

Development of a Real-Time Image Capture and Processing Pipeline for Plant and Soil Identification in Agricultural Fields

Abstract

This report presents the development of a real-time image capture and processing pipeline for detecting plants and soil in agricultural environments. Utilizing Raspberry Pi cameras and a Mask R-CNN model, the pipeline was designed to provide regular field snapshots, accessible for review via a web interface. Cloud-based storage solutions were employed for handling vast amounts of image data, while machine learning algorithms were leveraged for image analysis and object detection.

Two different models were explored during the project: an AutoML Vision model provided by Google Cloud Platform (GCP) and a locally trained Mask R-CNN model. The AutoML Vision model delivered a precision of 71% and a recall of 32%, indicating its ability to correctly predict and identify objects in the images. However, due to financial constraints, a transition was made to a custom Mask R-CNN model, which was trained using resources provided by Datature. The Mask R-CNN model reported a total loss of 1.46 for the training set and 6.8 for the validation set, suggesting better performance on the training set but potential overfitting. The classification, localization, and mask losses were 0.24, 0.68, and 5.72 respectively, reflecting the model's ability to predict the class probabilities, bounding box coordinates, and pixel-wise segmentation masks.

The project's main achievement is the integration of hardware and software components into a functioning pipeline capable of capturing field images, processing them in real-time, and providing accessible and interpretable results, paving the way for efficient and real-time agricultural monitoring..

Introduction

The rapid advancement of technology has profoundly influenced various sectors, and agriculture is no exception. From crop monitoring systems to automatic irrigation setups,

technology has immensely contributed to more effective and efficient agricultural practices. In the context of such technological integration in agriculture, this project set out to construct an automated, real-time image capture and processing pipeline, which can significantly facilitate the ongoing monitoring of agricultural fields.

This project's primary motivation was to address the prevailing challenges in real-time agricultural monitoring, which often involves laborious and time-consuming manual surveillance. Developing an automated system can greatly enhance the monitoring efficiency, enabling faster detection of any anomalies and thereby aiding in timely intervention. In this regard, the implementation of machine learning models for image analysis is instrumental, as they can accurately identify and classify various elements in the field, such as differentiating between plant and soil regions.

In the process of designing this pipeline, Raspberry Pi cameras were chosen as the hardware component for their ease of setup, high flexibility, and low cost, making them an optimal choice for a proof-of-concept prototype. On the software side, two models were evaluated: the AutoML Vision model from Google Cloud Platform (GCP) and a custom-trained Mask R-CNN model. The selection criteria involved assessing the model's precision, recall, and overall performance with the available resources and project requirements. Despite satisfactory performance from the GCP AutoML Vision model, it was deemed financially unsustainable for the project, leading to the adoption of the Mask R-CNN model. Mask R-CNN was chosen for its robust capabilities in instance segmentation tasks, proving its effectiveness in differentiating intricate details within images, which was critical for this project.

The pipeline aimed to provide updated field snapshots that could be viewed and evaluated through a web interface. The evaluation in this context refers to the visual assessment of the processed images that delineate plant and soil regions. It enables real-time tracking of the field's state, facilitating immediate identification of any areas of concern.

Overall, this project aspired to integrate hardware and software components into a functioning system capable of delivering real-time agricultural field monitoring.

Methodology

The methodology applied in this project was comprehensive and multifaceted, incorporating several stages of data handling, analysis, model training, and deployment.

