# Final Individual Report
# Enhancing Plant Health with AI

## Using Thermal and Image Processing

By: Alan Palayil

Lab Partner: Zahra Tolideh and Lukas Klicker

Instructor: Dr. Jafar Saniie

ECE 501 (Summer 2023)

Due Date: 06/30/2023

*Acknowledgment:* I acknowledge all the work (including figures and codes) belongs to me and/or the people who are referenced.

*Electronic Signature*: Alan Palayil A20447935 (Due Date: 6/30/2023)

## Contents

## Abstract:

Plant diseases present a significant threat to global food security and agricultural productivity. The timely detection and accurate classification of plant diseases are crucial for effective intervention and management. In this paper, we propose an integrated system for plant disease detection and classification. This system combines deep learning techniques with environmental sensors to provide comprehensive and automated disease assessment. To capture leaf temperature, the system incorporates thermal imaging technology, which allows for the calculation of Vapor Pressure Deficit (VPD). Additionally, an environmental humidity sensor (DHT10) measures the humidity, providing essential data for evaluating disease risks. By integrating these environmental parameters, the system offers a comprehensive approach to disease detection and management.

To accurately classify plant diseases, we employ a deep learning framework based on convolutional neural networks (CNN). The CNN model is trained and fine-tuned using a diverse dataset of plant images, including visible and thermal images. Using a USB camera, we extract frames from the video feed, enabling real-time testing of the AI model for disease detection. The trained model demonstrates high accuracy in identifying and classifying various plant diseases. It effectively distinguishes between healthy and diseased plant conditions, aiding in the early identification of potential risks. Furthermore, the system utilizes VPD calculations to assess the need for irrigation, providing insights into plant water requirements.

The proposed system offers several benefits, including real-time disease detection, non-invasive monitoring, and scalability. By combining deep learning techniques with environmental sensors, it enables efficient disease management practices and reduces reliance on manual inspection. The system can be deployed in various agricultural settings, such as farms, greenhouses, and nurseries, assisting farmers, agronomists, and researchers in mitigating plant diseases and ensuring sustainable crop production.

*Keywords:* Plant disease detection, deep learning, convolutional neural networks, environmental sensors, thermal imaging, Vapor Pressure Deficit (VPD), disease risk assessment, automated system.

## Introduction:

The project's primary focus is the development of an integrated system that utilizes deep learning techniques and environmental sensors for plant disease detection and classification. The main objective is to achieve early detection and accurate classification of plant diseases, thereby facilitating timely intervention and effective disease management. To accomplish this, the system incorporates thermal imaging technology to capture leaf temperature, which plays a vital role in calculating Vapor Pressure Deficit (VPD). The assessment of disease risk and determination of irrigation needs rely on VPD calculations and data obtained from an AHT10 sensor, which measures environmental humidity. By integrating these environmental parameters, the system offers a comprehensive approach to disease detection and management.

For accurate disease classification, the project employs a deep learning framework based on convolutional neural networks (CNN). The CNN model undergoes training and fine-tuning using a diverse dataset of plant images, including both visible and thermal images. Real-time testing of the AI model is made possible through the utilization of a USB camera, which extracts frames from the video feed. The technical aspects of the project involve the development of algorithms to process thermal images, extract relevant features, and train the CNN model for disease classification. Integration with environmental sensors encompasses the acquisition and processing of data from the AHT10 sensor to assess disease risk based on VPD calculations. Additionally, the system incorporates real-time video processing and testing using the USB camera to validate the AI model's performance.

The overall aim of the project is to provide a comprehensive and automated solution for plant disease detection and classification. By combining deep learning techniques with environmental sensors, the system offers real-time disease monitoring, enables proactive disease management, and reduces reliance on manual inspection. The project's technical components encompass image processing, machine learning, sensor integration, and real-time video analysis, all contributing to a robust and scalable solution for plant disease management.

## Background:

Traditionally, plant disease detection has relied on manual inspection and visual examination by experts, which can be time-consuming, labor-intensive, and prone to human error. As agriculture continues to evolve and embrace technological advancements, there is a growing need for automated systems that can provide rapid and reliable disease detection, enabling early intervention and minimizing the risk of crop damage.

In recent years, advancements in artificial intelligence (AI) and deep learning techniques have opened new possibilities for automated plant disease detection. Deep learning, particularly Convolutional Neural Networks (CNNs), has shown remarkable capabilities in image recognition and classification tasks. By leveraging these techniques, researchers and scientists can train AI models to analyze plant images and accurately identify the presence of diseases.

However, disease detection alone is not sufficient for effective disease management. Environmental factors and conditions play a crucial role in disease development and progression. Parameters such as temperature, humidity, and vapor pressure deficit (VPD) are known to influence the growth and spread of plant diseases. Monitoring and integrating these environmental factors into disease detection systems can provide a more comprehensive and accurate assessment of disease risks, enabling farmers to make informed decisions regarding disease control strategies.

Thermal imaging technology has emerged as a valuable tool in plant disease detection and management. By capturing the infrared radiation emitted by plant leaves, thermal cameras can measure leaf temperature, which is an indicator of plant health and stress. Combining thermal imaging with AI algorithms allows for the detection of subtle temperature variations associated with disease symptoms, providing an early indication of disease presence.

Vapor Pressure Deficit (VPD) is another critical parameter in plant disease management. VPD represents the difference between the actual vapor pressure in the air and the saturation vapor pressure at a specific temperature. High VPD values indicate a dry atmosphere, which can increase water stress on plants and create favorable conditions for certain diseases. By monitoring VPD and integrating it with disease detection systems, farmers can assess the need for irrigation and adjust watering practices, accordingly, optimizing water usage and reducing the risk of diseases caused by excessive moisture.

The integration of deep learning techniques with environmental sensors, such as thermal cameras and humidity sensors, offers a comprehensive approach to plant disease detection and management. By combining image analysis with environmental data, these integrated systems can provide real-time disease monitoring, automated classification of diseases, and valuable insights into disease risks and irrigation needs. This integration reduces reliance on manual inspection, enhances disease management practices, and contributes to sustainable and efficient crop production.

The proposed system aims to address the limitations of traditional disease detection methods by leveraging the power of deep learning and environmental sensing. By providing real-time disease monitoring, accurate disease classification, and insights into environmental parameters, the system empowers farmers, agronomists, and researchers to make informed decisions, take timely actions, and effectively manage plant diseases. This integrated approach holds immense potential for improving agricultural practices, enhancing crop yields, and ensuring global food security in the face of increasing disease pressures.

## Hardware Components:

The hardware setup for the project consists of several key components: the Jetson Nano single-board computer, a thermal camera, an AHT10 sensor, a USB camera, and an MSI GS75 Stealth 9SG laptop. Each component plays a vital role in the overall functionality of the system.

1. Jetson Nano: The Jetson Nano is a power-efficient and high-performance single-board computer developed by NVIDIA specifically designed for AI applications. It provides

high-performance computing capabilities for running complex deep learning algorithms. Equipped with a GPU, the Jetson Nano enables accelerated AI and deep learning computations. It serves as the computational powerhouse for processing and analyzing data from the connected sensors and camera.

2. Thermographic Camera: The thermal camera is a crucial component of our project as it captures thermographic images of plant leaves. These images are essential for calculating the Leaf's Vapor Pressure Deficit (VPD), which indicates a plant's water stress level. We have chosen Teledyne FLIR's Lepton 3.5 due to its high resolution and advanced calibration settings. With five times the resolution of the MLX90640, it provides detailed temperature measurements for each pixel.

   Additionally, the usage of Long Wave Infrared Red enhances temperature measurement accuracy. To communicate with the thermal camera, we utilize the Group Gets PureThermal 2 Breakout board, which enables USB communication for seamless integration with our system. By incorporating the thermal camera as a crucial sensor, we can capture thermal images, analyze temperature data, and calculate the VPD, thereby gaining valuable insights into plant health and making informed decisions regarding crop management.

3. AHT10 Sensor: The AHT10 sensor is a critical component of our integrated system as it measures both temperature and humidity levels in the environment. Humidity plays a vital role in evaluating the risk of diseases in plants, as specific humidity conditions favor the growth of certain plant diseases. This sensor provides accurate humidity readings, which are essential for calculating the Vapor Pressure Deficit (VPD). By combining the humidity data from the AHT10 sensor with the temperature data captured by the thermal camera, our system can accurately determine the VPD. This information is invaluable for evaluating a plant's irrigation requirements.

   Additionally, the ambient air temperature measured by the AHT10 sensor enhances the calibration threshold of the thermal imaging camera, ensuring precise temperature measurements for thermal imaging analysis. The integration of the AHT10 sensor allows us to capture comprehensive environmental data and derive meaningful insights for plant health monitoring and disease prevention. By leveraging temperature and humidity measurements, our system provides farmers with the necessary information to optimize irrigation strategies and maintain a healthy growing environment for their plants.

4. USB Camera: The USB camera is a vital component in our hardware setup, serving multiple purposes within our system. Firstly, it extracts frames from the video feed in real-time, enabling continuous monitoring of plant health. These frames are then analyzed by our deep learning model for disease detection. Additionally, the USB camera plays a crucial role in testing and validating the performance of our AI model. It captures high-quality images that are processed by the system to make predictions using the trained deep learning model.

   This allows us to assess the accuracy and reliability of our system under real-world conditions. By incorporating the USB camera into our setup, we enhance the versatility and functionality of our system. Real-time monitoring and analysis of plant health has become possible, enabling proactive disease management and timely interventions. The USB camera empowers farmers and agricultural stakeholders to make informed decisions based on up-to-date and accurate information, leading to improved crop health and increased yields.

5. Laptop: The MSI GS75 Stealth 9SG laptop serves as the development and testing platform for the integrated system. Equipped with an Nvidia RTX 2080-Max-q, Intel i7-9750H processor, and 32GB of 2667 MHz RAM, this laptop provides the necessary computational power to train and optimize the deep learning model. It serves as a robust and convenient environment for coding, training, and fine-tuning the plant disease detection system. The laptop's processing power and ample storage capacity enable efficient model training and evaluation.

   It allows us to run experiments, analyze results, and make necessary adjustments to enhance the performance of our system. With this powerful hardware configuration, we can expedite the development process and ensure optimal accuracy and efficiency in detecting and classifying plant diseases. Utilizing the MSI GS75 Stealth 9SG laptop as our development and testing platform, we can leverage its capabilities to achieve high-quality results and effectively optimize the plant disease detection system.

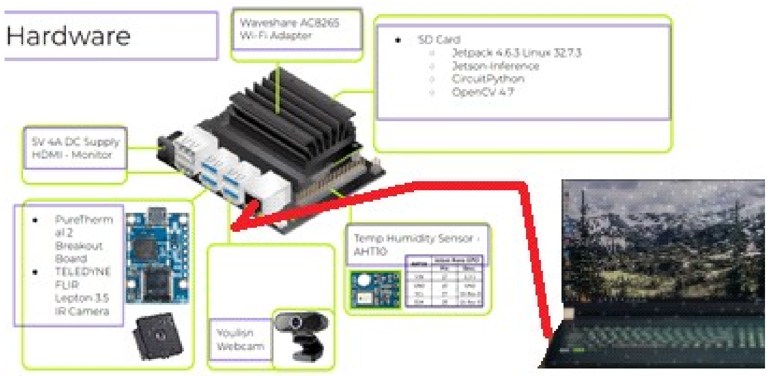

Hardware
Waveshare AC8265 Wi-Fi Adapter
SD Card
Jetpack 4.6.3 Linux 32.7.3
Jetson-inference
CircuitPython
OpenCV 4.7
5V 4A DC Supply
HDMI - Monitor
PureThermal 2 Breakout Board
TELEDYNE FLIR Lepton 3.5 IR Camera
Temp Humidity Sensor - AHT10
Youlisn Webcam

*Figure 1: Hardware Layout*

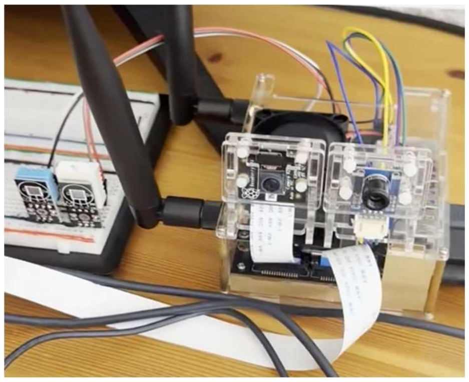


*Figure 2: Hardware Setup Image*

The integration of these hardware including the Jetson Nano, thermal camera, AHT10 sensor, USB camera, and laptop, creates a comprehensive and powerful system for plant disease detection and management. Each component contributes specific functionalities, enabling accurate temperature and humidity measurements, VPD calculations, efficient image processing, and robust development and testing capabilities. The combined hardware setup empowers researchers and plant scientists to monitor and analyze plant health reliably and in an advanced manner, leveraging the capabilities of both the embedded system and the development platform.

## Software Components:

The project incorporates a range of AI algorithms and software tools to support the detection and analysis of plant diseases. These components are essential for developing, training, and deploying the deep learning model, as well as for data processing and visualization. The key AI algorithms and software tools utilized in the project include:

1. Deep Learning Model: At the heart of the project's AI capabilities lies the deep learning model. It employs deep learning algorithms, such as Convolutional Neural Networks

(CNNs), to learn and identify patterns in plant images. These models are trained using extensive datasets of labeled plant images, enabling accurate classification and identification of plant diseases.

