeafDetection class and its relationships with other classes. The LeafDetection class has three instance variables: resnet50_model, zed_cam, and leaf_image. The resnet50_model variable is an instance of the keras.Model class and represents a ResNet50 model object that is loaded with weights. The zed_cam variable is an instance of the pyzed.Camera class and represents a ZED camera object used to capture images. The leaf_image variable is an instance of the PIL.Image class and represents a leaf image object used to manipulate the picture before processing.

The LeafDetection class also has three methods: capture_image(), process_image(), and write_data_on_frame(). These methods are used to capture an image using the ZED camera, process the image using the ResNet50 model, and write data on the frames using the OpenCV library, respectively.

The diagram also shows that the LeafDetection class uses four other classes: keras.Model, pyzed.Camera, PIL.Image, and cv2. These classes provide functionality for loading model weights, capturing images, manipulating images, and writing data on frames, respectively.

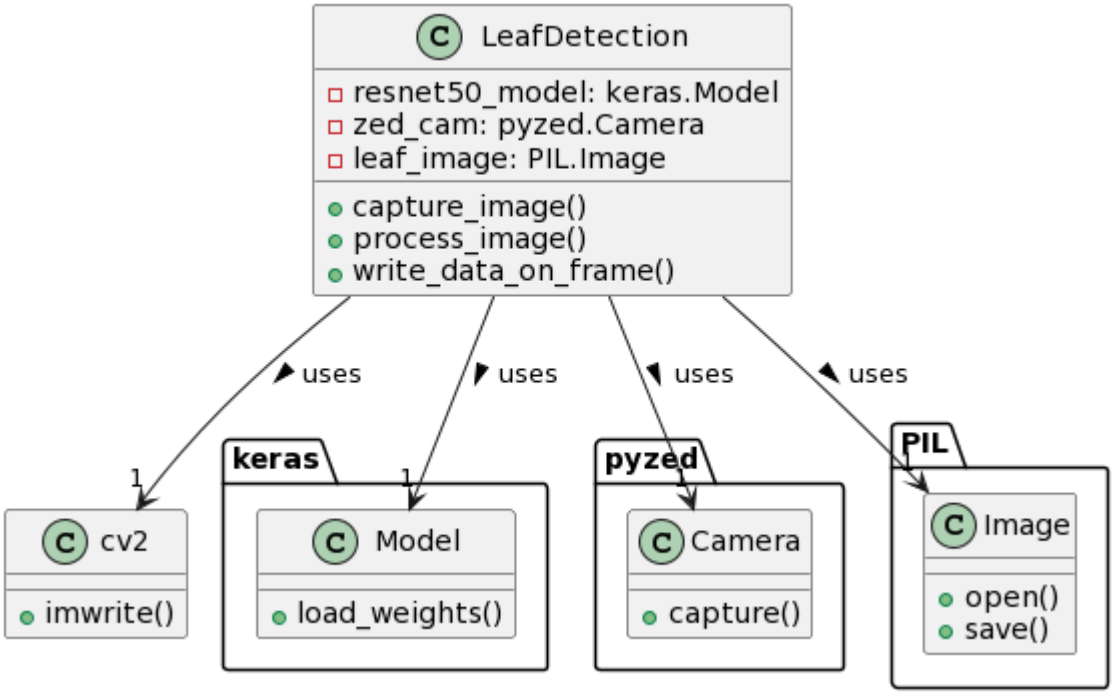


Figure 13.

The package diagram provided below represents the software algorithms used by the system to preprocess images, detect diseases, and display results. The Model package contains the ResNet50 Model component, which represents the machine learning model used by the system to predict diseases based on visual characteristics of plant images.

The arrows between the components in the diagram show the dependencies between those components. For example, the arrow from the ZED Camera component to the Jetson Nano component indicates that the ZED Camera captures images and sends them to the Jetson Nano for processing. Similarly, the arrow from the Image Preprocessing component to the ResNet50 Model component indicates that preprocessed images are sent from the Image Preprocessing component to the ResNet50 Model component for disease prediction.

Overall, this package diagram provides a high-level overview of the organization of the plant disease detection system into packages and shows how data flows between those packages during operation.