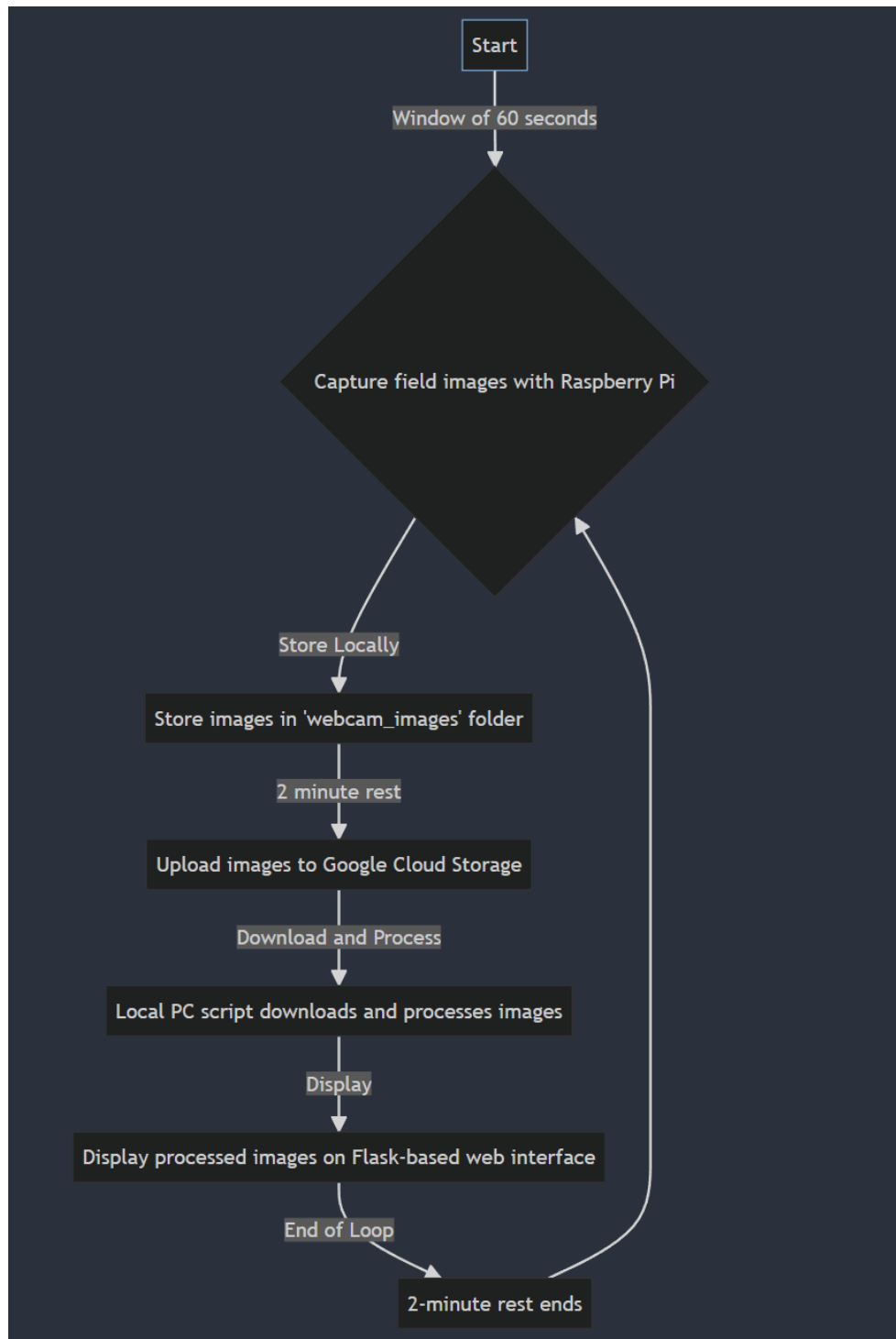


Fig 1 The basic flowchart for the methodology

Phase 1: Raspberry Pi Testing and Literature Review

In the initial month, the project commenced with the testing of all Raspberry Pi units. This stage was crucial to ensure the proper functioning of the hardware. A detailed literature review was carried out during this period. The review included studying 20 research papers that provided critical insights and theoretical grounding for the project.

The data extraction from the Raspberry Pi units was a complex and arduous process, especially without access to an HDMI monitor for the first 17-19 days. During this time, the project's workspace's projector was used for data extraction, necessitating numerous rounds of device connections and disconnections. (Even for using VNC we need to connect the the pi to a screen for the first time). Despite these challenges, a significant volume of clear field images, crucial for subsequent stages of the project, was successfully retrieved.

Phase 2: Data Handling and Raspberry Pi Configuration

The second stage involved further categorization of Raspberry Pi units into "masters" and "slaves" based on the configurations of their files. This stage was deemed necessary as the initial idea was to use a hard drive for uploading the batches of photos taken by each Raspberry Pi unit.

One paper titled "Internet of Things and Machine Learning based Intelligent Irrigation System for Agriculture" (DOI - Internet of Things and Machine Learning based Intelligent Irrigation System for Agriculture | IEEE Conference Publication | IEEE Xplore) significantly influenced the decision to use Google Cloud for simultaneous image upload and analysis.

Phase 3: Google Cloud and AutoML Vision Model Exploration

The Google Cloud Platform (GCP) was subsequently explored, leading to the discovery of the Auto Vision service. A new GCP account was created, utilizing the \$300 free credit to employ the Google Storage Blob service and the Auto Vision/Auto ML service.

These are the steps followed-

1. Creating a Student Account - A student account was created on GCP, providing a \$300 credit to be used for various services available on the platform.
2. Uploading Images to Cloud Storage - A Google Cloud Storage bucket was created to store all the images used in this project. The images were uploaded to the bucket, which serves as a centralized storage location for the dataset.
3. Connecting Vertex AI to Google Cloud Storage - The Google Cloud Storage bucket was connected to Vertex AI, enabling the platform to access the images for further processing and model training.
4. Labeling the Dataset - The images were labeled using bounding boxes to identify objects within the images. Two categories were used for labeling: plant and soil.
5. Splitting the Dataset - The dataset was split into training and validation sets. This division allows the model to be trained on one subset of the data and validated on another, ensuring a robust evaluation of the model's performance.
6. Training the Object Detection Model - The object detection model was trained using the custom dataset and Google's pre-trained AutoML model. The AutoML model employs transfer learning, which helps in training the model more quickly and accurately.

A custom model option was available but not chosen for this project. This alternative would have allowed for the creation of custom models from scratch, leveraging Google's computing resources for training.

7. Configuring the Model Training Process - Before starting the training process, the number of nodes required for the model was selected. In this case, one node was chosen.

8. Evaluating the Model - After the training was complete, the model's performance was evaluated using precision and recall scores. These metrics help assess the quality of the object detection model and its ability to accurately identify the categories of interest.

9. Deploying the Model - The trained object detection model was deployed using Google Model Registry Service. This deployment allowed for testing the model on random images, further evaluating its performance in real-world scenarios.

The Auto Vision model was trained using manually labelled images, a task requiring the laborious labelling of 100 images.