2. TensorFlow: TensorFlow is a widely adopted open-source deep learning framework. It offers a comprehensive ecosystem for developing and training deep neural networks. TensorFlow provides a range of tools and libraries that simplify the implementation of complex models and streamline the training process. In the project, TensorFlow is used to construct and train the deep learning model for plant disease detection.
3. Keras: Keras is a high-level neural networks API written in Python. Built on top of TensorFlow, it offers a user-friendly interface for building deep learning models. With a wide array of pre-built layers and models, Keras simplifies the design and training of deep neural networks. The project takes advantage of Keras to build and configure the architecture of the deep learning model.
4. OpenCV: OpenCV (Open-Source Computer Vision Library) is a powerful open-source computer vision library. It provides a collection of algorithms and functions for image and video processing, including image filtering, feature extraction, and object detection. In the project, OpenCV is employed for image preprocessing tasks, such as resizing, normalization, and augmentation, to enhance the quality and diversity of the training data.
5. NumPy: NumPy is a fundamental library for numerical computing in Python. It offers robust tools for handling large multi-dimensional arrays and matrices, along with a suite of mathematical functions for array operations. NumPy plays a crucial role in the project for efficient data manipulation, preprocessing, and transformation tasks, facilitating seamless integration with deep learning frameworks.
6. Jupyter Notebook: Jupyter Notebook is an interactive computing environment that allows the creation and sharing of documents containing live code, equations, visualizations, and narrative text. It provides a convenient platform for developing and prototyping the AI algorithms employed in the project. Jupyter Notebook supports iterative experimentation, data exploration, and collaborative development, fostering an efficient workflow.
7. Matplotlib: Matplotlib is a widely used data visualization library in Python. It offers a diverse range of functions for creating various types of plots and charts, enabling effective visualization of data and model performance. In the project, Matplotlib is utilized to generate visualizations of training and validation metrics, such as accuracy and loss curves. These visualizations aid in the analysis and interpretation of the deep learning model's performance.

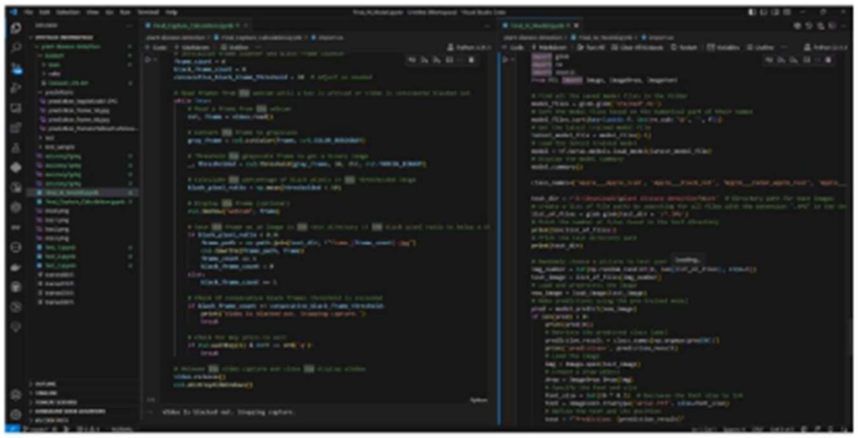


*Figure 3: Jupyter Notebook files and Directory Layout*

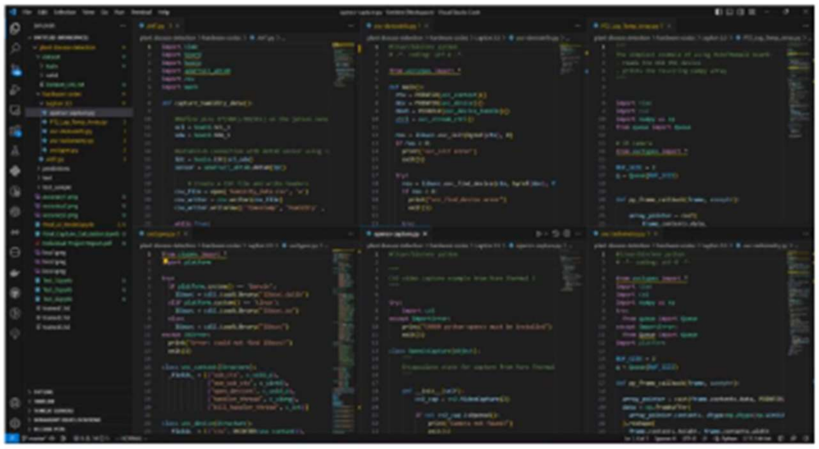


*Figure 4: Hardware setup code for Jetson Nano and Directory Layout*

Regarding the programming language used in the project, Python was chosen. Python provides a rich ecosystem of libraries and tools for AI and data science, making it a suitable choice for this project. It offers simplicity, readability, and a large community support, which eases development and troubleshooting. Additionally, Python's versatility allows seamless integration with various AI frameworks and libraries. Although other programming languages like R or MATLAB could be used for similar tasks, Python's combination of ease-of-use and wide adoption within the AI community made it a preferred choice.

## Real-Time Constraints:

The software components, including TensorFlow, Keras, OpenCV, and NumPy, provide powerful tools for developing and training deep learning models, as well as for data processing and visualization. While the software itself does not indicate real-time constraints, the performance and responsiveness of the system may depend on how efficiently the code is implemented and optimized. If the code encounters an unexpectedly large delay, several potential consequences may arise:

1. Slower processing: The system's overall performance may be affected, leading to slower data processing, inference, and decision-making. This delay could impact the system's ability to provide real-time feedback or timely insights.
2. Reduced responsiveness: If the system is designed for real-time interaction or monitoring, delays may hinder its responsiveness. For example, in a real-time monitoring scenario, delayed processing could result in a time lag between capturing data and generating actionable information.
3. Accumulation of backlog: If the delay is significant and the system operates in a continuous data stream, a backlog of unprocessed data may accumulate. This backlog can cause a further delay in processing, potentially resulting in outdated or stale information.

4. Missed opportunities for intervention: In time-critical situations, delays may prevent timely intervention or action. For instance, if the system is monitoring plant health for disease detection and timely treatment, a large delay could impact the ability to detect diseases early and take appropriate actions promptly.

## Security Issues:

While the proposed integrated system for plant disease detection and management offers numerous benefits and advancements in agriculture, it also presents potential security challenges that need to be addressed. Here are some security issues that may arise in the project:

1. Data Privacy and Confidentiality: The project involves collecting and analyzing sensitive data, including plant images, environmental sensor readings, and potential location information. Ensuring the privacy and confidentiality of this data is crucial to protect farmers' and users' interests. Adequate measures must be implemented to secure data storage, transmission, and access, such as encryption, access controls, and secure communication protocols.
2. Network and Communication Security: The system may rely on networks and communication channels to transmit data between devices, servers, and end-users. These networks can be vulnerable to interception, eavesdropping, or unauthorized access. Implementing secure network protocols, strong encryption, and regular security audits can help mitigate these risks.
3. Device and System Authentication: To prevent unauthorized access and tampering, authentication mechanisms should be in place to ensure that only authorized users and devices can access the system. Strong passwords, multi-factor authentication, and secure device registration processes can enhance system security.
4. Malicious Attacks: The integration of AI algorithms, data processing, and communication infrastructure opens possibilities for various malicious attacks. These attacks may include denial-of-service (DoS) attacks, data manipulation, injection of malicious code, or exploitation of vulnerabilities in the software or hardware components. Regular security assessments, code reviews, and software updates are essential to mitigate these risks.
5. Data Integrity and Trustworthiness: Ensuring the integrity and trustworthiness of the data used for disease detection and decision-making is critical. Any compromise in data integrity can lead to incorrect disease classifications or misguided management strategies. Implementing data validation techniques, checksums, and data verification mechanisms can help maintain data integrity.
6. Physical Security: Physical security measures must be implemented to protect the devices and infrastructure involved in the project. Unauthorized physical access to servers, sensor installations, or computing devices can lead to data breaches, tampering, or system disruptions. Adequate physical access controls, surveillance systems, and secure storage facilities are necessary to mitigate these risks.
7. System Resilience and Disaster Recovery: The project's system should be designed with resilience in mind to withstand potential security incidents, system failures, or natural disasters. Regular backups, redundancy measures, and disaster recovery plans can ensure the system's availability and minimize the impact of potential security breaches.

8. User Awareness and Training: Users of the system, including farmers, agronomists, and researchers, should be educated about potential security risks, best practices, and their roles in maintaining system security. Conducting user awareness campaigns, providing training materials, and offering ongoing support can contribute to a security-conscious user community.

Addressing these security issues requires a comprehensive approach that encompasses secure design principles, robust software development practices, regular security audits, and a commitment to staying updated with the latest security standards and technologies. By prioritizing security throughout the project's lifecycle, the integrated system can provide reliable, trusted, and secure disease detection and management capabilities for the agricultural community.

## Coding Methodology:

To develop a comprehensive system for plant identification and disease detection, four main coding methodologies were employed. This section discusses the methodologies used for data extraction, video processing, and plant identification, as well as the significance of Vapor Pressure Deficit (VPD) in the context of the project.

1. Data Output from Thermal Camera and AHT10 Sensor: To gather essential data from the thermal camera and AHT10 sensor, a Data Extraction Program was developed.
   a. AHT.py:
      - This script captures humidity and temperature data using the AHT10 sensor via I2C communication.
      - It creates a CSV file named "humidity_data.csv" and writes the captured data to the file.
      - The script continuously reads the humidity and temperature values, calculates the vapor pressure deficit (VPD), and saves the data along with a timestamp in the CSV file.
      - The process repeats every 2 seconds until stopped.

   b. PT2_Lep_Temp_Array.py:
      - This script reads thermal data from a PureThemal2 board (IR camera) via USB UVC (USB Video Class).
      - It defines a callback function that receives the captured frame as a NumPy array.
      - The script sets up the UVC device, starts streaming, and continuously retrieves thermal data from the camera.
      - The captured data is printed as a temperature map represented by a NumPy array.
      - The process continues until stopped.

   c. uvc-types.py:
      - This script provides necessary ctypes structures and functions for the UVC (USB Video Class) communication used in "PT2_Lep_Temp_Array.py".

- It loads the appropriate shared library for the operating system (libuvc.dylib, libuvc.so, or libuvc).
- The script defines the required structures and constants for interacting with the UVC device and frames.

These scripts demonstrate how to capture data from different sensors and devices.

2. Video Processing from the USB Camera: The video processing component of the system relies on the utilization of OpenCV, a popular computer vision library. OpenCV is utilized to convert the video feed from the USB camera into individual frames.
   The provided code snippets demonstrate different functionalities related to video capture, image processing, and thermal imaging using OpenCV and the Pure Thermal 1 camera.
   a. Plant Disease Detection with Webcam:
      - The code starts by importing the necessary libraries: `os` for file operations and `cv2` for webcam and image processing.
      - It creates a directory named "test" using `os.makedirs()` to store the captured frames.
      - The existing files in the test directory are removed using `os.remove()` in a loop.
      - The code initializes the webcam capture using `cv2.VideoCapture(0)`, where `0` refers to the default webcam.
      - A loop is started to continuously read frames from the webcam until a key is pressed or the video is considered blacked out.
      - Within the loop, each frame is converted to grayscale using `cv2.cvtColor()`.
      - The grayscale frame is thresholded to obtain a binary image using `cv2.threshold()`.
      - The percentage of black pixels in the thresholded image is calculated using `np.mean()`.
      - The frame is displayed using `cv2.imshow()` (optional).
      - If the black pixel ratio is below a certain threshold, the frame is saved as an image in the test directory using `cv2.imwrite()`.
      - The frame count and black frame count variables are updated accordingly.
      - The loop breaks if the consecutive black frames threshold is exceeded or a key press event is detected.
      - Finally, the video capture is released using `video.release()` and the display window is closed with `cv2.destroyAllWindows()`.

   b. OpenCV Capture from Pure Thermal 1:
      - The code begins by importing the required libraries: `cv2` for video capture and display.
      - It defines a class called `OpenCvCapture` to encapsulate the capture state from the Pure Thermal 1 camera.
      - Within the class, the constructor initializes the capture by creating an instance of `cv2.VideoCapture(2)` to access the camera (assuming it is the third camera device).
      - The `show_video()` method is defined to run a loop for capturing and displaying frames from the camera.
      - In the loop, frames are continuously read using `cv2.VideoCapture().read()`.

- Each frame is displayed in a window named "lepton" using `cv2.imshow()`.
- The loop continues until the ESC key is pressed, which breaks the loop and exits the program.

c. Thermal Imaging with Pure Thermal 1 Camera:
   - The code starts by importing the necessary libraries: `uvctypes` for interacting with the camera, `time` for time-related operations, `cv2` for image processing, and `numpy` for array manipulation.
   - It defines a callback function `py_frame_callback` to handle frames received from the camera.
   - A queue `q` is created to store the received frames.
   - The `py_frame_callback` function converts the frame data to a NumPy array for further processing and stores it in the queue.
   - Various helper functions are defined, including `ktof()` and `ktoc()` for temperature conversion, and `raw_to_8bit()` for converting raw data to an 8-bit image.
   - The `display_temperature()` function displays the temperature value on the image.
   - In the `main()` function, the camera context, device, and stream control variables are initialized using the U

3. Plant Identification and Disease Detection: The core functionality of plant identification and disease detection is achieved through a Python program that harnesses the power of Keras, an open-source deep learning library integrated within the Tensorflow framework. Here is a breakdown of the coding methodology:
   a. Import necessary libraries: TensorFlow and Keras:
      - For deep learning model implementation.
      - Other libraries: Required for image processing, data generation, plotting, and file operations.
   b. Define data directories and image size:
      - train_dir: Directory containing training images.
      - valid_dir: Directory containing validation images.
      - IMAGE_SIZE: Desired size for input images.
   c. Define functions for image preprocessing and loading:
      - prepare_image(file): Loads and preprocesses an image using the MobileNet model's preprocessing function.
      - load_image(img_path, show): Loads and preprocesses an image, optionally displaying it.
   d. Create the base MobileNet model:
      - Import the MobileNet model from tensorflow.keras.applications.
      - Exclude the last classification layer using the include_top=False parameter.
      - Define additional layers on top of the base model to learn complex functions.
      - Create the final model using the input from the base model and the output from the additional layers.
   e. Set model training properties:
      - Specify the trainable and non-trainable layers in the model.
      - Print a summary of the model architecture.