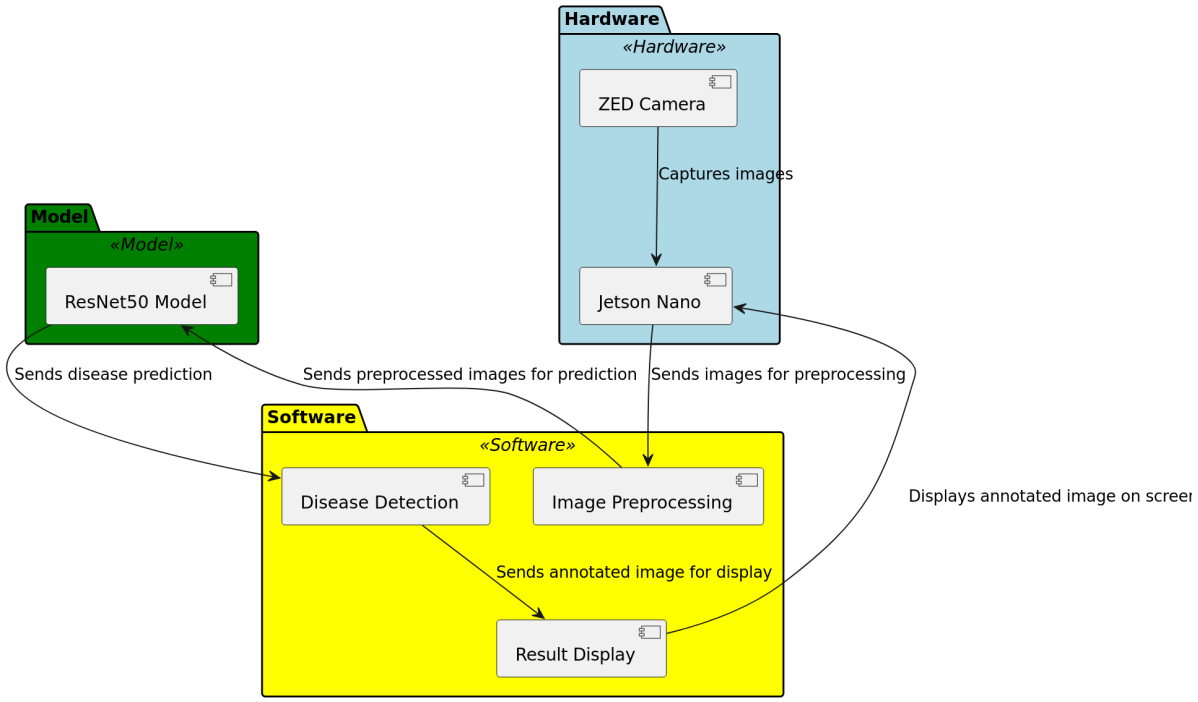


Figure 14.

4.4 Pseudo code

Load the ResNet50 model with pre-trained weights

model = load_resnet50_model()

Initialize the ZED camera

zed_camera = initialize_zed_camera()

Continuously capture images and detect diseases

while True:

    # Capture an image using the ZED camera

    image = zed_camera.capture_image()

    # Preprocess the image

    # This step involves several image processing techniques such as resizing the image to the required dimensions,

    # normalizing the pixel values, and applying data augmentation techniques such as rotation and flipping.

    preprocessed_image = preprocess_image(image)

Use the ResNet50 model to predict the disease present in the image

The model processes the preprocessed image and outputs a prediction of the disease present in the image.

This prediction is based on the probability of each disease being present in the image.

disease_prediction = model.predict(preprocessed_image)

Annotate the image with the predicted disease and any other relevant information

This step involves writing text onto the captured image to indicate the predicted disease and any other relevant information.

annotated_image = annotate_image(image, disease_prediction)

Display the annotated image on a screen

This step involves sending the annotated image to a screen for display so that the user can see the predicted disease and take appropriate action.

display_image(annotated_image)

4.5 Interface Design

This user interface provides a real-time view of the images captured by the left and right cameras of the ZED stereo camera. The images are displayed in two panels, one for each camera, and are updated continuously as new images are captured. On top of these images, the user interface displays rectangles representing the detection area. These rectangles show the region of the image where the plant disease detection algorithm is being applied and can be adjusted by the user to focus on specific areas of interest.

In addition to the rectangles representing the detection area, the user interface also displays annotations on top of the images. These annotations provide information about the predicted disease and any other relevant information. For example, if the plant disease detection algorithm predicts that a particular region of the image contains a diseased leaf, an annotation could be displayed on top of that region indicating the predicted disease and its probability.

Overall, this user interface provides a clear and intuitive way for users to interact with the plant disease detection system and to visualize its results in real-time. By displaying real-time

images from the ZED camera and providing annotations with relevant information, this user interface allows users to quickly and accurately detect plant diseases and take appropriate action to prevent their spread.