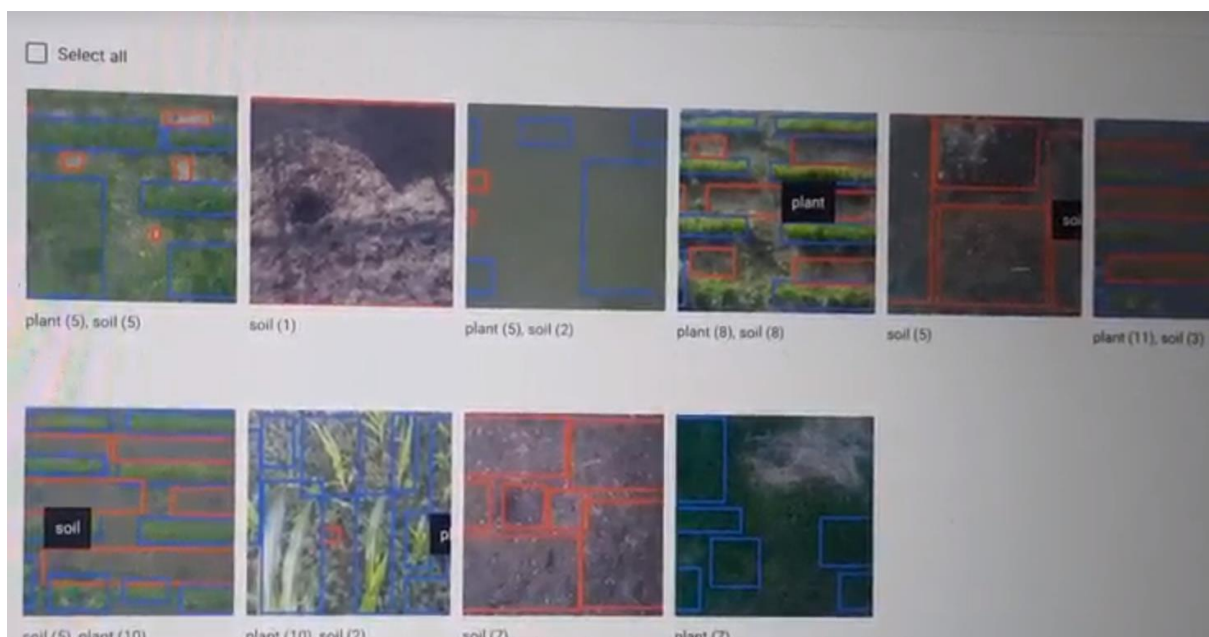


Fig 2 Labelled images using gcp labelling service for training the gcp model, blue represents plant and red represents soil

However, it was later realized that the Auto ML service wasn't economically viable due to the swift depletion of the free GCP credit.

Results of GCP Vision model:

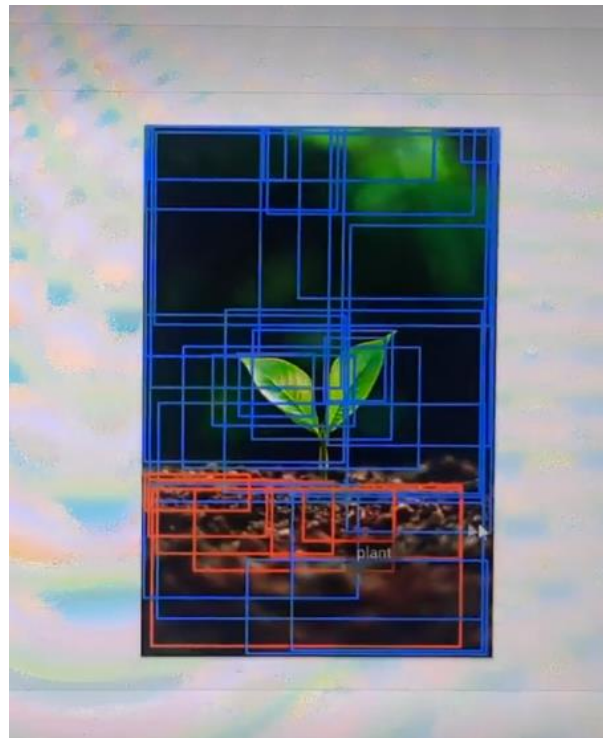


fig 3 Output image of GCP vision model, the blue boxes are predicting the plants and the red boxes are showing the soils. (if we increase the threshold confidence score the number of boxes will go down)

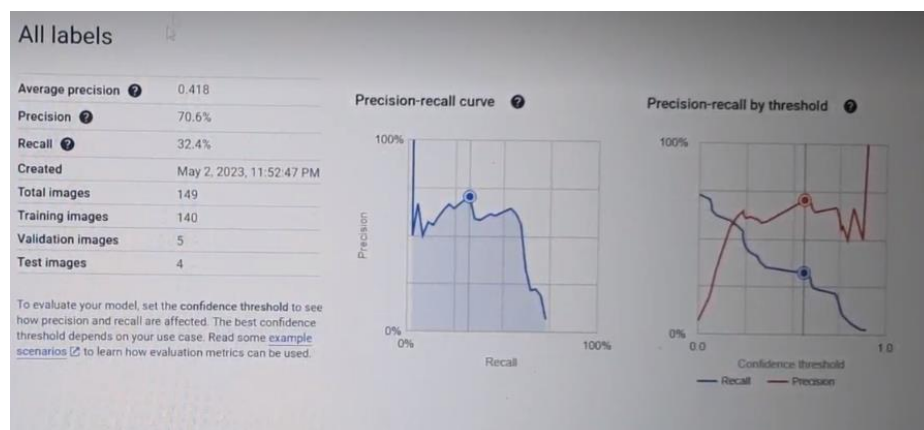


fig 4 Model metrics of the gcp model

Model Evaluation of the GCP vision model:

Precision: With a precision score of 71%, this indicates that when my model predicts an object to be either a plant or soil, it is correct 71% of the time. In other words, 71% of the objects identified by the model are true positives (correctly identified), while the remaining 29% are false positives (incorrectly identified).