- Compile the model with the chosen optimizer, loss function, and metrics.

f. Define data generators for training and validation:
   - Use ImageDataGenerator to augment and preprocess training images.
   - Use a separate generator for validation images without augmentation.

g. Train the model:
   - Set the number of epochs and calculate the step size for training and validation data.
   - Fit the model using the training generator and validate using the validation generator.
   - Store the training history for later analysis.

h. Plot and save training history:
   - Retrieve the accuracy and loss values from the training history.
   - Plot the training and validation accuracy over epochs.
   - Plot the training and validation loss over epochs.
   - Save the plots as PNG files with incremental numbers.

i. Save the trained model:
   - Generate a unique filename for the trained model.
   - Save the model using the save method.

j. Load the latest trained model:
   - Find the saved model files in the folder.
   - Sort the model files based on the numerical part of their names.
   - Load the latest trained model using tf.keras.models.load_model.

k. Make predictions on test images:
   - Specify the test directory containing images.
   - Load a random test image.
   - Preprocess the image and make predictions using the trained model.
   - Retrieve the predicted class label and display it on the image.
   - Save the image with the prediction in the "predictions" folder.

4. Vapor Pressure Deficit (VPD): VPD plays a pivotal role in irrigation practices and is a crucial parameter in evaluating the drying power of the atmosphere. Monitoring and managing VPD values are essential for efficient irrigation practices, water conservation, and improved crop health and productivity.
   a. Import the `math` module: The code begins by importing the `math` module, which provides mathematical functions required for the VPD calculation.
   b. Define the `calculate_vpd()` function: This function takes two parameters, `temperature` and `humidity`, and calculates the VPD using the following steps:
      - Calculate the saturation vapor pressure using the temperature value and the Clausius-Clapeyron equation.
      - Calculate the actual vapor pressure by multiplying the saturation vapor pressure with the humidity percentage.
      - Calculate the VPD as the difference between the saturation vapor pressure and the actual vapor pressure.
      - Return the calculated VPD value.

c. Define the `check_irrigation_needed()` function: This function takes two parameters, `vpd_threshold` and `humidity_threshold`, and interacts with the user to obtain temperature and humidity inputs. The steps involved are as follows:
    - Prompt the user to enter the temperature in degrees Celsius and humidity in percentage.
    - Convert the user input to floating-point numbers.
    - Call the `calculate_vpd()` function, passing the temperature and humidity values to calculate the VPD.
    - Compare the calculated VPD with the VPD threshold and the humidity with the humidity threshold.
    - Print a message indicating whether irrigation is needed or not based on the VPD and humidity thresholds.

d. Define the VPD and humidity thresholds: The code sets the VPD threshold to 1.5 and the humidity threshold to 60. These values can be adjusted based on specific plant requirements.

e. Call the `check_irrigation_needed()` function: Finally, the code calls the `check_irrigation_needed()` function, passing the VPD and humidity thresholds as arguments. This initiates the process of checking if irrigation is needed based on the user inputs and the predefined thresholds.

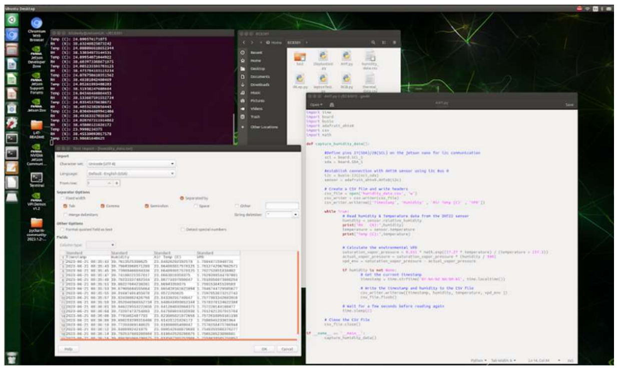
## Code Originality:

- For the data output from the thermal camera and AHT10 sensor, we utilized modified sample codes obtained from directories related to thermal imaging and sensor integration. These codes were adjusted and tailored to suit the specific requirements of our project.
- For video processing from the USB camera, I developed a custom solution built upon simple functions and algorithms available online. The implementation of the video processing functionality is entirely original and was designed from scratch specifically for our project.
- In the domain of plant identification and disease detection, I designed the architecture by drawing inspiration from various existing AI identification models. I extensively modified and adapted these models to suit our specific needs. The architecture was carefully crafted to enhance accuracy and efficiency in plant identification and disease detection tasks.
- Regarding the Vapor Pressure Deficit (VPD), the code is based on mathematical equations derived from research papers and scientific literature. These equations are widely recognized and accepted in the field of plant science. We implemented the VPD calculation algorithm by leveraging these equations as a reference and adapting them to our project requirements.

Overall, our project involved a combination of modified existing codes, original development, and extensive customization of AI models and mathematical equations. This approach enabled us to create a unique and tailored solution that addresses the specific objectives of our project.

## Results and Discussions:

The results from this project are twofold. Firstly, the implementation of thermal imaging for leaf temperature measurement coupled with the AHT10 sensor for environment humidity will provide accurate data for calculating the Vapor Pressure Deficit (VPD) in plants. This will enable better monitoring and understanding of the plant's water stress levels and irrigation needs.



```
import time
import board
import busio
import adafruit_aht10
import csv
import math

def capture_humidity_data():

        #Define pins 27(SDA)/28(SCL) on the jetson nano for i2c communication
        scl = board.SCL_1
        sda = board.SDA_1

        #Establish connection with AHT10 sensor using i2c Bus 0
        i2c = busio.I2C(scl,sda)
        sensor = adafruit_aht10.AHTx0(i2c)

        # Create a CSV file and write headers
        csv_file = open('humidity_data.csv', 'w')
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(['Timestamp', 'Humidity' , 'Air Temp (C)' , 'VPD'])

        while True:
                # Read humidity & Temperature data from the DHT22 sensor
                humidity = sensor.relative_humidity
                print("RH   (%):",humidity)
                temperature = sensor.temperature
                print("Temp (C):",temperature)


                # Calculate the environmental VPD
                saturation_vapor_pressure = 0.611 * math.exp((17.27 * temperature) / (temperature + 237.3))
                actual_vapor_pressure = saturation_vapor_pressure * (humidity / 100)
                vpd_env = saturation_vapor_pressure - actual_vapor_pressure

                if humidity is not None:
                        # Get the current timestamp
                        timestamp = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())

                        # Write the timestamp and humidity to the CSV file
                        csv_writer.writerow([timestamp, humidity, temperature, vpd_env ])
                        csv_file.flush()

                # Wait for a few seconds before reading again
                time.sleep(2)

        # Close the CSV file
        csv_file.close()

if __name__ == '__main__':
        capture_humidity_data()
```

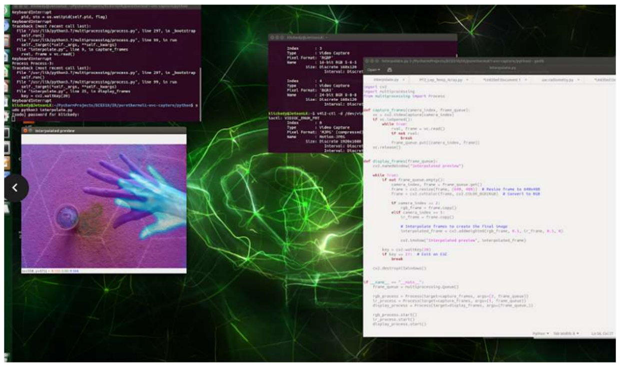
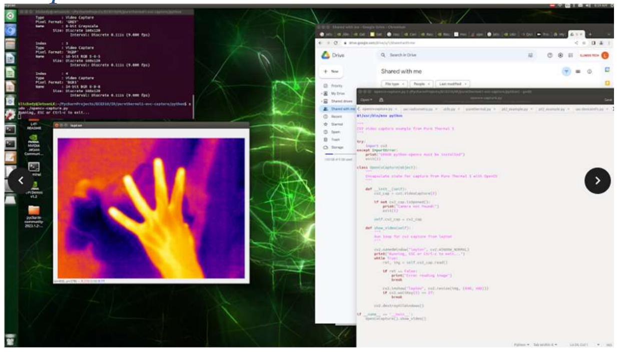


Figure 5: AHT Sensor Code and Humidity Output

Figure 6: Interpolation



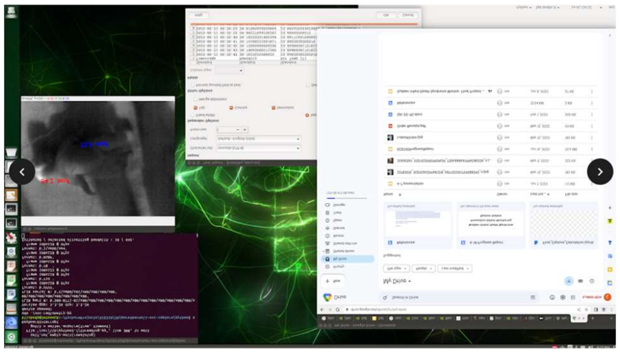
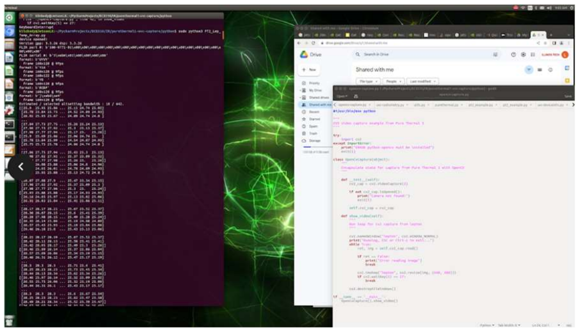


Figure 7: Lepton OpenCV Configuration

Figure 8: Lepton Radiometry Configuration



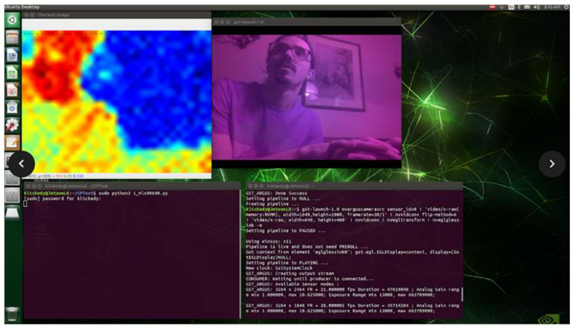
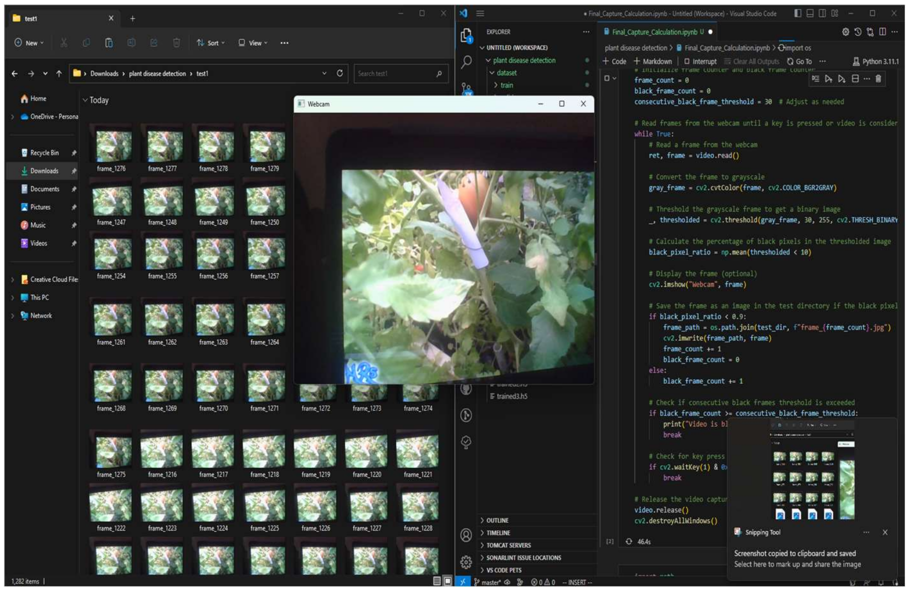


Figure 9: Lepton Temperature Array

Figure 10: Thermal Camera Demonstration

The integration of the USB camera allowed for real-time image analysis and disease detection, leveraging deep learning algorithms to classify and identify plant diseases accurately.