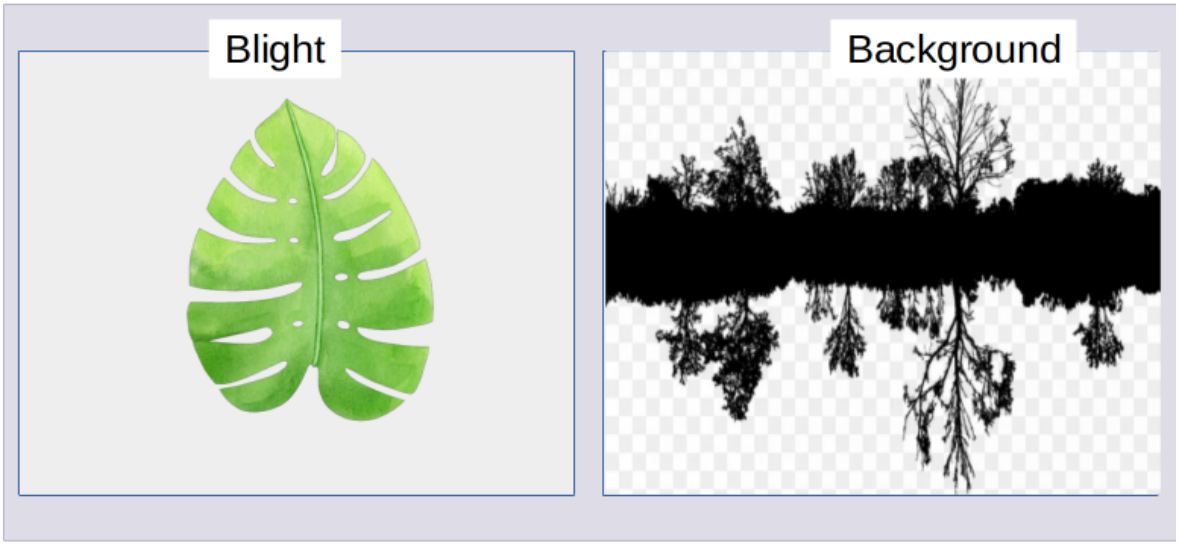


Figure 15.

Draft

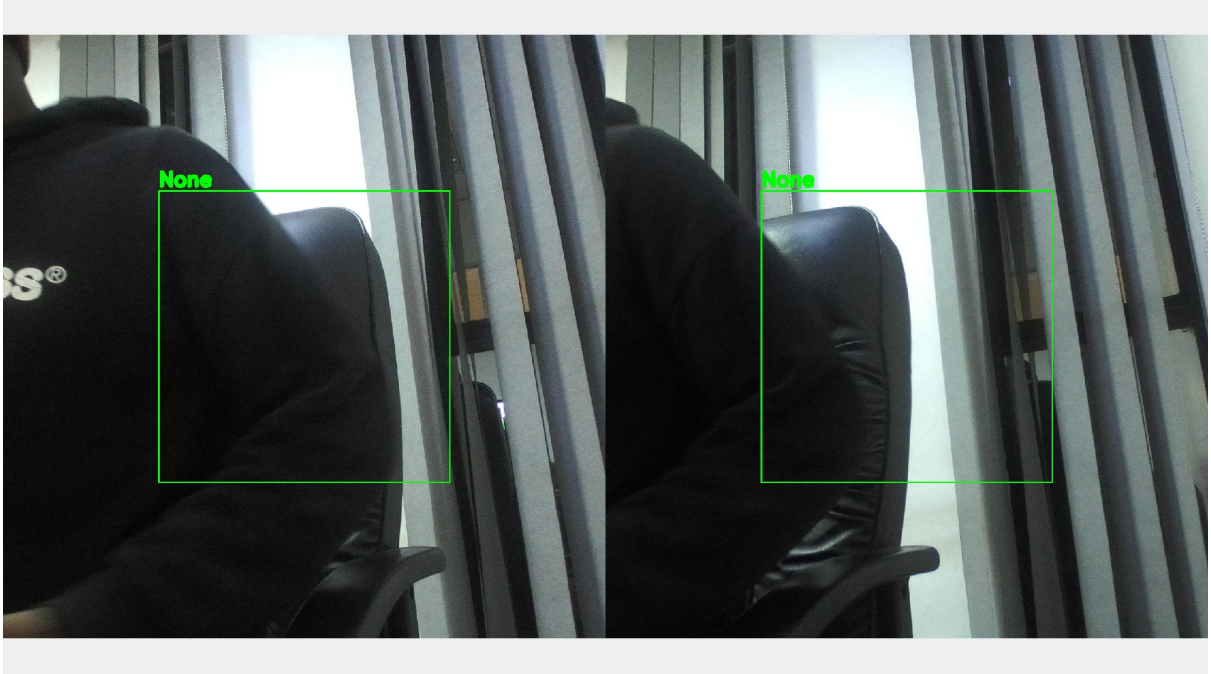
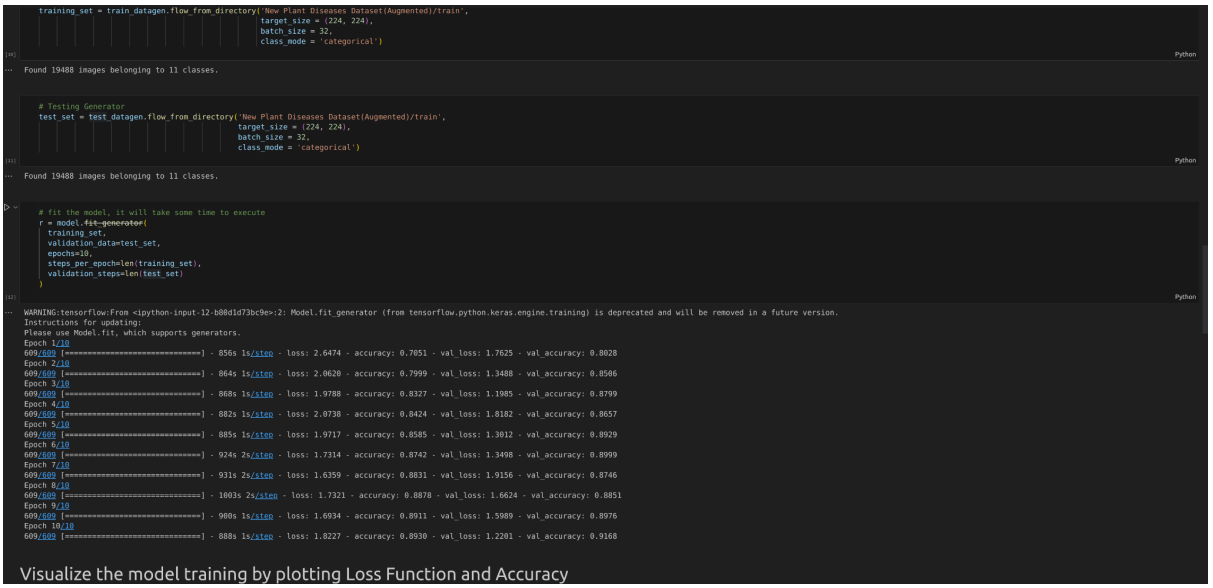