Recall: With a recall score of 32%, this means that my model is able to correctly identify only 32% of the actual objects (plants or soil) in the images. In other words, out of all the true objects present in the images, the model is able to detect 32% of them, while missing the remaining 68%.

Phase 4: Model Training and Evaluation with Local Resources

Given the economical constraints of GCP, the project then shifted towards local resources for the training and prediction of the Mask R-CNN model. The Matterport Mask R-CNN tutorial [GitHub - matterport/Mask_RCNN: Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow](https://github.com/matterport/Mask_RCNN) was initially used for this purpose, but difficulties with local RAM and GPU constraints hindered progress.

Alternative platforms like Colab Pro were explored but didn't yield the desired results. Subsequently, the website Datature was discovered, which provided powerful GPUs and 300 minutes of free training time. It required the relabelling of 133 images as the previously labelled images from GCP could not be transferred to this new platform. A service called makesense.ai was used for this labelling task.



fig 5 Annotated images for the Mask Rcn model where green represents soil and blue represents plant

After training,

the model and its associated weights were downloaded from Datature in the form of a .pb file. This phase also involved a detailed evaluation of the trained model. The overall model performance was evaluated using different metrics such as total loss, classification loss, localization loss, and mask loss.

The **total loss** for the training set was recorded as **1.46**, while the validation set exhibited a significantly higher total loss of 6.8. These values suggested that the model was **overfitting on the training data**.

The total loss is a combination of multiple loss components. It is the sum of the classification loss, localization loss, and mask loss.

Total Loss = Classification Loss + Localization Loss + Mask Loss



fig 6 showing the total loss of the trained mask rcnn model

The **classification loss** measures the model's effectiveness in correctly identifying whether a given pixel belongs to the plant or the soil class. Lower classification loss denotes better model performance as it suggests that the model's predictions are closer to the actual labels.

For the training and validation sets the classification loss were **0.24** and **0.48**, respectively, indicating superior model performance on the training set compared to the validation set. This was **another indication of overfitting**.

To mitigate this, various strategies could be employed in future iterations of the project. One approach could be gathering more diverse data for training so the model learns to generalize better. Regularization techniques such as dropout or L1/L2 regularization can also be used to prevent overfitting. Additionally, using data augmentation techniques, such as random rotations, scaling, and cropping, can help the model generalize better to unseen data.

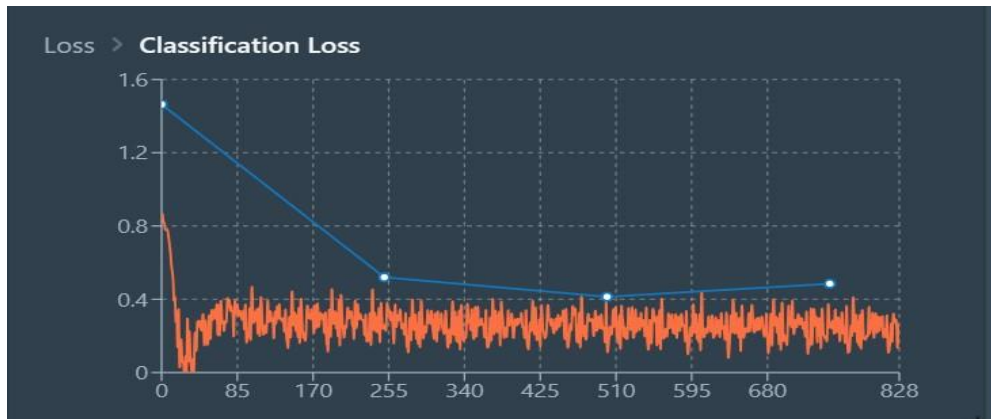


fig 7 showing the classification loss of the trained mask rcnn model

The **localization loss**, refers to how accurately our trained Mask R-CNN model predicts the position and size of the bounding boxes that encapsulate the detected objects - in this case, plants and soil patches - in an image

The localization loss was 0.68 which indicates some discrepancies between the predicted bounding boxes and the actual regions of plants and soil.

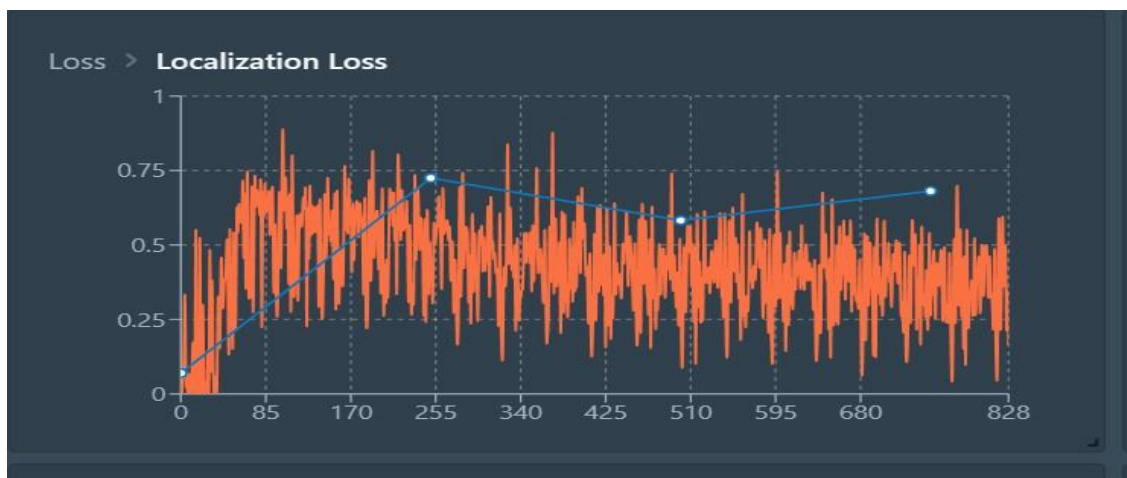


fig 8 showing the localization loss of the trained mask rcnn model

These discrepancies might result from various factors such as illumination differences, occlusions, or even the diversity in the shapes and sizes of the plants and soil patches in the field.