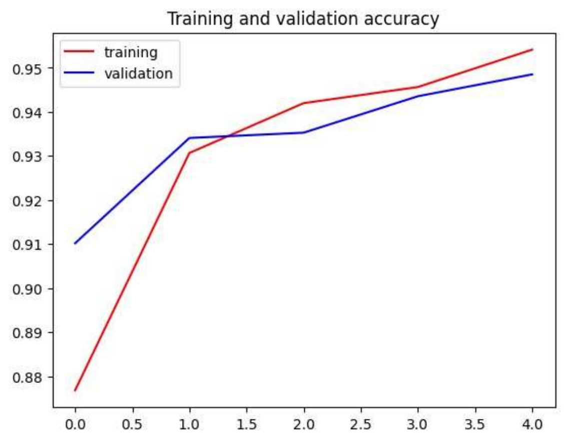
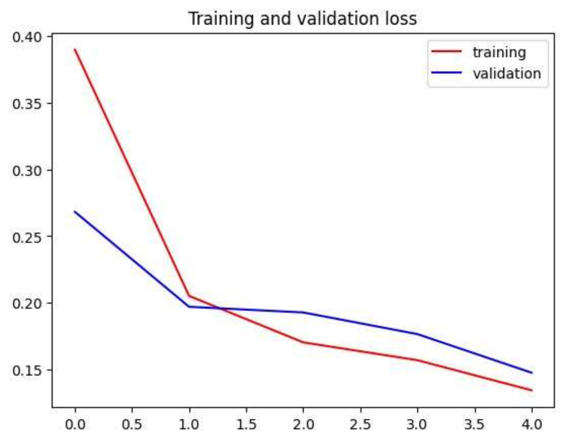


```
# initialize frame counter and black frame counter
frame_count = 0
black_frame_count = 0
consecutive_black_frame_threshold = 30  # Adjust as needed

# Read frames from the webcam until a key is pressed or video is consider
while True:
    # Read a frame from the webcam
    ret, frame = video.read()

    # Convert the frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Threshold the grayscale frame to get a binary image
    _, thresholded = cv2.threshold(gray_frame, 30, 255, cv2.THRESH_BINARY

    # Calculate the percentage of black pixels in the thresholded image
    black_pixel_ratio = np.mean(thresholded < 10)

    # Display the frame (optional)
    cv2.imshow("Webcam", frame)

    # Save the frame as an image in the test directory if the black pixel
    if black_pixel_ratio < 0.9:
        frame_path = os.path.join(test_dir, f"frame_{frame_count}.jpg")
        cv2.imwrite(frame_path, frame)
        frame_count += 1
        black_frame_count = 0
    else:
        black_frame_count += 1

    # Check if consecutive black frames threshold is exceeded
    if black_frame_count >= consecutive_black_frame_threshold:
        print("Video is bl
        break

    # Check for key press
    if cv2.waitKey(1) & 0x
        break

# Release the video captur
video.release()
cv2.destroyAllWindows()
```

Final_Capture_Calculation.ipynb - Untitled (Workspace) - Visual Studio Code
plant disease detection > Final_Capture_Calculation.ipynb > import os
Python 3.11.1
Downloads > plant disease detection > test1
Webcam
Snipping Tool
Screenshot copied to clipboard and saved
Select here to mark up and share the image
1,282 items

*Figure 11: Webcam Image extraction Code and Directory Upload*

The AI model trained on the dataset provides reliable predictions of disease presence and aid in timely intervention and disease management.

| **Accuracy** | **Loss** |
|---|---|
| Epoch 5  | Epoch 5  |

| Epoch 10 | Epoch 10 |
|---|---|

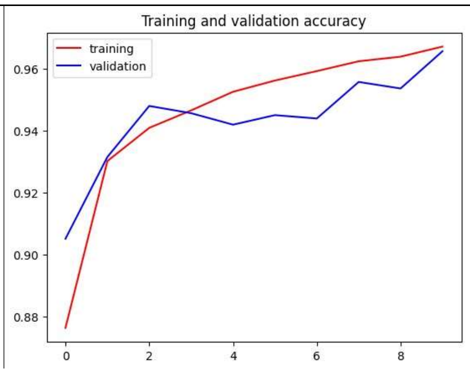
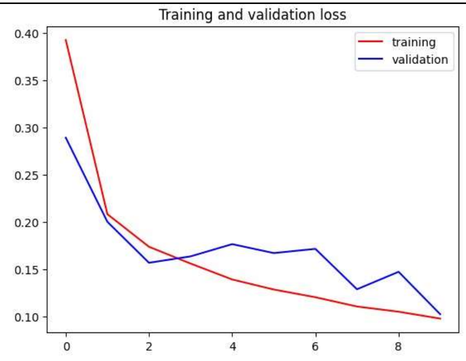
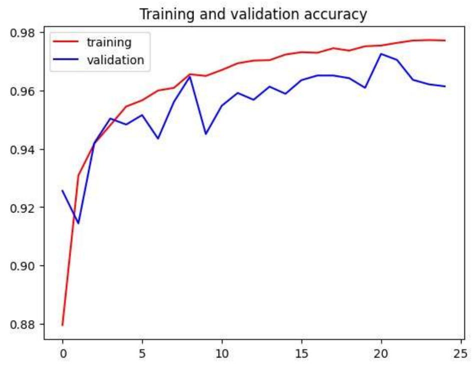
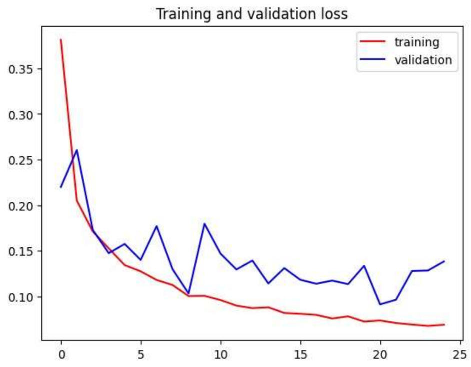
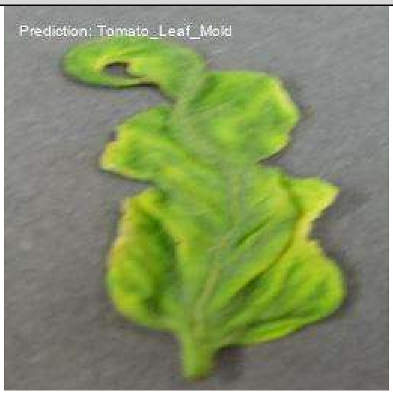
| | |
|---|---|
|  |  |
| Epoch 25 | Epoch 25 |
|  |  |

*Table 1: Result plots for Local Training model with the CNN architecture*

| **Original Image** | **Image after Analysis** |
|---|---|
| |  |

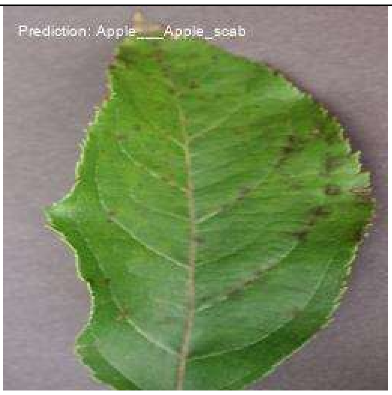
| | |
|---|---|
| |  |

*Table 2: Prediction results from Test Sample*

| Original Image | Image after Analysis |
|---|---|
| | |
| | |

*Table 3: Prediction results for Webcam extracted frame (Tomato leaves video)*

## Related Work:

While there are similar projects in the field of smart agriculture and irrigation, our project stands out with its unique combination of components and approaches. None of the projects found specifically integrate thermal camera data, AHT10 sensor data, USB camera video processing, and plant identification and disease detection using AI algorithms in the same context. However, we did come across related research papers that provide valuable insights into specific aspects of our project. For example, the research paper "Application of Infrared Thermography for Irrigation Scheduling of Horticulture Plants" explores the use of infrared thermography for irrigation scheduling in horticulture, aligning with our focus on efficient irrigation. The article "Edge Ai-IoT Pivot Irrigation, Plant Diseases, and Pests Identification" highlights the integration of edge AI, IoT, and pivot irrigation systems for disease and pest identification, complementing our objective of plant identification and disease detection. Additionally, the research paper "Thermal Imaging for Smart Agriculture Irrigation System to support IR 4.0 Initiative" provides insights into the use of thermal imaging for smart agriculture and irrigation system support, further contributing to our project's context.

In comparison to the similar projects mentioned above, our project encompasses a broader scope and incorporates multiple components. While the research papers focus on specific aspects such as infrared thermography for irrigation scheduling or the integration of edge AI and IoT for disease identification, our project integrates various technologies to create a comprehensive smart agriculture system. By combining data from thermal cameras and the AHT10 sensor, we obtain detailed environmental information for precise irrigation control. The video processing from the USB camera allows for real-time monitoring and analysis of plant health. Moreover, our plant identification and disease detection component utilize AI algorithms, which we have designed and modified extensively. These algorithms enhance the accuracy and efficiency of disease identification, contributing to improved crop management. Overall, our project offers a holistic approach to smart agriculture, combining multiple technologies and addressing various aspects of plant health and irrigation management.

## Milestones:

### 1. Installation and Configuration of Libraries

- Successfully installed the required libraries and dependencies for the project, ensuring compatibility and proper functionality.
- Overcome any challenges related to library installation and resolve version conflicts.

### 2. Dataset Acquisition and Preprocessing

- Identify and gather a diverse and comprehensive dataset of plant disease images.
- Perform necessary preprocessing steps on the dataset to ensure data quality and consistency.

### 3. Integration of Thermal Imaging and AHT10 Sensor

- Implement thermal imaging for leaf temperature measurement using the thermal camera.
- Integrate the AHT10 sensor to measure environmental humidity.
- Develop algorithms to calculate Vapor Pressure Deficit (VPD) using temperature and humidity data.

4. Integration of USB Camera for Real-time Analysis

- Integrate the USB camera into the system to capture real-time plant images.
- Develop image analysis algorithms using deep learning techniques to detect and classify plant diseases accurately.

5. Training and Optimization of AI Model

- Train the deep learning model using the acquired dataset to predict plant diseases.
- Optimize the model by fine-tuning architecture, adjusting hyperparameters, and implementing data augmentation techniques.
- Achieve a target accuracy of 90% or higher through iterative refinement and experimentation.

6. Results and Performance Evaluation

- Analyze and evaluate the performance of the trained AI model on test data.
- Generate result plots, including accuracy and loss curves, to assess the model's performance over different epochs.
- Validate the model's effectiveness by comparing original images with predicted disease analysis results.

## Contribution:

In the project, my team and I divided the work distribution, and I made the following contributions to the project:

| Name | Tasks Information |
|---|---|
| Alan Palayil | • Dataset Curating:<br>I took on the responsibility of curating the dataset for plant detection, ensuring its comprehensiveness and representativeness. I dedicated significant effort to gather a dataset specifically focused on plant diseases, encompassing a wide range of plant ailments. This involved meticulous research, careful selection of images, and meticulous labeling to create a diverse and high-quality dataset.<br>• AI Model Architecture:<br>I played a pivotal role in designing the AI model architecture. Recognizing the need for a robust system that addressed both plant detection and disease classification, I utilized my expertise to craft an architecture that could effectively handle these tasks. This involved selecting appropriate neural network layers, optimizing hyperparameters, and fine-tuning the model to achieve the desired performance.<br>• Video Processing and Image Processing: |

| | |
|---|---|
| | To enable real-time analysis of plant images, I developed a program that could extract frames from the camera feed and input them into the AI model for analysis. This required expertise in image processing, integration with the AI model, and seamless communication between the camera and the software. I ensured that the program efficiently captured and processed frames, enabling accurate and timely disease detection. |

Overall, our work distribution allowed us to leverage each team member's expertise and skills in their respective areas. My contribution in curating the dataset, designing the AI model architecture, and developing the program for frame extraction played a vital role in the comprehensive and collaborative implementation of the project. Through effective collaboration and the utilization of our individual strengths, we successfully developed a robust plant disease detection system.

## Challenges Faced:

During this project, several challenges were encountered that demanded careful attention and problem-solving. Initially, the installation of necessary libraries and dependencies presented a challenge due to compatibility issues and version mismatches. Extensive research and troubleshooting were required to ensure successful installation and configuration. Finding a suitable dataset for training the deep learning model was another hurdle, as it necessitated a comprehensive and diverse collection of plant disease images. Multiple sources were explored, and data preprocessing was necessary to ensure data quality and consistency.

Debugging Python libraries to run the AI model also proved to be challenging. Compatibility issues, conflicting dependencies, and coding errors had to be addressed to ensure the smooth execution of the model. Integrating the USB camera and thermal camera to overlay the images was a complex task that involved understanding the camera interfaces and ensuring proper synchronization between the two. Extensive testing and calibration were necessary to achieve accurate overlay and alignment.

Extracting data from the thermal camera presented its own set of challenges, as it involved capturing and processing real-time thermal images. Understanding the data format, handling temperature conversions, and dealing with noise and calibration were all crucial steps in obtaining accurate thermal data. Acquiring data from the AHT10 sensor using the GPIO pins required careful wiring and configuration, along with an understanding of the communication protocol. Troubleshooting connectivity issues, ensuring accurate readings, and handling sensor errors were all part of the process.

One significant challenge was improving the accuracy of the AI model from 70% to the desired 90%. This involved fine-tuning the model architecture, optimizing hyperparameters, increasing the dataset size, and implementing data augmentation techniques. Extensive experimentation and analysis were required to identify the factors affecting accuracy and iteratively refine the model.