Figure 16.

Actual

Training Dashboard



```
training_set = train_datagen.flow_from_directory('New Plant Diseases Dataset(Augmented)/train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

Python

```
Found 19488 images belonging to 11 classes.
```

```
# Testing Generator
test_set = test_datagen.flow_from_directory('New Plant Diseases Dataset(Augmented)/train',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

Python

```
Found 19488 images belonging to 11 classes.
```

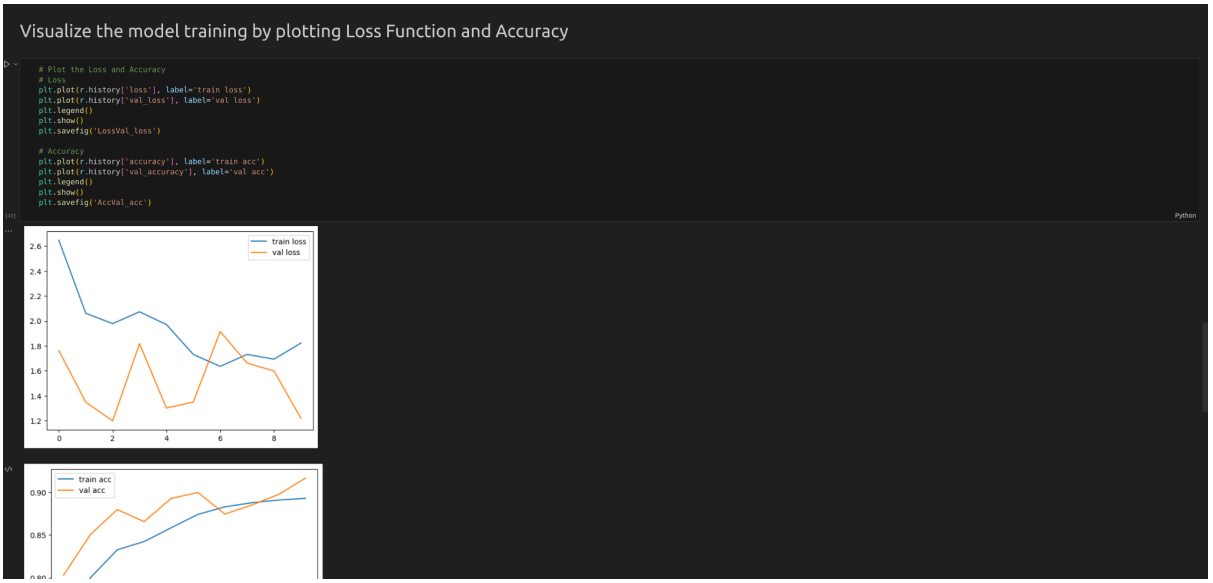
```
# fit the model, it will take some time to execute
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=10,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

Python