The **mask loss**, corresponds to how accurately our trained Mask R-CNN model can predict pixel-wise segmentation masks for the detected objects, namely the regions in the image containing plants and soil.

In this project, a mask loss of 5.72 signifies a relatively high discrepancy between the model's predicted segmentation masks and the true mask

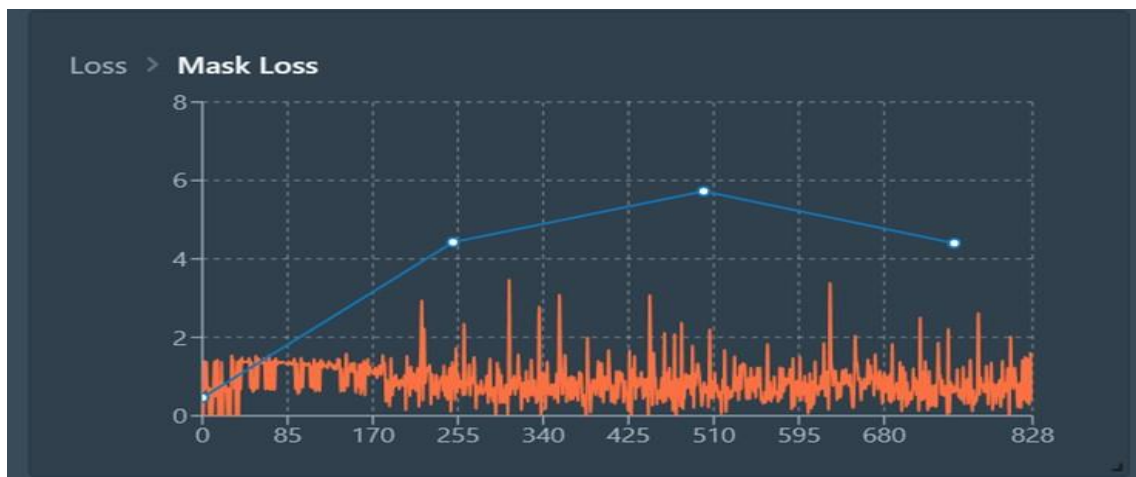


fig 9 showing the mask loss of the trained mask rcnn model

Phase 5: Raspberry Pi Scripts and Flask-based Interface Development

During the model training and evaluation stage, scripts were also being developed for the Raspberry Pi units. These scripts enabled the Raspberry Pis to capture images of the field at a one-minute window. The images, named in a specific format ("rasp{raspberry pi number}_{picture number}"), were saved locally on the Raspberry Pi devices (***Stores all those***

images in a local folder called webcam_images which lies inside the local raspberry folder called Test.

The Test folder also contains the myscriptmodified.py file which will make all this happen and the google cloud api key which is basically a json file) .

After that period of 1 minute all the raspberry pis will take a rest for 2minutes

in that resting period of 2 minute all the raspberry pis will upload their batches of photos present in their local folder to google cloud with the same name.

so suppose there are 5 raspberry pis and each of them takes 5 photos in that 1 minute window

so after every 1 minute total $5 \times 5 = 25$ photos will uploaded with the same file names as a result the previous batch of 25 photos in the cloud storage will be overwritten.