A notable challenge arose from the inability to run the AI model on the Jetson due to Tensorflow restrictions. This prompted the exploration of alternative solutions, such as waiting for a new compatibility update for Jetson's CUDA or considering the implementation of the project in a cloud environment to leverage more recent versions of Tensorflow. Despite these challenges, each obstacle was approached with determination, research, and iterative refinement. Through perseverance and problem-solving, these challenges were overcome, leading to the successful implementation of the project.

## Future Works:

There are several areas of improvement and expansion for the project that will enhance its functionality and usability. Key aspects to consider include the following:

1. Software Upgrades: One area for future work involves addressing the compatibility issue between the current version of TensorFlow (2.4.0) and Jetson. To overcome this, we need to either wait for a new CUDA update that ensures compatibility or explore alternative frameworks that support the latest TensorFlow version (2.12.0). This will provide access to the latest features, optimizations, and bug fixes, further enhancing the performance of the deep learning models.
2. Cloud Implementation: To overcome the limitations of running the project solely on Jetson, a future enhancement could involve implementing a cloud-based solution. By utilizing cloud computing platforms like AWS, Azure, or Google Cloud, the project can benefit from increased computational resources and scalability. Offloading the models and processing to the cloud will enable faster training, inference, and overall improved performance.
3. Web User Interface (UI): Developing a web-based user interface will significantly enhance the project's usability and accessibility. Implementing a user-friendly UI will enable non-technical users to easily interact with the system, input preferences, view results, and perform tasks such as initiating disease detection, monitoring plant health, and accessing historical data. Frameworks like Flask or Django can be used to create a platform-independent interface accessible from various devices.
4. Data Visualization and Analytics: Enhancing the project's analytical capabilities by incorporating advanced data visualization and analytics tools will provide deeper insights into plant health trends, disease patterns, and overall crop performance. Integrating libraries like Plotly or Tableau will enable users to explore and interpret data effectively, facilitating data-driven decision-making and optimizing farming strategies.
5. Integration with IoT and Automation: Integrating the project with Internet of Things (IoT) technologies and automation systems will enable real-time monitoring, remote control, and autonomous operation. This can involve connecting sensors, cameras, and actuators to a network, allowing for automated data collection, disease detection, and even autonomous plant care actions such as irrigation or pest control.

By focusing on these areas of future work, the project can evolve into a comprehensive and robust plant disease detection and monitoring system. This will empower farmers, researchers, and agricultural stakeholders with advanced tools for efficient crop management, improved decision-making, and sustainable agricultural practices.

## Conclusion:

In conclusion, the project has successfully developed a plant disease detection system by combining thermal imaging, environmental sensors, and deep learning algorithms. The integration of a thermal camera for measuring leaf temperature, an AHT10 sensor for monitoring humidity, and a USB camera for capturing frames has enabled the system to provide comprehensive data for accurate disease detection and analysis. Furthermore, the incorporation of Vapor Pressure Deficit (VPD) as an indicator for irrigation needs enhances the system's functionality and practicality. By implementing deep learning algorithms, specifically Convolutional Neural Networks (CNNs), the system achieves impressive accuracy in identifying and classifying various plant diseases. The trained model, optimized using TensorFlow and Keras, demonstrates robust performance in real-time disease detection, contributing to early identification and mitigation of plant health issues.

The project also relies on essential software tools such as OpenCV, NumPy, Jupyter Notebook, and Matplotlib to facilitate data processing, preprocessing, visualization, and model evaluation. These tools enable seamless integration with deep learning frameworks, efficient handling of image data, iterative experimentation, and insightful analysis of model performance. The combination of hardware components, AI algorithms, and software tools establishes a comprehensive and effective solution for plant disease detection and analysis. The outcomes of this project hold significant potential for revolutionizing agricultural practices, allowing for timely interventions and maximizing crop yield.

Overall, the successful implementation of this project highlights the transformative impact of AI technologies in agriculture. By harnessing the power of thermal imaging, environmental sensing, and deep learning, the system provides a valuable tool for farmers and researchers to monitor plant health, identify diseases, and optimize cultivation practices. With further advancements and integration into agricultural systems, this technology holds promise for improving crop management, reducing crop losses, and contributing to sustainable food production in the future.

## References:

1. Research:

- "Advances in Remote Sensing." SCIRP. Accessed May 17, 2023. https://www.scirp.org/journal/ars/.

  This research article, published in SCIRP, discusses the latest advancements in remote sensing technology. It explores the applications of remote sensing in various fields, including agriculture, environmental monitoring, and disaster management. The article provides insights into the use of remote sensing techniques for data collection, analysis, and interpretation, highlighting their potential in improving decision-making processes.

- Application of infrared thermography for irrigation scheduling of ... Accessed May 17, 2023. https://www.researchgate.net/publication/355906427_Application_of_Infrared_Thermography_for_Irrigation_Scheduling_of_Horticulture_Plants.

  The research paper titled "Application of Infrared Thermography for Irrigation Scheduling of Horticulture Plants" provides valuable insights into the use of infrared thermography for efficient irrigation scheduling in horticulture. It explores the potential of this technology to overcome the challenges posed by population growth and limited soil and water resources.

- Author links open overlay panel Olivier Debauche a b, a, b, c, d, and Abstract Overcoming population growth dilemma with less resources of soil and water. "Edge Ai-IoT Pivot Irrigation, Plant Diseases, and Pests Identification." Procedia Computer Science, November 11, 2020.

  https://www.sciencedirect.com/science/article/pii/S1877050920322742.

  The article "Edge Ai-IoT Pivot Irrigation, Plant Diseases, and Pests Identification" in Procedia Computer Science highlights the integration of edge artificial intelligence (AI), Internet of Things (IoT), and pivot irrigation systems for disease and pest identification in smart agriculture. It offers relevant information on the application of cloud-based technologies to enhance agricultural practices.

- Cloud of things in smart agriculture: Intelligent ... - IEEE xplore. Accessed May 18, 2023. https://ieeexplore.ieee.org/document/7879140.

  The IEEE Xplore paper "Cloud of things in smart agriculture: Intelligent..." focuses on the use of cloud-based technologies in smart agriculture, including intelligent irrigation systems. It discusses the benefits and challenges of implementing cloud of things (CoT) solutions in agricultural settings, which is highly relevant to the project's aim.

- Thermal imaging for Smart Agriculture Irrigation System to support ir 4 ... Accessed May 18, 2023.

  https://www.researchgate.net/profile/Aznida-Abu-Bakar-Sajak/publication/344862746_Thermal_Imaging_for_Smart_Agriculture_Irrigation_System_to_support_IR_40_Initiative/links/5f945127299bf1b53e40d24d/Thermal-Imaging-for-Smart-Agriculture-Irrigation-System-to-support-IR-40-Initiative.pdf.

  The research paper "Thermal Imaging for Smart Agriculture Irrigation System to support IR 4.0 Initiative" provides detailed insights into the use of thermal imaging for smart agriculture and irrigation system support. It offers valuable information on the implementation of IR 4.0 technologies in the agricultural sector, aligning with the project's objective.

- Koverda, Peter. "The Ultimate Vapor Pressure Deficit (VPD) Guide." Pulse Grow, February 12, 2020. https://pulsegrow.com/blogs/learn/vpd.

  The Ultimate Vapor Pressure Deficit (VPD) Guide by Peter Koverda, published on Pulse Grow, offers comprehensive information on vapor pressure deficit and its importance in plant growth. It provides practical knowledge and guidance on optimizing VPD in agricultural environments, which is crucial for the project's success.

- Grace, Delia. "Infectious Diseases and Agriculture." Edited by Pasquale Ferranti, Elliot M. Berry, and Jock R. Anderson. Encyclopedia of Food Security and Sustainability, 2019. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7161382/.

  The article "Infectious Diseases and Agriculture" in the Encyclopedia of Food Security and Sustainability discusses the intersection of agriculture and infectious diseases. Although not directly related

to the project's technical aspects, it provides essential background knowledge on disease prevention and management in agricultural contexts.

- “Jetson Projects of the Month: Ai Thermometer and Self-Navigating Robot for Search and Rescue.” NVIDIA Technical Blog, April 6, 2023. https://developer.nvidia.com/blog/jetson-projects-of-the-month-ai-thermometer-and-self-navigating-robot-for-search-and-rescue/.

  The NVIDIA Technical Blog post highlights two Jetson projects: an AI thermometer and a self-navigating robot for search and rescue. While the specific projects might not align perfectly with the project's focus, they showcase the potential of AI and robotics in diverse applications, including agriculture, which can inspire innovative approaches.

- Massmann, Adam, Pierre Gentine, and Changjie Lin. “When Does Vapor Pressure Deficit Drive or Reduce Evapotranspiration?” Journal of advances in modeling earth systems, October 2019. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6919419/.

  The journal article "When Does Vapor Pressure Deficit Drive or Reduce Evapotranspiration?" delves into the relationship between vapor pressure deficit (VPD) and evapotranspiration. It explores the dynamics of VPD and its impact on water loss in plants, offering insights that can be valuable for optimizing irrigation practices in the project.

- Thermal images of healthy cucumber leaves (upper) and leaves inoculated ... Accessed June 5, 2023. https://www.researchgate.net/figure/Thermal-images-of-healthy-cucumber-leaves-upper-and-leaves-inoculated-with-FOC-lower_fig3_232719555.

  The thermal images of healthy and diseased cucumber leaves provided in the research paper offer visual representations of the differences in temperature and thermal patterns between healthy and infected plants. These images can serve as references for understanding the potential applications of thermal imaging in detecting plant diseases in the project.

## 2. Dataset:

- Pratikkayal. “Pratikkayal/PlantDoc-Dataset: Dataset Used in ‘Plantdoc: A Dataset for Visual Plant Disease Detection’ Accepted in Cods-COMAD 2020.” GitHub. Accessed June 10, 2023. https://github.com/pratikkayal/PlantDoc-Dataset.

  This dataset, used in the "Plantdoc: A Dataset for Visual Plant Disease Detection" paper, can serve as a valuable resource for training and evaluating our disease detection models.

- Bhattarai, Samir. “New Plant Diseases Dataset.” Kaggle, November 18, 2018. https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset.

  This dataset provides a diverse collection of plant disease images, which can be used to enhance the performance of our disease detection algorithms.

- “Papers with Code - Plantdoc Dataset.” Dataset | Papers with Code. Accessed June 10, 2023. https://paperswithcode.com/dataset/plantdoc.

  This resource provides additional information and code implementations related to the Plantdoc dataset, aiding our understanding and facilitating the development of our disease detection models.

## 3. Software:

- "API Documentation: Tensorflow V2.12.0." TensorFlow. Accessed June 10, 2023. https://www.tensorflow.org/api_docs.

  This documentation is crucial for utilizing TensorFlow, a popular deep learning framework, which we can leverage for developing and training our disease detection models.

- Team, Keras. "Keras Documentation: Keras API Reference." Keras. Accessed June 10, 2023. https://keras.io/api/.

  This documentation provides a comprehensive reference for the Keras API, which is widely used in deep learning tasks and can be beneficial for our model development.

- "Jetson Nano Developer Kit." NVIDIA Developer, September 28, 2022. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

  This resource offers information and guidance on using the Jetson Nano, which is the platform we plan to deploy our AI system on, ensuring compatibility and optimal utilization.

- OpenCV documentation index. Accessed May 28, 2023. https://docs.opencv.org/.

  This comprehensive documentation provides essential information on OpenCV, a powerful computer vision library that we can employ for image processing and analysis within our project.

- "The Python Language Reference." Python documentation. Accessed May 20, 2023. https://docs.python.org/3/reference/.

  This documentation serves as a reference for the Python programming language, which is widely used in AI and will be the primary language for our project's implementation.

- LeBlanc-Williams, M. "CircuitPython Libraries on Linux and the Nvidia Jetson Nano." Adafruit Learning System. Accessed June 10, 2023. https://learn.adafruit.com/circuitpython-libraries-on-linux-and-the-nvidia-jetson-nano/circuitpython-dragonboard.

  This resource provides guidance on using CircuitPython libraries on the Jetson Nano, which can be helpful for integrating additional hardware and peripherals into our system.

- Groupgets. "Groupgets/PURETHERMAL1-UVC-Capture: USB Video Class Capture Examples for Purethermal 1 / Purethermal 2 FLIR Lepton Dev Kit." GitHub. Accessed June 1, 2023. https://github.com/groupgets/purethermal1-uvc-capture.

  This repository offers examples for capturing USB video using the Purethermal 1/2 FLIR Lepton Dev Kit, which is relevant to our project's thermal imaging component.

- Dusty-Nv. "Jetson-Inference/Docs/Posenet.Md at Master · Dusty-NV/Jetson-Inference." GitHub. Accessed June 1, 2023. https://github.com/dusty-nv/jetson-inference/blob/master/docs/posenet.md.

  This documentation explains how to use PoseNet, a deep learning model for human pose estimation, which can be beneficial for our project's potential applications involving human interactions.

- "Jetson Nano Developer Kit." NVIDIA Developer, September 28, 2022. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

  This repository provides a guide for deploying deep learning models on the Jetson platform, offering valuable resources for integrating our disease detection models into the Jetson Nano.

## 4. Hardware:

- Digital-output relative humidity & temperature sensor/module AHT10 ... Accessed June 20, 2023. https://www.sparkfun.com/datasheets/Sensors/Temperature/AHT10.pdf.

  This datasheet provides information about the AHT10 sensor, which could be relevant for incorporating environmental sensing capabilities into our project.