```
WARNING:tensorflow:From <ipython-input-12-b80d1d73bc9e>:2: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/10
609/609 [==============================] - 856s 1s/step - loss: 2.6474 - accuracy: 0.7051 - val_loss: 1.7625 - val_accuracy: 0.8028
Epoch 2/10
609/609 [==============================] - 864s 1s/step - loss: 2.0620 - accuracy: 0.7999 - val_loss: 1.3488 - val_accuracy: 0.8506
Epoch 3/10
609/609 [==============================] - 868s 1s/step - loss: 1.9788 - accuracy: 0.8327 - val_loss: 1.1985 - val_accuracy: 0.8799
Epoch 4/10
609/609 [==============================] - 882s 1s/step - loss: 2.0738 - accuracy: 0.8424 - val_loss: 1.8182 - val_accuracy: 0.8657
Epoch 5/10
609/609 [==============================] - 885s 1s/step - loss: 1.9717 - accuracy: 0.8585 - val_loss: 1.3012 - val_accuracy: 0.8929
Epoch 6/10
609/609 [==============================] - 924s 2s/step - loss: 1.7314 - accuracy: 0.8742 - val_loss: 1.3498 - val_accuracy: 0.8999
Epoch 7/10
609/609 [==============================] - 931s 2s/step - loss: 1.6359 - accuracy: 0.8831 - val_loss: 1.9156 - val_accuracy: 0.8746
Epoch 8/10
609/609 [==============================] - 1003s 2s/step - loss: 1.7321 - accuracy: 0.8878 - val_loss: 1.6624 - val_accuracy: 0.8851
Epoch 9/10
609/609 [==============================] - 900s 1s/step - loss: 1.6934 - accuracy: 0.8911 - val_loss: 1.5989 - val_accuracy: 0.8976
Epoch 10/10
609/609 [==============================] - 888s 1s/step - loss: 1.8227 - accuracy: 0.8930 - val_loss: 1.2201 - val_accuracy: 0.9168
```

Visualize the model training by plotting Loss Function and Accuracy

Figure 17.



Visualize the model training by plotting Loss Function and Accuracy

```
# Plot the Loss and Accuracy
# Loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# Accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```

Python

Figure 18.

CHAPTER 5 ANALYSIS

5.1 Introduction

The main objective of the implementation phase is to make sure that the developed system fulfills all the goals mentioned in the proposal. The system is tested as a whole to check if it meets the specified requirements. System testing evaluates the software quality based on various criteria such as reliability and usability. Testing is done throughout the system development process. The system should meet all its proposed goals and function as described in the proposal. It is tested for both functional and non-functional requirements. This section discusses how the system was tested after its development and describes the training procedures for its implementation. An evaluation is also conducted to check if the system achieves its stated objectives.

5.2 Code Samples

Training Model

```
from collections import OrderedDict
import time
from torch.optim import lr_scheduler
import copy
#============================#
#pytorch imports             #
#============================#
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.models import ResNet18_Weights
from data import *
model = models.resnet50(weights=ResNet50_Weights.DEFAULT)
# Freeze parameters so we don't backprop through them

for param in model.parameters():
  param.requires_grad = False



from collections import OrderedDict
```

```
# Creating the classifier ordered dictionary first

classifier = nn.Sequential(OrderedDict([
                          ('fc1', nn.Linear(1024, 512)),
                          ('relu', nn.ReLU()),
                          ('fc2', nn.Linear(512, 39)),
                          ('output', nn.LogSoftmax(dim=1))
                          ]))

# Replacing the pretrained model classifier with our classifier
model.fc = classifier

#Function to train the model
def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
  since = time.time()

  best_model_wts = copy.deepcopy(model.state_dict())
  best_acc = 0.0

  for epoch in range(1, num_epochs+1):
    print('Epoch {}/{}'.format(epoch, num_epochs))
    print('-' * 10)

    # Each epoch has a training and validation phase
    for phase in ['train', 'val']:
      if phase == 'train':
        scheduler.step()
        model.train()  # Set model to training mode
      else:
        model.eval()   # Set model to evaluate mode

      running_loss = 0.0
      running_corrects = 0

      # Iterate over data.
      for inputs, labels in dataloaders[phase]:
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
```

```
            optimizer.zero_grad()

            # forward
            # track history if only in train
            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                _, preds = torch.max(outputs, 1)

                # backward + optimize only if in training phase
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            # statistics
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]

        print('{} Loss: {:.4f} Acc: {:.4f}'.format(
            phase, epoch_loss, epoch_acc))

        # deep copy the model
        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())

    print()

time_elapsed = time.time() - since
print('Training complete in {:.0f}m {:.0f}s'.format(
    time_elapsed // 60, time_elapsed % 60))
print('Best valid accuracy: {:4f}'.format(best_acc))

# load best model weights
model.load_state_dict(best_model_wts)
return model

# Train a model with a pre-trained network
```

```
num_epochs = 30
if use_gpu:
  print ("Using GPU: "+ str(use_gpu))
  model = model.cuda()

# NLLLoss because our output is LogSoftmax
criterion = nn.NLLLoss()

# Adam optimizer with a learning rate
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)
# Decay LR by a factor of 0.1 every 5 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)


model_ft = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=30)

# Save the checkpoint

model.class_to_idx = dataloaders['train'].dataset.class_to_idx
model.epochs = num_epochs
checkpoint = {'input_size': [3, 224, 224],
              'batch_size': dataloaders['train'].batch_size,
               'output_size': 39,
               'state_dict': model.state_dict(),
               'data_transforms': data_transforms,
               'optimizer_dict':optimizer.state_dict(),
               'class_to_idx': model.class_to_idx,
               'epoch': model.epochs}
torch.save(checkpoint, 'restomato_checkpoint.pth')
```