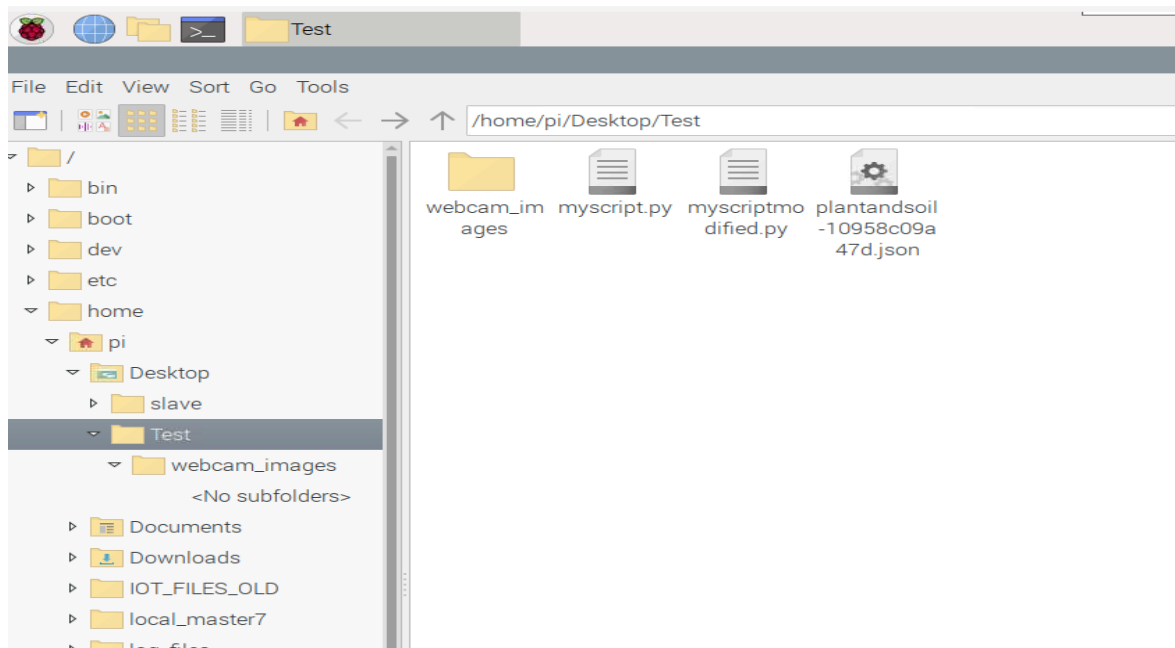


fig 10 Folder structure of each raspberry pis

plantrasp							
Location	Storage class	Public access	Protection				
us (multiple regions in United States)	Standard	Not public	None				
OBJECTS CONFIGURATION PERMISSION PROTECTION LIFECYCLE OBSERVABILITY NEW INVENTORY REPORTS NEW							
Buckets > plantrasp							
UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA MANAGE HOLDS DOWNLOAD DELETE							
Filter by name prefix only <input type="text"/> Filter Filter objects and folders							
<input type="checkbox"/>	Name	Size	Type	Created	Storage class	Last modified	Public access
<input type="checkbox"/>	rasp1_0.jpg	63.2 KB	image/jpeg	23 May 2023, 12:05:44	Standard	23 May 2023, 12:05:44	Not public
<input type="checkbox"/>	rasp1_1.jpg	62.8 KB	image/jpeg	23 May 2023, 12:05:57	Standard	23 May 2023, 12:05:57	Not public
<input type="checkbox"/>	rasp1_2.jpg	62.6 KB	image/jpeg	23 May 2023, 12:06:10	Standard	23 May 2023, 12:06:10	Not public
<input type="checkbox"/>	rasp1_3.jpg	61.8 KB	image/jpeg	23 May 2023, 12:06:23	Standard	23 May 2023, 12:06:23	Not public
<input type="checkbox"/>	rasp1_4.jpg	62.3 KB	image/jpeg	23 May 2023, 12:06:37	Standard	23 May 2023, 12:06:37	Not public
<input type="checkbox"/>	rasp2_0.jpg	76.8 KB	image/jpeg	23 May 2023, 12:05:44	Standard	23 May 2023, 12:05:44	Not public
<input type="checkbox"/>	rasp2_1.jpg	79.6 KB	image/jpeg	23 May 2023, 12:05:57	Standard	23 May 2023, 12:05:57	Not public
<input type="checkbox"/>	rasp2_2.jpg	79.9 KB	image/jpeg	23 May 2023, 12:06:11	Standard	23 May 2023, 12:06:11	Not public
<input type="checkbox"/>	rasp2_3.jpg	79.4 KB	image/jpeg	23 May 2023, 12:06:24	Standard	23 May 2023, 12:06:24	Not public
<input type="checkbox"/>	rasp2_4.jpg	79.4 KB	image/jpeg	23 May 2023, 12:06:37	Standard	23 May 2023, 12:06:37	Not public

fig 11 This is how the google cloud storage will look like after every batch get uploaded

In parallel, a Flask-based HTML interface was developed to display the output images from an 'output' folder. The interface was designed to refresh every 60 seconds, thereby updating the display with any new output images.

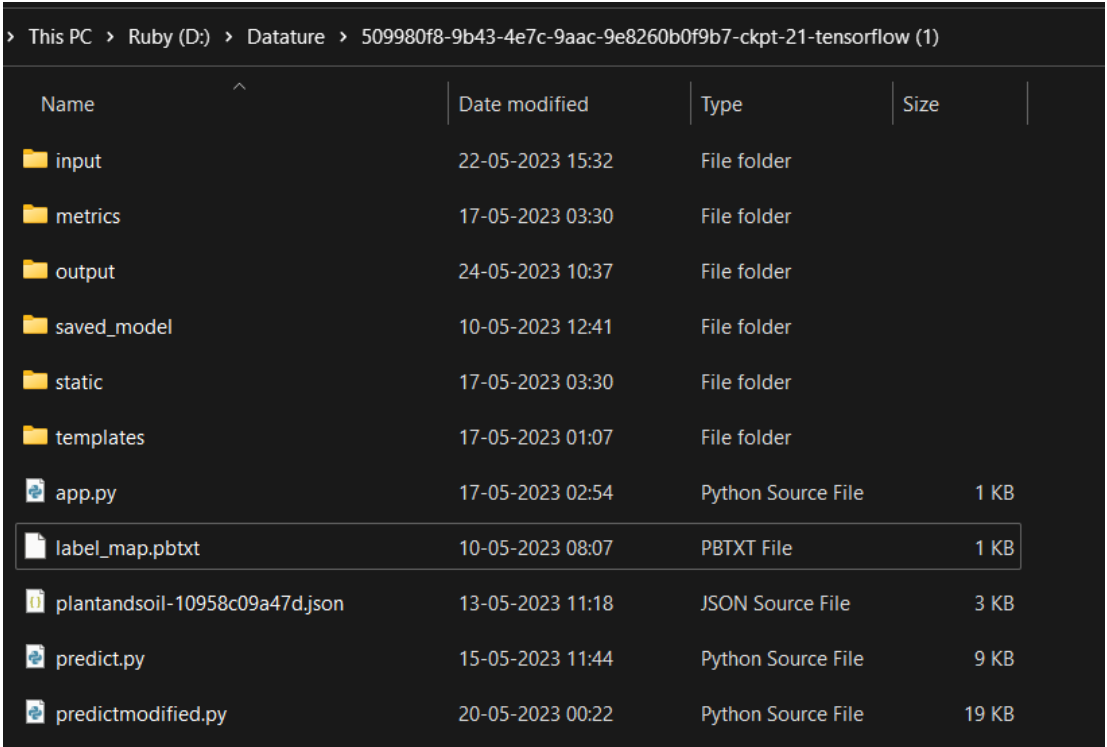
Phase 6: Integration and Operation

The final phase involved integrating all the developed components to establish a functional image capture and processing pipeline. The workflow was as follows:

1. Each Raspberry Pi unit captured field images in a window of 1 minute.
2. The images were named as "RASP{RASP NUMBER}_{IMAGE NUMBER}" and stored in a local folder called 'webcam_images' located within the local Raspberry Pi folder named 'Test'.
3. After that 1 minute window, the Raspberry Pi units entered a two-minute resting period during which they uploaded their batches of photos to Google Cloud.

4. A script running on a local PC ('predictmodified.py') downloaded the current batch of images from the cloud storage during this resting period. It then loaded and used the trained Mask R-CNN model to predict the outputs on these images, drawing bounding boxes around the identified regions, and stored the output in a designated folder.
5. A Flask-based web interface, which refreshed every 60 seconds, displayed these processed images.

The above workflow, once set in motion, provided real-time snapshots of the field with detected plant and soil regions, creating a continuous and automated field monitoring system.



Name	Date modified	Type	Size
input	22-05-2023 15:32	File folder	
metrics	17-05-2023 03:30	File folder	
output	24-05-2023 10:37	File folder	
saved_model	10-05-2023 12:41	File folder	
static	17-05-2023 03:30	File folder	
templates	17-05-2023 01:07	File folder	
app.py	17-05-2023 02:54	Python Source File	1 KB
label_map.pbtxt	10-05-2023 08:07	PBTEXT File	1 KB
plantandsoil-10958c09a47d.json	13-05-2023 11:18	JSON Source File	3 KB
predict.py	15-05-2023 11:44	Python Source File	9 KB
predictmodified.py	20-05-2023 00:22	Python Source File	19 KB