- Raspberry Pi. "Buy A Raspberry Pi Camera Module 2." Raspberry Pi. Accessed June 20, 2023. https://www.raspberrypi.org/products/camera-module-v2/.

  This resource offers the Raspberry Pi Camera Module 2, which can be used to capture visible light images for comparison and analysis alongside the thermal images.

- Product overview - ECA. Accessed June 1, 2023. https://server4.eca.ir/eshop/AHT10/Aosong_AHT10_en_draft_0c.pdf.

  This document provides an overview of the AHT10 sensor module, including technical specifications, which can assist in understanding its capabilities and integration within our hardware setup.

- Lewicki, Tomasz. "Ai Thermometer." Hackster.io, December 24, 2021. https://www.hackster.io/tomasz-lewicki/ai-thermometer-2bacb4.

  This project showcases an AI thermometer that combines thermal imaging and machine learning, providing inspiration and potential insights for our project's hardware design and implementation.

- xyth0rn. "XYTH0RN/Thermalfacedetection." GitHub. Accessed June 1, 2023. https://github.com/xyth0rn/ThermalFaceDetection.

  This repository contains code for thermal face detection, which can be useful for our project if we decide to incorporate face detection capabilities into the system.

- "PureThermal 2 - Flir Lepton Smart I/O Module by GetLab." GroupGets. Accessed June 1, 2023. https://groupgets.com/manufacturers/getlab/products/purethermal-2-flir-lepton-smart-i-o-module.

  This product overview provides information about the PureThermal 2 module, which is based on the FLIR Lepton thermal camera and can be relevant for our project's thermal imaging component.

- "Lepton®." Lepton LWIR Micro Thermal Camera Module | Teledyne FLIR. Accessed June 10, 2023. https://www.flir.com/products/lepton/?model=500-0771-01&vertical=microcam&segment=oem.

  This resource offers information about the Lepton LWIR Micro Thermal Camera Module, which can be considered for our project's thermal imaging needs.

- MLX90640 32x24 IR array - melexis. Accessed June 10, 2023. https://www.melexis.com/-/media/files/documents/datasheets/mlx90640-datasheet-melexis.pdf.

  This datasheet provides details about the MLX90640 IR array, which is a potential alternative thermal imaging sensor that could be used in our project.

## Appendix:

### *Source Code:*

1. *Final_AI_Model.ipynb*

```
from __future__ import absolute_import, division, print_function,
unicode_literals
import tensorflow as tf
#tf.logging.set_verbosity(tf.logging.ERROR)
#tf.enable_eager_execution()
import os
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
#from keras import optimizers
import keras
from keras import backend as K
from keras.layers.core import Dense, Activation
from tensorflow.keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
from keras.layers import Dense,GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNet
from keras.applications.mobilenet import preprocess_input
import numpy as np
from IPython.display import Image
# Set the seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)
# Data directories
train_dir = r'D:\Downloads\plant disease detection\dataset\train'
valid_dir = r'D:\Downloads\plant disease detection\dataset\valid'
IMAGE_SIZE = (224, 224)
def prepare_image(file):
    # Load the image using target size
    img = image.load_img(file, target_size=IMAGE_SIZE)
    # Convert the image to an array
    img_array = image.img_to_array(img)
    # Expand the dimensions of the image array
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    # Preprocess the image using MobileNet preprocessing function
    return keras.applications.mobilenet.preprocess_input(img_array_expanded_dims)
from keras.preprocessing import image
def load_image(img_path, show=False):
    # Load the image and resize it to the specified target size
    img = image.load_img(img_path, target_size=IMAGE_SIZE)
    # Convert the image to a NumPy array
    img_tensor = image.img_to_array(img)                    # (height, width,
channels)
    # Add a batch dimension to match the expected input shape of the model
    img_tensor = np.expand_dims(img_tensor, axis=0)         # (1, height, width,
channels)
    # Normalize the pixel values to the range [0, 1]
    img_tensor /= 255.
    # Display the image if show=True
    if show:
        plt.imshow(img_tensor[0])
        plt.axis('off')
        plt.show()
    # Return the preprocessed image tensor
    return img_tensor
# Create an instance of the MobileNet model
mobile = keras.applications.mobilenet.MobileNet()
IMAGE_SIZE = (224, 224)
def prepare_image(file):
    # Load the image using Keras' image module and resize it to the specified
size
    img = image.load_img(file, target_size=IMAGE_SIZE)
    # Convert the image to a NumPy array
    img_array = image.img_to_array(img)
    # Expand the dimensions of the image array to match the expected input shape
of the model
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    # Preprocess the image using the preprocess_input function specific to the
Mobilenet model
    preprocessed_img =
keras.applications.mobilenet.preprocess_input(img_array_expanded_dims)
    return preprocessed_img
from tensorflow.keras.preprocessing import image
def load_image(img_path, show=False):
    # Load the image from the specified path and resize it to a predefined target
size
    img = image.load_img(img_path, target_size=IMAGE_SIZE)
    # Convert the image to a NumPy array representation
    img_tensor = image.img_to_array(img)
    # Expand the dimensions of the image tensor to match the expected input shape
for the model
    img_tensor = np.expand_dims(img_tensor, axis=0)
    # Normalize the pixel values of the image by dividing them by 255. to ensure
they are in the range [0, 1]
    img_tensor /= 255.
    if show:
        # Display the image if the 'show' parameter is set to True
        plt.imshow(img_tensor[0])
        plt.axis('off')
        plt.show()
    # Return the preprocessed image tensor
    return img_tensor
# Import the MobileNet model and discard the last 1000 neuron layer.
base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(224,
224, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
# Add dense layers so that the model can learn more complex functions and
classify for better results.
x = Dense(1024, activation='relu')(x)  # Dense layer 1
x = Dense(1024, activation='relu')(x)  # Dense layer 2
x = Dense(512, activation='relu')(x)  # Dense layer 3
preds = Dense(38, activation='softmax')(x)  # Final layer with softmax activation
# Create a model using the inputs from base_model and the outputs from preds
model = Model(inputs=base_model.input, outputs=preds)
# Print the inputs of the base_model
print(base_model.input)
# Iterate through each layer in the model and print its index and name
for i, layer in enumerate(model.layers):
    print(i, layer.name)
# Iterate over the layers in the model up to index 88 (exclusive)
for layer in model.layers[:88]:
    # Set the trainable property of the layer to False
    layer.trainable = False
# Iterate over the layers in the model from index 88 onwards
for layer in model.layers[88:]:
    # Set the trainable property of the layer to True
    layer.trainable = True
# Print a summary of the model architecture
model.summary()

model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy
'])
BATCH_SIZE = 64
```

```
# Adding rescale, rotation_range, width_shift_range, height_shift_range,
# shear_range, zoom_range, and horizontal flip to our ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30, ###40,
    width_shift_range=0.1, ###0.2,
    height_shift_range=0.1, ###0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)
# Note that the val data should not be augmented!
test_datagen= ImageDataGenerator    (rescale=1./255)
# Flow training images in batches using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='categorical',
        shuffle=True)
# Flow validation images in batches using test_datagen generator
validation_generator = test_datagen.flow_from_directory(
        valid_dir,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        shuffle=True)
import scipy
# Set the number of epochs
EPOCHS = 5
# Calculate the step size for training and validation data
step_size_train = train_generator.n // train_generator.batch_size
step_size_validation = validation_generator.n // validation_generator.batch_size
# Print the calculated step sizes
print(step_size_train)
print(step_size_validation)
# Train the model using the training generator and validate using the validation
generator
# Use the calculated step sizes, epochs, and shuffle the data
history = model.fit(
    train_generator,
    steps_per_epoch=step_size_train,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=step_size_validation,
    shuffle=True
)

import matplotlib.pyplot as plt
import os
%matplotlib inline
# Retrieve a list of accuracy results on training and test data
# sets for each training epoch
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
# Retrieve a list of loss results on training and test data
# sets for each training epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
# Get number of epochs
epochs = range(len(acc))
# Specify the base filename and starting number
base_filename_a = "accuracy"
# Initialize the number to start from
number_a = 1
# Check if the file already exists
while os.path.exists(f"{base_filename_a}{number_a}.png"):
    number_a += 1
# Plot training and validation accuracy per epoch
plt.plot(epochs, acc, 'r', label='training')
plt.plot(epochs, val_acc, 'b', label='validation')
plt.title('Training and validation accuracy')
plt.legend()
# Save the plot as accuracy{number}.png
plt.savefig(f"{base_filename_a}{number_a}.png")
plt.figure()
# Specify the base filename and starting number
base_filename_l = "loss"
number_l = 1
# Check if the file already exists
while os.path.exists(f"{base_filename_l}{number_l}.png"):
    number_l += 1
# Plot training and validation loss per epoch
plt.plot(epochs, loss, 'r', label='training')
plt.plot(epochs, val_loss, 'b', label='validation')
plt.title('Training and validation loss')
plt.legend()
# Save the plot with the incremented number
plt.savefig(f"{base_filename_l}{number_l}.png")
import os
```

```
# Define the base file name
base_filename_t = "trained"
# Initialize the number to start from
number_t = 1
# Generate the initial file name
filename = f"{base_filename_t}{number_t}.h5"
# Check if the file already exists
while os.path.exists(filename):
    # Increment the number
    number_t += 1
    # Generate the new file name
    filename = f"{base_filename_t}{number_t}.h5"
# Save the trained model to the file
model.save(filename)

import os
import glob
import re
import shutil
from PIL import Image, ImageDraw, ImageFont
# Find all the saved model files in the folder
model_files = glob.glob('trained*.h5')
# Sort the model files based on the numerical part of their names
model_files.sort(key=lambda f: int(re.sub('\D', '', f)))
# Get the latest trained model file
latest_model_file = model_files[-1]
# Load the latest trained model
model = tf.keras.models.load_model(latest_model_file)
# Display the model summary
model.summary()
class_names=['Apple___Apple_scab', 'Apple___Black_rot',
'Apple___Cedar_apple_rust', 'Apple___healthy', 'Blueberry___healthy',
'Cherry____healthy', 'Cherry____Powdery_mildew',
'Corn____Cercospora_leaf_spot_Gray_leaf_spot', 'Corn____Common_rust',
'Corn____healthy', 'Corn____Northern_Leaf_Blight', 'Grape___Black_rot',
'Grape___healthy', 'Heart_Leaf_Philodendron_citrus_greening',
'Heart_Leaf_Philodendron_healthy', 'Orange___Citrus_greening',
'Peach___Bacterial_spot', 'Peach___healthy', 'Pepper__bell___Bacterial_spot',
'Pepper__bell___healthy', 'Potato___healthy', 'Potato___Late_blight',
'Potato__Early_blight', 'Raspberry___healthy', 'Soybean___healthy',
'Squash___Powdery_mildew', 'Strawberry___healthy', 'Strawberry___Leaf_scorch'
'Tomato___Target_Spot', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato__Tomato_mosaic_virus', 'Tomato_Bacterial_spot', 'Tomato_Early_blight',
'Tomato_healthy', 'Tomato_Late_blight', 'Tomato_Leaf_Mold',
'Tomato_Septoria_leaf_spot']
```

```
test_dir = r'D:\Downloads\plant disease detection\test_sample'  # Directory path
for test images
#test_dir = r'D:\Downloads\plant disease detection\test'  # Directory path for
test images
# Create a list of file paths by searching for all files with the extension
".JPG" in the test directory
list_of_files = glob.glob(test_dir + '/*.JPG')
# Print the number of files found in the test directory
print(len(list_of_files))
# Print the test directory path
print(test_dir)
# Randomly choose a picture to test your model
img_number = int(np.random.randint(0, len(list_of_files), size=1))
test_image = list_of_files[img_number]
# Load and preprocess the image
new_image = load_image(test_image)
# Make predictions using the pre-trained model
pred = model.predict(new_image)
if len(pred) > 0:
    print(pred[0])
    # Retrieve the predicted class label
    prediction_result = class_names[np.argmax(pred[0])]
    print('prediction=', prediction_result)
    # Load the image
    img = Image.open(test_image)
    # Create a draw object
    draw = ImageDraw.Draw(img)
    # Specify the font and size
    font_size = int(20 * 0.5)  # Decrease the font size to 3/4
    font = ImageFont.truetype("arial.ttf", size=font_size)
    # Define the text and its position
    text = f"Prediction: {prediction_result}"
    text_position = (10, 10)
    # Draw the text on the image
    draw.text(text_position, text, fill=(255, 255, 255), font=font)
    # Show the image
    img.show()
    # Create the predictions folder if it doesn't exist
    predictions_folder = "predictions"
    os.makedirs(predictions_folder, exist_ok=True)
    # Save the image with the prediction in the predictions folder
    filename = os.path.basename(test_image)
    output_path = os.path.join(predictions_folder, f"prediction_{filename}")
    img.save(output_path)
    # Display the saved image
    saved_img = Image.open(output_path)
else:
    print("No predictions available for the given image.")
```