Table 2.

Loading model for inference, along with camera vision and image processing

```
#import pyzed.sl as sl
from collections import OrderedDict
#from datetime import datetime
from torch.autograd import Variable
import time
```

```
from PIL import Image
#import matplotlib.pyplot as plt
import copy
import cv2
import pyzed.sl as sl
import numpy as np
import os

#================================#
#pytorch imports              #
#================================#
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import models, datasets, transforms
from torchvision.models import ResNet18_Weights

#from train import Model as plantmodel

class ObjectDetection:
  """
  The class performs generic object detection on a video file.
  It uses yolo5 pretrained model to make inferences and opencv2 to manage frames.
  Included Features:
  1. Reading and writing of video file using  Opencv2
  2. Using pretrained model to make inferences on frames.
  3. Use the inferences to plot boxes on objects along with labels.
  Upcoming Features:
  """
  def __init__(self):
    """
    :param input_file: provide youtube url which will act as input for the model.
    :param out_file: name of a existing file, or a new file in which to write the output.
    :return: void
    """
    self.model, self.class_to_idx = self.load_model('res18tomato_checkpoint.pth')

    self.idx_to_class = { v : k for k,v in self.class_to_idx.items()}
    #self.model.conf = 0.4 # set inference threshold at 0.3
    #self.model.iou = 0.3 # set inference IOU threshold at 0.3
```

```
    #self.model.classes = [0] # set model to only detect "Person" class
    self.device = 'cuda' if torch.cuda.is_available() else 'cpu'

def load_model(self, filepath):

    checkpoint = torch.load(filepath)
    model = models.resnet18()
    classifier = nn.Sequential(OrderedDict([
                          ('fc1', nn.Linear(512, 512)),
                          ('relu', nn.ReLU()),
                          #('dropout1', nn.Dropout(p=0.2)),
                          ('fc2', nn.Linear(512, 39)),
                          ('output', nn.LogSoftmax(dim=1))
                          ]))

    # Replacing the pretrained model classifier with our classifier
    model.fc = classifier
    model.load_state_dict(checkpoint['state_dict'])
    return model, checkpoint['class_to_idx']

def score_frame(self, frame, topk=1):
    """
    function scores each frame of the video and returns results.
    :param frame: frame to be infered.
    :return: labels and coordinates of objects found.

    """
    pil_image = Image.fromarray(frame).convert('RGB')
    image_np = np.array([self.process_image(pil_image)])

    image = torch.FloatTensor(image_np)
    self.model.eval()
    output = self.model.forward(Variable(image))
    pobabilities = torch.exp(output).data.numpy()[0]

    top_idx = np.argsort(pobabilities)[-topk:][::-1]
    top_class = [self.idx_to_class[x] for x in top_idx]
    top_probability = pobabilities[top_idx]

    return top_probability, top_class
```

```
  def draw_predictions(self):
      # Create a Camera object
      zed = sl.Camera()

      # Create a InitParameters object and set configuration parameters
      init_params = sl.InitParameters()
      init_params.camera_resolution = sl.RESOLUTION.HD2K  # Use HD2K video mode for higher
quality images
      init_params.camera_fps = 60  # Set fps at 60

      # Open the camera
      err = zed.open(init_params)
      if err != sl.ERROR_CODE.SUCCESS:
          exit(1)

      # Create Mat objects for left and right images
      image_left = sl.Mat()
      image_right = sl.Mat()

  # Get screen size
      cv2.namedWindow("ZED", cv2.WND_PROP_FULLSCREEN)
      cv2.setWindowProperty("ZED", cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN)
      screen_width = cv2.getWindowImageRect("ZED")[2]



  # Capture new images until 'q' is pressed
      key = ''
      while key != ord('q'):
              # Grab an image
          if zed.grab() == sl.ERROR_CODE.SUCCESS:
              # Retrieve left and right images
              zed.retrieve_image(image_left, sl.VIEW.LEFT)
              zed.retrieve_image(image_right, sl.VIEW.RIGHT)
              # Combine left and right images horizontally
              combined_image = np.hstack((self.label_frame(image_left.get_data()),
self.label_frame(image_right.get_data())))

              cv2.imshow("ZED", combined_image)
              key = cv2.waitKey(1)
```

```
    # Close the camera and destroy window
    zed.close()
    cv2.destroyAllWindows()

  def label_frame(self,frame):

    # Preprocess the frame for AI inference

     # Perform AI inference on the frame
        prediction = self.score_frame(frame)

    # Draw boxes on regions of interest with some text
        print(prediction[0][0])
        frame_height, frame_width = frame.shape[:2]
        rect_width, rect_height = 600, 600

        x1 = (frame_width - rect_width) // 2
        y1 = (frame_height - rect_height) // 2
        x2 = x1 + rect_width
        y2 = y1 + rect_height

        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        print(prediction)
        try:
            if prediction[0][0]>0.6:
                label=prediction[1][0]

            else:
                label='unsure'
        except Exception as e:
            pass
        accuracy=int(prediction[0][0]*100)
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(frame, f'{label}: {accuracy}%', (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX,
1.4, (0, 255, 0), 8)

       # Display the resulting frame
        return frame
```

```
  def process_image(self, image):
      width, height = image.size
      min_dim = min(width, height)
      left = (width - min_dim) // 2
      top = (height - min_dim) // 2
      right = (width + min_dim) // 2
      bottom = (height + min_dim) // 2
      image = image.crop((left, top, right, bottom))
      size = 256, 256
      image.thumbnail(size, Image.Resampling.LANCZOS)
      image = image.crop((128 - 112, 128 - 112, 128 + 112, 128 + 112))
      npImage = np.array(image)
      npImage = npImage/255.

      imgA = npImage[:,:,0]
      imgB = npImage[:,:,1]
      imgC = npImage[:,:,2]

      imgA = (imgA - 0.485)/(0.229)
      imgB = (imgB - 0.456)/(0.224)
      imgC = (imgC - 0.406)/(0.225)

      npImage[:,:,0] = imgA
      npImage[:,:,1] = imgB
      npImage[:,:,2] = imgC

      npImage = np.transpose(npImage, (2,0,1))

      return npImage


  def __call__(self):
      self.draw_predictions()


a = ObjectDetection()
a()
```