fig 12 Folder structure of the local computer

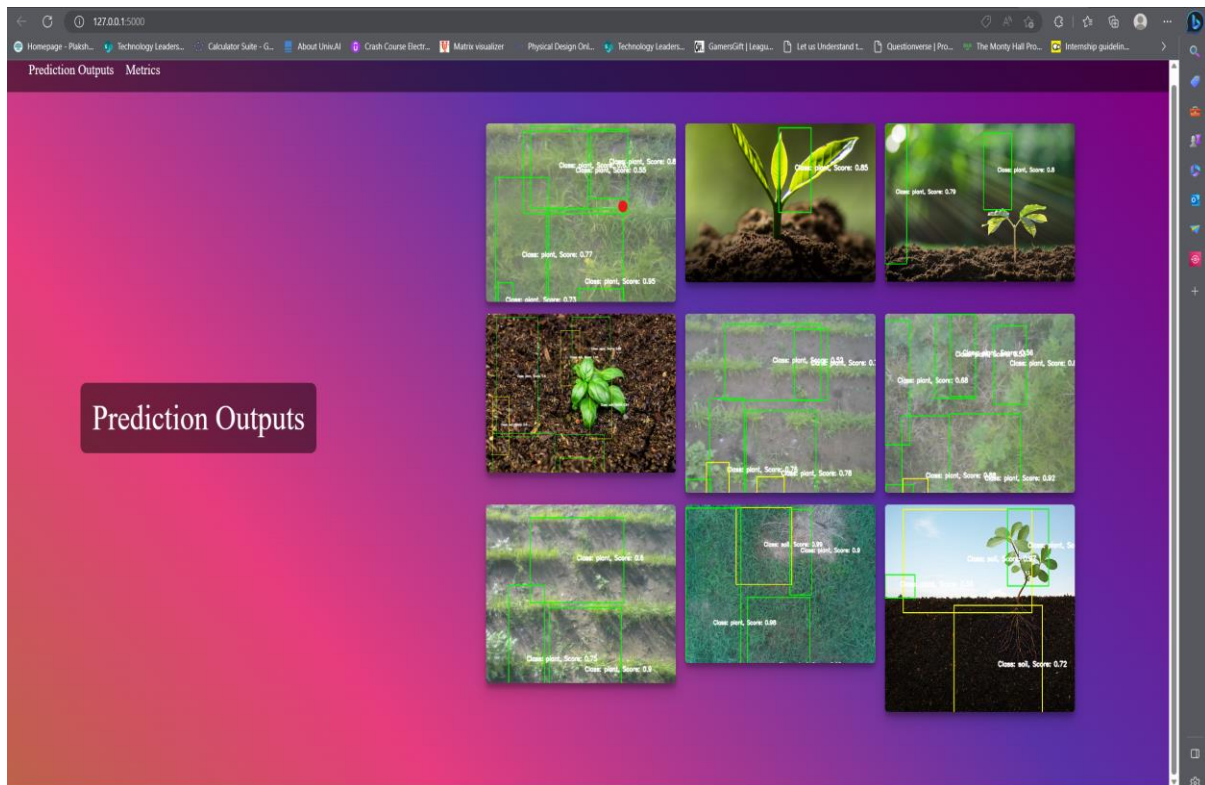


fig 13 The web interface showing the outputs

To reproduce the Results:-

- 1) Start the myscriptmodified.py file located in the local (/home/pi/Desktop/Test) of each R-pi via terminal.
- 2) Go to the google cloud storage account to monitor the uploaded file (optional)
- 3) Go to the local computer and navigate to the project folder
- 4) Now you have to run the predictmodified.py to do that open cmd/terminal while you are in that folder and put the following :

```
python3 predictmodified.py -i "D:\Datature\509980f8-9b43-4e7c-9aac-9e8260b0f9b7-ckpt-21-tensorflow (1)\input" -o "D:\Datature\509980f8-9b43-4e7c-9aac-9e8260b0f9b7-ckpt-21-tensorflow (1)\output" -m "D:\Datature\509980f8-9b43-4e7c-9aac-9e8260b0f9b7-ckpt-21-tensorflow (1)\saved_model" -l "D:\Datature\509980f8-9b43-4e7c-9aac-9e8260b0f9b7-ckpt-21-tensorflow (1)\label_map.pbtxt" -t 0.5 -b "plantrasp"
```

Replace the paths accordingly

- 5) After that navigate to the input folder to see the downloaded images uploaded to the google cloud from r-pis and also go to the output folder (optional)
- 6) Then open the app.py and run it , it will then generate the local web server which will show the outputs.

Providing the links to important files that I have uploaded in sharepoint

This is the main folder-

<https://plakshauniversity1.sharepoint.com/:f:/s/CenterforDigitalAgriculture/ErPnSXQWZ0pOjDCY3lVRe-UBngeDGmnCiyLLUby2gSYirA?e=oPu8nC>

The GCP Auto Vision model -

<https://plakshauniversity1.sharepoint.com/:v:/s/CenterforDigitalAgriculture/EToQIWn1Hd9BmHRt5H-OeZ0BMZeDOXVBIO6U6PFS9BG2VA?e=dGhg91>

The commented raspberry pi script link -

<https://plakshauniversity1.sharepoint.com/:u:/s/CenterforDigitalAgriculture/EWrudzOmk0pNuf7WJQsgocABHfhXjYnaLPHU9eOxRvIYtg?e=aLHEeL>

The saved model weights -

https://plakshauniversity1.sharepoint.com/:f:/s/CenterforDigitalAgriculture/EvZQjivJxlCr8549v_g4IcBSBjHKNbjpX173BWSGiRbxQ?e=YERwAn

The commented predictmodified.py file -

<https://plakshauniversity1.sharepoint.com/:u:/s/CenterforDigitalAgriculture/Edpv-34A5ONGhfkMPromKDYBAIs0pMc4X3eohGlt7ZMTTg?e=V8prum>

The flask app -

https://plakshauniversity1.sharepoint.com/:u:/s/CenterforDigitalAgriculture/ETo_7FPtrexAnRWHwN7LFM0B--oRVRhaXbwHMrkoAf6EHA?e=Ro7fzU

The static folder which has all the css -

<https://plakshauniversity1.sharepoint.com/:f:/s/CenterforDigitalAgriculture/EmzH8nHWB-utJkQRB-tmShCoBPweROSPoVvj1Q16x5Zm5rQ?e=miJDUW>

The template folder which has html files -

https://plakshauniversity1.sharepoint.com/:f:/s/CenterforDigitalAgriculture/Eu2s6LTG_cZHicD1e4smK-sB37sGYIsGxIK1DO1w-K8sSQ?e=otnlNu

Planned future work

1)Automated Irrigation:

As part of making our solution more comprehensive and useful for farmers, we plan to automate the irrigation process.

This will involve the use of soil moisture sensors, humidity sensors, and temperature sensors to provide real-time data on field conditions.

Based on this data, relay motors can be controlled to regulate the irrigation process, ensuring that the crops receive the optimum amount of water

2) Plant Disease Detection:

We plan to extend the capabilities of our system beyond just detecting plants and soil. Our next major goal is to incorporate the detection of plant diseases into our system.

This advancement will require the use of multispectral sensors to capture more detailed and comprehensive image data.

A significant requirement for this development will be access to a large dataset of diseased plant images and their spectral profiles to train our machine learning model accurately.

Conclusion

This project aimed at creating a real-time image capture and processing pipeline for field environments using Raspberry Pi cameras and a Mask R-CNN model. The successful execution of the project demonstrated the potential of integrating Internet of Things (IoT) devices with machine learning for real-time field environment monitoring. The pipeline provided updated snapshots of the field, which could be viewed and evaluated through a web interface.

The project successfully navigated through several stages of data handling, machine learning model training, and deployment. Nevertheless, the model exhibited overfitting, indicating the need for further improvements, such as data augmentation, early stopping, or employing a more diverse dataset for training. Future work will focus on enhancing the model performance while maintaining real-time image capture and processing capabilities.

This project marks an essential step in leveraging technology for real-time agricultural monitoring, offering potential for significant advancements in precision agriculture.

References-

(<https://stackoverflow.com/>, n.d.)

(<https://www.reddit.com/>, n.d.)

(<https://www.datature.io/>, n.d.)

(<https://www.analyticsvidhya.com/blog/>, n.d.)

(<https://www.sciencedirect.com/>, n.d.)

(<https://www.ieee.org/>, n.d.)

<https://plakshauniversity1.sharepoint.com/:f/s/CenterforDigitalAgriculture/EviyUtOzlyJMu5tQlRsQ4ZgBLocRBHFYR97ILNLEjA6suA?e=E5rtVT> (the link to the literature survey folder

along with the summary and DOI)