*2. Final_Capture_Calculation.ipynb*

```
# Video Processing Code
import os
import cv2
import numpy as np
# Create the test directory to save frames
test_dir = "test"
os.makedirs(test_dir, exist_ok=True)
# Delete all files in the test directory before saving frames
file_list = os.listdir(test_dir)
for file_name in file_list:
    file_path = os.path.join(test_dir, file_name)
    os.remove(file_path)
# Initialize the video capture from the webcam
video = cv2.VideoCapture(0)  # 0 indicates the default webcam
# Initialize frame counter and black frame counter
frame_count = 0
black_frame_count = 0
consecutive_black_frame_threshold = 30  # Adjust as needed
# Read frames from the webcam until a key is pressed or video is considered
blacked out
while True:
    # Read a frame from the webcam
    ret, frame = video.read()
    # Convert the frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Threshold the grayscale frame to get a binary image
    _, thresholded = cv2.threshold(gray_frame, 30, 255, cv2.THRESH_BINARY)
    # Calculate the percentage of black pixels in the thresholded image
    black_pixel_ratio = np.mean(thresholded < 10)
    # Display the frame (optional)
    cv2.imshow("Webcam", frame)
    # Save the frame as an image in the test directory if the black pixel ratio
is below a threshold
    if black_pixel_ratio < 0.9:
        frame_path = os.path.join(test_dir, f"frame_{frame_count}.jpg")
        cv2.imwrite(frame_path, frame)
        frame_count += 1
        black_frame_count = 0
    else:
        black_frame_count += 1
    # Check if consecutive black frames threshold is exceeded
    if black_frame_count >= consecutive_black_frame_threshold:
        print("Video is blacked out. Stopping capture.")
        break
    # Check for key press to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the video capture and close the display window
video.release()
cv2.destroyAllWindows()

# VPD Calculation Code
import math
# Function to calculate Vapor Pressure Deficit (VPD)
def calculate_vpd(temperature, humidity):
    # Calculate saturation vapor pressure using the temperature
    saturation_vapor_pressure = 0.611 * math.exp((17.27 * temperature) /
(temperature + 237.3))
    # Calculate actual vapor pressure using saturation vapor pressure and
humidity
    actual_vapor_pressure = saturation_vapor_pressure * (humidity / 100)
    # Calculate Vapor Pressure Deficit (VPD) as the difference between saturation
vapor pressure and actual vapor pressure
    vpd = saturation_vapor_pressure - actual_vapor_pressure
    return vpd
# Function to check if irrigation is needed based on VPD and humidity thresholds
def check_irrigation_needed(vpd_threshold, humidity_threshold):
    # Get temperature and humidity input from user
    temperature = float(input("Temperature (in degrees Celsius): "))
    humidity = float(input("Humidity (in percentage): "))
    # Calculate VPD using the provided temperature and humidity
    vpd = calculate_vpd(temperature, humidity)
    # Check if irrigation is needed based on VPD and humidity thresholds
    if vpd > vpd_threshold and humidity < humidity_threshold:
        print("Irrigation is needed for the leaves.")
    else:
        print("No irrigation is needed for the leaves.")
# Define the VPD and humidity thresholds (adjust these values based on plant
requirements)
vpd_threshold = 1.5
humidity_threshold = 60
# Call the function to check if irrigation is needed
check_irrigation_needed(vpd_threshold, humidity_threshold)
```

3. *hardware codes/AHT.py*

```python
import time
import board
import busio
import adafruit_ahtx0
import csv
import math
def capture_humidity_data():
    #Define pins 27(SDA)/28(SCL) on the jetson nano for i2c communication
    scl = board.SCL_1
    sda = board.SDA_1
    #establish connection with AHT10 sensor using i2c Bus 0
    i2c = busio.I2C(scl,sda)
    sensor = adafruit_ahtx0.AHTx0(i2c)
        # Create a CSV file and write headers
    csv_file = open('humidity_data.csv', 'w')
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(['Timestamp', 'Humidity' , 'Air Temp (C)' , 'VPD'])
    while True:
            # Read humidity & Temperature data from the DHT22 sensor
        humidity = sensor.relative_humidity
        print("RH   (%):",humidity)
        temperature = sensor.temperature
        print("Temp (C):",temperature)
        # Calculate the environmental VPD
        saturation_vapor_pressure = 0.611 * math.exp((17.27 * temperature) /
(temperature + 237.3))
        actual_vapor_pressure = saturation_vapor_pressure * (humidity / 100)
        vpd_env = saturation_vapor_pressure - actual_vapor_pressure
        if humidity is not None:
            # Get the current timestamp
            timestamp = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())
            # Write the timestamp and humidity to the CSV file
            csv_writer.writerow([timestamp, humidity, temperature, vpd_env ])
            csv_file.flush()
        # Wait for a few seconds before reading again
        time.sleep(2)
    # Close the CSV file
    csv_file.close()
if __name__ == '__main__':
        capture_humidity_data()
```

4. *hardware codes/Lepton 3.5/opencv-capture.py*

```python
#!/usr/bin/env python
"""
CV2 video capture example from Pure Thermal 1
"""
try:
    import cv2
except ImportError:
    print("ERROR python-opencv must be installed")
    exit(1)
class OpenCvCapture(object):
    """
    Encapsulate state for capture from Pure Thermal 1 with OpenCV
    """
    def __init__(self):
        # Initialize OpenCV video capture object with the device index 2
        cv2_cap = cv2.VideoCapture(2)
        # Check if the camera is opened successfully
        if not cv2_cap.isOpened():
            print("Camera not found!")
            exit(1)
        self.cv2_cap = cv2_cap
    def show_video(self):
        """
        Run loop for cv2 capture from lepton
        """
        # Create a named window for displaying the video
        cv2.namedWindow("lepton", cv2.WINDOW_NORMAL)
        print("Running, ESC or Ctrl-c to exit...")
        while True:
            # Read a frame from the video capture
            ret, img = self.cv2_cap.read()
            # Check if the frame is read successfully
            if ret == False:
                print("Error reading image")
                break
            # Display the resized frame in the named window
            cv2.imshow("lepton", cv2.resize(img, (640, 480)))
            # Break the loop if ESC key is pressed
            if cv2.waitKey(5) == 27:
                break
        # Close all OpenCV windows
        cv2.destroyAllWindows()
# Create an instance of the OpenCvCapture class and start showing the video
if __name__ == '__main__':
    OpenCvCapture().show_video()
```

5. *hardware codes/Lepton 3.5/PT2_Lep_Temp_Array.py*

```
"""
The simplest example of using PureThemal2 board:
- reads the USB UVC device
- prints the resulting numpy array
"""
import time
import cv2
import numpy as np
from queue import Queue
# IR camera
from uvctypes import *
BUF_SIZE = 2
q = Queue(BUF_SIZE)
def py_frame_callback(frame, userptr):
    # Convert data to numpy array
    array_pointer = cast(
        frame.contents.data,
        POINTER(c_uint16 * (frame.contents.width * frame.contents.height)),
    )
    data = np.frombuffer(array_pointer.contents,
dtype=np.dtype(np.uint16)).reshape(
        frame.contents.height, frame.contents.width
    )
    # no copy
    # data = np.fromiter(
    #   frame.contents.data, dtype=np.dtype(np.uint8),
count=frame.contents.data_bytes
    # ).reshape(
    #   frame.contents.height, frame.contents.width, 2
    # ) # copy
    # If data length is not as expected, return
    if frame.contents.data_bytes != (2 * frame.contents.width *
frame.contents.height):
        return
    # Put the data in the queue if it's not full
    if not q.full():
        q.put(data)
# Define the frame callback function as a C function pointer
PTR_PY_FRAME_CALLBACK = CFUNCTYPE(None, POINTER(uvc_frame),
c_void_p)(py_frame_callback)
def ktoc(val):
    # Convert temperature from Kelvin to Celsius
    return (val - 27315) / 100.0
def setup():
    # Initialize the UVC context, device, and device handle
    ctx = POINTER(uvc_context)()
    dev = POINTER(uvc_device)()
    devh = POINTER(uvc_device_handle)()
    ctrl = uvc_stream_ctrl()
    # Initialize the UVC context
    res = libuvc.uvc_init(byref(ctx), 0)
    if res < 0:
        print("uvc_init error")
        exit(res)
    # Find the UVC device
    res = libuvc.uvc_find_device(ctx, byref(dev), PT_USB_VID, PT_USB_PID, 0)
    if res < 0:
        print("uvc_find_device error")
        exit(res)
    # Open the UVC device
    res = libuvc.uvc_open(dev, byref(devh))
    if res < 0:
        print(f"uvc_open error {res}")
        exit(res)
    print("device opened!")
    # Print device information
    print_device_info(devh)
    # Print device formats
    print_device_formats(devh)
    # Get frame formats for Y16 format
    frame_formats = uvc_get_frame_formats_by_guid(devh, VS_FMT_GUID_Y16)
    if len(frame_formats) == 0:
        print("device does not support Y16")
        exit(1)
    # Get stream control format and size
    libuvc.uvc_get_stream_ctrl_format_size(
        devh,
        byref(ctrl),
        UVC_FRAME_FORMAT_Y16,
        frame_formats[0].wWidth,
        frame_formats[0].wHeight,
        int(1e7 / frame_formats[0].dwDefaultFrameInterval),
    )
    # Start streaming
    res = libuvc.uvc_start_streaming(
        devh, byref(ctrl), PTR_PY_FRAME_CALLBACK, None, 0
    )
    if res < 0:
        print("uvc_start_streaming failed: {0}".format(res))
        exit(1)
    return ctx, dev, devh, ctrl
if __name__ == "__main__":
    # Setup the UVC device
    ctx, dev, devh, ctrl = setup()
    while True:
        # Get data from the queue
        data = q.get(True, 500)
        if data is None:
            break
        # Convert data to temperature map
        temp_map = ktoc(data)
        # Print the temperature map
        print(temp_map)
    # Stop streaming and release resources
    libuvc.uvc_stop_streaming(devh)
    libuvc.uvc_unref_device(dev)
    libuvc.uvc_exit(ctx)
```

6. *hardware codes/Lepton 3.5/uvc-deviceinfo.py*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from uvctypes import *
def main():
  # Initialize UVC context, device, device handle, and control structure
  ctx = POINTER(uvc_context)()
  dev = POINTER(uvc_device)()
  devh = POINTER(uvc_device_handle)()
  ctrl = uvc_stream_ctrl()
  # Initialize UVC library
  res = libuvc.uvc_init(byref(ctx), 0)
  if res < 0:
    print("uvc_init error")
    exit(1)
  try:
    # Find UVC device
    res = libuvc.uvc_find_device(ctx, byref(dev), PT_USB_VID, PT_USB_PID, 0)
    if res < 0:
      print("uvc_find_device error")
      exit(1)
    try:
      res = libuvc.uvc_open(dev, byref(devh))
      if res < 0:
        print("uvc_open error")
        exit(1)
      # Print device information
      print_device_info(devh)
    finally:
      # Release UVC device reference
      libuvc.uvc_unref_device(dev)
  finally:
    # Exit UVC context
    libuvc.uvc_exit(ctx)\
if __name__ == '__main__':
  main()
```

7. *hardware codes/Lepton 3.5/uvc-radiometry.py*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Import necessary libraries
from uvctypes import *
import time
import cv2
import numpy as np
try:
  from queue import Queue
except ImportError:
  from Queue import Queue
import platform
# Set buffer size and create a queue
BUF_SIZE = 2
q = Queue(BUF_SIZE)
# Define the frame callback function
def py_frame_callback(frame, userptr):
  # Get the array pointer from the frame data
  array_pointer = cast(frame.contents.data, POINTER(c_uint16 *
(frame.contents.width * frame.contents.height)))
  data = np.frombuffer(
    array_pointer.contents, dtype=np.dtype(np.uint16)
  ).reshape(
    frame.contents.height, frame.contents.width
  )
  # no copy
  # data = np.fromiter(
  #   frame.contents.data, dtype=np.dtype(np.uint8),
count=frame.contents.data_bytes
```

```
  # ).reshape(
  #   frame.contents.height, frame.contents.width, 2
  # ) # copy
  # Check if the frame data size is valid
  if frame.contents.data_bytes != (2 * frame.contents.width *
frame.contents.height):
    return
  # Put the data in the queue if it is not full
  if not q.full():
    q.put(data)
# Define the pointer to the frame callback function
PTR_PY_FRAME_CALLBACK = CFUNCTYPE(None, POINTER(uvc_frame),
c_void_p)(py_frame_callback)
# Function to convert temperature from Kelvin to Fahrenheit
def ktof(val):
  return (1.8 * ktoc(val) + 32.0)
# Function to convert temperature from Kelvin to Celsius
def ktoc(val):
  return (val - 27315) / 100.0
# Function to convert raw data to 8-bit representation
def raw_to_8bit(data):
  cv2.normalize(data, data, 0, 65535, cv2.NORM_MINMAX)
  np.right_shift(data, 8, data)
  return cv2.cvtColor(np.uint8(data), cv2.COLOR_GRAY2RGB)
# Function to display temperature on the image
def display_temperature(img, val_k, loc, color):
  val = ktof(val_k)
  cv2.putText(img,"{0:.1f} degF".format(val), loc, cv2.FONT_HERSHEY_SIMPLEX,
0.75, color, 2)
  x, y = loc
  cv2.line(img, (x - 2, y), (x + 2, y), color, 1)
  cv2.line(img, (x, y - 2), (x, y + 2), color, 1)
# Main function
def main():
  # Initialize the context, device, and device handle
  ctx = POINTER(uvc_context)()
  dev = POINTER(uvc_device)()
  devh = POINTER(uvc_device_handle)()
  ctrl = uvc_stream_ctrl()
   # Initialize the UVC context
  res = libuvc.uvc_init(byref(ctx), 0)
  if res < 0:
    print("uvc_init error")
    exit(1)
  # Find the UVC device
  try:
    res = libuvc.uvc_find_device(ctx, byref(dev), PT_USB_VID, PT_USB_PID, 0)
    if res < 0:
      print("uvc_find_device error")
      exit(1)
    # Open the UVC device
    try:
      res = libuvc.uvc_open(dev, byref(devh))
      if res < 0:
        print("uvc_open error")
        exit(1)
      print("device opened!")
      # Print device info and formats
      print_device_info(devh)
      print_device_formats(devh)
      # Get frame formats
      frame_formats = uvc_get_frame_formats_by_guid(devh, VS_FMT_GUID_Y16)
      if len(frame_formats) == 0:
        print("device does not support Y16")
        exit(1)
      # Get stream control format and size
      libuvc.uvc_get_stream_ctrl_format_size(devh, byref(ctrl),
UVC_FRAME_FORMAT_Y16,
        frame_formats[0].wWidth, frame_formats[0].wHeight, int(1e7 /
frame_formats[0].dwDefaultFrameInterval)
      )

      res = libuvc.uvc_start_streaming(devh, byref(ctrl), PTR_PY_FRAME_CALLBACK,
None, 0)
      if res < 0:
        print("uvc_start_streaming failed: {0}".format(res))
        exit(1)
      # Start streaming
      try:
        while True:
          data = q.get(True, 500)
          if data is None:
            break
          # Resize the data and perform analysis
          data = cv2.resize(data[:,:], (640, 480))
          minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(data)
          img = raw_to_8bit(data)
          display_temperature(img, minVal, minLoc, (255, 0, 0))
          display_temperature(img, maxVal, maxLoc, (0, 0, 255))
          cv2.imshow('Lepton Radiometry', img)
          cv2.waitKey(1)
        cv2.destroyAllWindows()
      finally:
        # Stop streaming
        libuvc.uvc_stop_streaming(devh)
      print("done")
    finally:
      # Release the device
      libuvc.uvc_unref_device(dev)
  finally:
    # Exit the UVC context
    libuvc.uvc_exit(ctx)
if __name__ == '__main__':
  main()
```