Table 3.

5.3 Software Testing

White Box: [Unit test]

Unit testing is an important part of the software development process. It involves testing individual modules or components of a system to ensure that they function as expected. For a disease detection system, this could involve testing the accuracy of the diagnostic algorithm or the functionality of the data processing engine.

Test cases:

Unit test	Open Camera
Test Case	Run code without camera
Expected result	Error throw from python env
Actual result	'CAMERA NOT FOUND'

Table 4.

Unit test	Break Video into frames
Test Case	Push image instead of frame
Expected result	Error from pillow
Actual result	ValueError: may indicate binary incompatibility. Expected 96 from C header, got 80 from PyObject

Table 5.

Unit test	Stream Video
Test Case	Disconnect all displays
Expected result	Error code from python egl
Actual result	Message "Could not get EGL display connection"

Table 6.

Unit test	Resnet inference engine
Test Case	Download picture of know disease, and use flask appp to feed image into system to test performance
Expected result	Tomato_Yellow_Leaf_Curl_Virus
Actual result	Tomato_Yellow_Leaf_Curl_Virus

Table 7.

Black box testing

Black box testing is a software testing technique in which the functionality of the software under test (SUT) is tested without looking at the internal code structure, implementation details, or knowledge of the software's internal paths. In other words, it does not place much emphasis on the system's internal design or mechanism, but rather on the degree of the output produced in relation to the input. This is the technique that is most commonly used for validation.

With respect to black box testing a disease detection system, it could be tested for the following:

- Incorrect or missing functions
- Behavior or performance errors
- Interface errors

Test Case	Try running the system on a screen image of a diseased leaf
Expected result	Septoria_leaf_spot
Actual result	'Background 87%' [failed probably due to reflective surface]

Table 8.

Test Case	Run 5 inferences on leaves with known diseases
Expected result	100% correctness
Actual result	80% correctness

Table 9.

Test Case	Open A background without leaves
Expected result	No results from Interface
Actual result	'Background 96%'

Table 10.