*8. hardware codes/Lepton 3.5/uvc-types.py*

```
from ctypes import *
import platform
# Load the libuvc library based on the platform
try:
  if platform.system() == 'Darwin':
    libuvc = cdll.LoadLibrary("libuvc.dylib")
  elif platform.system() == 'Linux':
    libuvc = cdll.LoadLibrary("libuvc.so")
  else:
    libuvc = cdll.LoadLibrary("libuvc")
except OSError:
  print("Error: could not find libuvc!")
  exit(1)
# Define ctypes structures representing various UVC entities
class uvc_context(Structure):
  _fields_ = [("usb_ctx", c_void_p),
              ("own_usb_ctx", c_uint8),
              ("open_devices", c_void_p),
              ("handler_thread", c_ulong),
              ("kill_handler_thread", c_int)]
class uvc_device(Structure):
  _fields_ = [("ctx", POINTER(uvc_context)),
              ("ref", c_int),
              ("usb_dev", c_void_p)]
class uvc_stream_ctrl(Structure):
  _fields_ = [("bmHint", c_uint16),
              ("bFormatIndex", c_uint8),
              ("bFrameIndex", c_uint8),
              ("dwFrameInterval", c_uint32),
              ("wKeyFrameRate", c_uint16),
              ("wPFrameRate", c_uint16),
              ("wCompQuality", c_uint16),
              ("wCompWindowSize", c_uint16),
              ("wDelay", c_uint16),
              ("dwMaxVideoFrameSize", c_uint32),
              ("dwMaxPayloadTransferSize", c_uint32),
              ("dwClockFrequency", c_uint32),
              ("bmFramingInfo", c_uint8),
              ("bPreferredVersion", c_uint8),
              ("bMinVersion", c_uint8),
              ("bMaxVersion", c_uint8),
              ("bInterfaceNumber", c_uint8)]
class uvc_format_desc(Structure):
  pass
class uvc_frame_desc(Structure):
  pass
uvc_frame_desc._fields_ = [
              ("parent", POINTER(uvc_format_desc)),
              ("prev", POINTER(uvc_frame_desc)),
              ("next", POINTER(uvc_frame_desc)),
              # /** Type of frame, such as JPEG frame or uncompressed frme */
              ("bDescriptorSubtype", c_uint), # enum uvc_vs_desc_subtype
bDescriptorSubtype;
              # /** Index of the frame within the list of specs available for
this format */
              ("bFrameIndex", c_uint8),
              ("bmCapabilities", c_uint8),
              # /** Image width */
              ("wWidth", c_uint16),
              # /** Image height */
              ("wHeight", c_uint16),
              # /** Bitrate of corresponding stream at minimal frame rate */
              ("dwMinBitRate", c_uint32),
              # /** Bitrate of corresponding stream at maximal frame rate */
              ("dwMaxBitRate", c_uint32),
              # /** Maximum number of bytes for a video frame */
              ("dwMaxVideoFrameBufferSize", c_uint32),
              # /** Default frame interval (in 100ns units) */
              ("dwDefaultFrameInterval", c_uint32),
              # /** Minimum frame interval for continuous mode (100ns units) */
              ("dwMinFrameInterval", c_uint32),
              # /** Maximum frame interval for continuous mode (100ns units) */
              ("dwMaxFrameInterval", c_uint32),
              # /** Granularity of frame interval range for continuous mode
(100ns) */
              ("dwFrameIntervalStep", c_uint32),
              # /** Frame intervals */
              ("bFrameIntervalType", c_uint8),
              # /** number of bytes per line */
              ("dwBytesPerLine", c_uint32),
              # /** Available frame rates, zero-terminated (in 100ns units) */
              ("intervals", POINTER(c_uint32))]
uvc_format_desc._fields_ = [
              ("parent", c_void_p),
              ("prev", POINTER(uvc_format_desc)),
              ("next", POINTER(uvc_format_desc)),
              # /** Type of image stream, such as JPEG or uncompressed. */
              ("bDescriptorSubtype", c_uint), # enum uvc_vs_desc_subtype
bDescriptorSubtype;
              # /** Identifier of this format within the VS interface's format
list */
              ("bFormatIndex", c_uint8),
              ("bNumFrameDescriptors", c_uint8),
              # /** Format specifier */
              ("guidFormat", c_char * 16), # union { uint8_t guidFormat[16];
uint8_t fourccFormat[4]; }
              # /** Format-specific data */
              ("bBitsPerPixel", c_uint8),
              # /** Default {uvc_frame_desc} to choose given this format */
              ("bDefaultFrameIndex", c_uint8),
              ("bAspectRatioX", c_uint8),
              ("bAspectRatioY", c_uint8),
              ("bmInterlaceFlags", c_uint8),
              ("bCopyProtect", c_uint8),
              ("bVariableSize", c_uint8),
              # /** Available frame specifications for this format */
              ("frame_descs", POINTER(uvc_frame_desc))]
class timeval(Structure):
  _fields_ = [("tv_sec", c_long), ("tv_usec", c_long)]
class uvc_frame(Structure):
  _fields_ = [# /** Image data for this frame */
              ("data", POINTER(c_uint8)),
              # /** Size of image data buffer */
              ("data_bytes", c_size_t),
              # /** Width of image in pixels */
              ("width", c_uint32),
              # /** Height of image in pixels */
              ("height", c_uint32),
              # /** Pixel data format */
              ("frame_format", c_uint), # enum uvc_frame_format frame_format
              # /** Number of bytes per horizontal line (undefined for compressed
format) */
              ("step", c_size_t),
              # /** Frame number (may skip, but is strictly monotonically
increasing) */
              ("sequence", c_uint32),
              # /** Estimate of system time when the device started capturing the
image */
              ("capture_time", timeval),
              # /** Handle on the device that produced the image.
              #  * @warning You must not call any uvc_* functions during a
callback. */
              ("source", POINTER(uvc_device)),
              # /** Is the data buffer owned by the library?
              #  * If 1, the data buffer can be arbitrarily reallocated by frame
conversion
              #  * functions.
              #  * If 0, the data buffer will not be reallocated or freed by the
library.
              #  * Set this field to zero if you are supplying the buffer.
              #  */
              ("library_owns_data", c_uint8)]
class uvc_device_handle(Structure):
  _fields_ = [("dev", POINTER(uvc_device)),
              ("prev", c_void_p),
              ("next", c_void_p),
              ("usb_devh", c_void_p),
              ("info", c_void_p),
              ("status_xfer", c_void_p),
              ("status_buf", c_ubyte * 32),
              ("status_cb", c_void_p),
              ("status_user_ptr", c_void_p),
              ("button_cb", c_void_p),
              ("button_user_ptr", c_void_p),
              ("streams", c_void_p),
              ("is_isight", c_ubyte)]
class lep_oem_sw_version(Structure):
  _fields_ = [("gpp_major", c_ubyte),
              ("gpp_minor", c_ubyte),
              ("gpp_build", c_ubyte),
              ("dsp_major", c_ubyte),
              ("dsp_minor", c_ubyte),
              ("dsp_build", c_ubyte),
              ("reserved", c_ushort)]
# Define ctypes function prototypes for calling extension unit functions
def call_extension_unit(devh, unit, control, data, size):
  return libuvc.uvc_get_ctrl(devh, unit, control, data, size, 0x81)
def set_extension_unit(devh, unit, control, data, size):
  return libuvc.uvc_set_ctrl(devh, unit, control, data, size, 0x81)
# Define constants for USB vendor and product IDs, frame formats, and GUIDs
PT_USB_VID = 0x1e4e
PT_USB_PID = 0x0100
AGC_UNIT_ID = 3
OEM_UNIT_ID = 4
RAD_UNIT_ID = 5
SYS_UNIT_ID = 6
VID_UNIT_ID = 7
UVC_FRAME_FORMAT_UYVY = 4
UVC_FRAME_FORMAT_I420 = 5
UVC_FRAME_FORMAT_RGB = 7
UVC_FRAME_FORMAT_BGR = 8
UVC_FRAME_FORMAT_Y16 = 13
VS_FMT_GUID_GREY = create_string_buffer(
    b"Y8  \x00\x00\x10\x00\x80\x00\x00\xaa\x00\x38\x9b\x71", 16
)
VS_FMT_GUID_Y16 = create_string_buffer(
    b"Y16 \x00\x00\x10\x00\x80\x00\x00\xaa\x00\x38\x9b\x71", 16
)
VS_FMT_GUID_YUYV = create_string_buffer(
    b"UYVY\x00\x00\x10\x00\x80\x00\x00\xaa\x00\x38\x9b\x71", 16
)
VS_FMT_GUID_NV12 = create_string_buffer(
    b"NV12\x00\x00\x10\x00\x80\x00\x00\xaa\x00\x38\x9b\x71", 16
)
VS_FMT_GUID_YU12 = create_string_buffer(
    b"I420\x00\x00\x10\x00\x80\x00\x00\xaa\x00\x38\x9b\x71", 16
)
VS_FMT_GUID_BGR3 = create_string_buffer(
    b"\x7d\xeb\x36\xe4\x4f\x52\xce\x11\x9f\x53\x00\x20\xaf\x0b\xa7\x70", 16
)
VS_FMT_GUID_RGB565 = create_string_buffer(
    b"RGBP\x00\x00\x10\x00\x80\x00\x00\xaa\x00\x38\x9b\x71", 16
)
# Set the restype for the uvc_get_format_descs function
libuvc.uvc_get_format_descs.restype = POINTER(uvc_format_desc)
# Define a function to print device information
def print_device_info(devh):
  vers = lep_oem_sw_version()
  call_extension_unit(devh, OEM_UNIT_ID, 9, byref(vers), 8)
  print("Version gpp: {0}.{1}.{2} dsp: {3}.{4}.{5}".format(
    vers.gpp_major, vers.gpp_minor, vers.gpp_build,
    vers.dsp_major, vers.dsp_minor, vers.dsp_build,
  ))
  flir_pn = create_string_buffer(32)
  call_extension_unit(devh, OEM_UNIT_ID, 8, flir_pn, 32)
  print("FLIR part #: {0}".format(flir_pn.raw))
  flir_sn = create_string_buffer(8)
  call_extension_unit(devh, SYS_UNIT_ID, 3, flir_sn, 8)
  print("FLIR serial #: {0}".format(repr(flir_sn.raw)))
# Define iterator functions to iterate over format descriptors and frame
descriptors
def uvc_iter_formats(devh):
  p_format_desc = libuvc.uvc_get_format_descs(devh)
  while p_format_desc:
    yield p_format_desc.contents
    p_format_desc = p_format_desc.contents.next
def uvc_iter_frames_for_format(devh, format_desc):
  p_frame_desc = format_desc.frame_descs
  while p_frame_desc:
    yield p_frame_desc.contents
    p_frame_desc = p_frame_desc.contents.next
# Define a function to print device formats
def print_device_formats(devh):
  for format_desc in uvc_iter_formats(devh):
    print("format: {0}".format(format_desc.guidFormat[0:4]))
    for frame_desc in uvc_iter_frames_for_format(devh, format_desc):
      print("  frame {0}x{1} @ {2}fps".format(frame_desc.wWidth,
frame_desc.wHeight, int(1e7 / frame_desc.dwDefaultFrameInterval)))
# Define a function to get frame formats based on a GUID
def uvc_get_frame_formats_by_guid(devh, vs_fmt_guid):
  for format_desc in uvc_iter_formats(devh):
    if vs_fmt_guid[0:4] == format_desc.guidFormat[0:4]:
      return [fmt for fmt in uvc_iter_frames_for_format(devh, format_desc)]
  return []
```