CHAPTER 6 Conclusions and Recommendations

6.1 Results and summary

In this project, a system was designed to detect plant diseases using a Jetson Nano to process images captured by a ZED camera. The goal of the project was to develop an accurate and efficient system for detecting plant diseases in real-time.

Initially, image processing was done using the ResNet152 model. However, it was found that the system was slow in processing the images. To improve the speed of the system, the model was switched to ResNet18. While this improved the speed of the system, the accuracy was not satisfactory.

To improve the accuracy of the system, several attempts were made. A CNN with fewer layers was used and the model was trained with more epochs. However, these attempts did not result in a significant improvement in accuracy. The InceptionV3 model was also tried but it did not improve the accuracy of the system.

It was found that the accuracy of the detection was heavily affected by lighting. When images from the dataset were used to test the inference engine, an accuracy of 95% was achieved. However, when using images captured by the ZED camera, the accuracy dropped to less than 70%.

In addition to changes in the image processing model, changes were also made to the hardware components of the system. The ZED camera was initially powered by OpenCV but was later switched to the ZED camera library for better graphics.

After trying several different models, including ResNet18 which was found to be inaccurate, ResNet50 was ultimately chosen as it provided a good balance between speed and accuracy.

Overall, while some progress was made in improving the speed and accuracy of the system, challenges were faced due to factors such as lighting. Further research and development may be needed to achieve a satisfactory level of accuracy in detecting plant diseases using this system.

6.2 Scope of Future Work & Recommendations

Generative Adversarial Network (GAN): GANs can generate additional training data for the model, which can help address the issue of limited training data and improve the accuracy of the model. In one study, a Wasserstein GAN with gradient penalty (WGAN-GP) was combined with label smoothing regularization (LSR) to improve the prediction accuracy and address the overfitting problem under limited training data. The results showed that this approach improved the overall classification accuracy of plant diseases by 24.4% as compared to 20.2% using classic data augmentation and 22% using synthetic samples without LSR.

The explicit control of hidden classes when training the model: This involves training and validating a deep learning-based detector using target and control classes on images collected in various greenhouses. The generated features are then applied for testing the inference of the system on data from new greenhouse conditions where the goal is to detect target classes exclusively. By having explicit control over inter- and intra-class variations, the model can distinguish data variations that make the system more robust when applied to new scenarios. In one study, this approach achieved a recognition rate of target classes of 93.37% mean average precision on the inference dataset.

Infrared thermography: this is a thermal imaging technology that can be used to detect plants stressed by biotic and abiotic factors. Temperature fluctuations have been linked to pathogen attacks in various instances, and infrared thermography can allow for early and quick disease measurement. This technique is non-destructive, non-invasive, and non-contact, and can be used to determine the thermal characteristics of any material of concern.Infrared thermography has been applied in many agriculture sections, including monitoring greenhouse and nurseries, programming irrigation, estimating crop yield, detecting plant disease, assessing fruit maturity, and detecting bruising in vegetables and fruits.

Increase the diversity of the dataset: Creating or adjusting a dataset to fit the specific situation can be a good approach to improve the accuracy of the plant disease detection system. By collecting and labeling images of plants in the specific environment and conditions that are working with, one can create a dataset that is more representative of the data that the model will encounter in practice. This can help improve the accuracy of the

model by reducing the domain shift between the training data and the data encountered in practice.Creating or adjusting a dataset can indeed be a challenging task. It involves collecting and labeling a large number of images, which can be time-consuming and require expertise in the domain. However, the effort can be worthwhile if it results in improved accuracy of your plant disease detection system. There are also tools and techniques available that can help with the process of creating or adjusting a dataset, such as data augmentation techniques and active learning approaches.

Using Jetson nano with battery pack: The Jetson Nano is a small, powerful computer designed for embedded applications and AI projects. It can be powered using a power package such as a 5V 4A power supply with a barrel jack connector. Here are some points to note on how to use the Jetson Nano efficiently with a power package:

Choose the right power supply: Make sure to choose a power supply that meets the requirements of the Jetson Nano. The recommended power supply is 5V 4A with a barrel jack connector.

- Connect the power supply correctly: Connect the power supply to the Jetson Nano using the barrel jack connector. Make sure to connect the power supply correctly to avoid damaging the device.
- Monitor power usage: The Jetson Nano has an onboard power monitor that can help you track the power usage of the device. Use this feature to monitor the power usage and make sure that the device is operating within its power limits.
- Optimise your code: Make sure to optimise your code to make efficient use of the Jetson Nano’s resources. This can help reduce power consumption and improve performance.
- Use power-saving features: The Jetson Nano has several power-saving features that can help reduce power consumption. Make use of these features to improve the efficiency of your system.