applied sciences

MDPI

Article

Development of a Prototype Solution for Reducing Soup Waste in an Institutional Canteen

Ana Correia [1], Clara Aidos [1], João M. L. P. Caldeira [1,2,*] and Vasco N. G. J. Soares [1,2,3]

[1] Polytechnic Institute of Castelo Branco, Av. Pedro Álvares Cabral, nº 12, 6000-084 Castelo Branco, Portugal; ana.correia2@ipcbcampus.pt (A.C.), clara.aidos@ipcbcampus.pt (C.A.), vasco.g.soares@ipcb.pt (V.N.G.J.S.)
[2] Instituto de Telecomunicações, Rua Marquês d'Ávila e Bolama, 6201-001 Covilhã, Portugal
[3] AMA—Agência para a Modernização Administrativa, Rua de Santa Marta, nº 55, 1150-294 Lisbon, Portugal
* Correspondence: jcaldeira@ipcb.pt

Abstract: Food waste has gained increasing attention and debate, given its economic, environmental, social, and nutritional implications. One-third of food intended for human consumption is wasted. Although it is present at all stages of the food supply chain, it is in the final stages of consumption, such as households and food services, that the problem becomes most evident. This work builds on a previous study by the same authors, which identified computer vision as a suitable technology for identifying and quantifying food waste in institutional canteens. Based on this result, this paper describes the proposal and implementation process of a prototype demonstration. It is based on a Raspberry Pi 4 platform, a Resnet-50 model adapted with the Faster Region-Convolutional Neural Network (Faster R-CNN) model, and an algorithm for feature extracting. A specially built dataset was used to meet the challenge of detecting soup bowls and classifying waste in their consumption. A web application was developed to visualize the data collected, supporting decision making for more efficient food waste management. The prototype was subjected to validation and functional tests, and proved to be a viable, low-cost solution.

Keywords: food waste; institutional canteens; computer vision; ResNet-50; Faster R-CNN; feature extraction; Internet of Things; prototype

Citation: Correia, A.; Aidos, C.; Caldeira, J.M.L.P.; Soares, V.N.G.J. Development of a Prototype Solution for Reducing Soup Waste in an Institutional Canteen. *Appl. Sci.* **2024**, *14*, 5729. https://doi.org/10.3390/app14135729

Academic Editor: Luca Fiori

Received: 7 June 2024
Revised: 26 June 2024
Accepted: 27 June 2024
Published: 30 June 2024

1. Introduction

Food waste is a global problem that is manifested by the significant food loss throughout the supply chain [1]. This phenomenon, especially visible in the retail and final consumption phases [2], is a consequence of the behavior adopted by retailers and consumers, which results in a substantial reduction in the amount of food consumed [3]. The magnitude of this problem is staggering, as around a third of food intended for human consumption, equivalent to approximately 1.3 billion tons per year, is lost or wasted worldwide [1]. The need to address this issue is clear, especially considering target 12.3 of the Sustainable Development Goals (SDGs), which aims to halve per capita food waste by 2030, both at the level of retailers and consumers and along production and supply chains [4].

Although food waste occurs at all stages of the food supply chain, in developed countries, it tends to be more evident in the final stages of consumption, such as households and food services. The work carried out in this paper focuses on institutional canteens, where food waste includes both prepared meals that have not been sold (i.e., leftovers) and food that remains on plates after the meal has been consumed (i.e., scraps).

The relevance of food waste in institutional canteens lies not only in the significant volume of food wasted, but also in the opportunities this context offers for implementing technological solutions aimed at reducing waste. Institutional canteens, due to their

standardized and controlled nature, are an ideal setting for the application of monitoring and data analysis technologies.

One of the main challenges in quantifying and reducing food waste is the lack of automated and accurate methods for monitoring waste continuously and in real time. This work follows on from the conclusions presented in a previous study [5] by the same authors of this paper. Within the scope of that work, the problem related to food waste was framed and a comprehensive analysis of the state of the art was carried out, with a focus on computer vision techniques. A critical review of the convolutional neural network (CNN) models and datasets commonly used in this area was carried out, and the performance of Inception-V3 and ResNet-50 was evaluated, along with the Food-101 dataset [5].

Giving continuity to that work, the aim of this paper is to propose, implement, test, and validate a prototype based on the Internet of Things (IoT) and computer vision techniques to quantify food waste in an institutional canteen, particularly soup. The choice of focusing food waste on soup is due to the difficulties in detecting waste in a second course meal, as people tend to scatter the leftovers. In addition, the weight of the plate may not be a good indicator, as fruit peelings may be added, for example, which could affect the results. The work carried out here for soup can also be applied to desserts.

This paper is organized as follows. Section 2 presents the prototype demonstrator developed as part of this work. It describes and justifies the architecture adopted, as well as the hardware and software components and the implementation process. Section 3 describes the experiments conducted to validate the prototype. Finally, Section 4 presents the conclusions of this paper, highlighting the results obtained and discussing possible directions for future work.

2. Prototype

This section presents the developed prototype as part of this work. It describes the architecture that was defined, the decisions made in its specification, and details the process of implementing the prototype's hardware and software.

2.1. Architecture

Figure 1 shows the architecture of the prototype solution developed, which makes it possible to quantify the food waste of soup in an institutional canteen, in the process of delivering the tray after the meal has been consumed.

The physical component consists of a camera and an infrared sensor connected to a Raspberry Pi [6], with the aim of capturing images of the trays. The captured images are not stored locally on this device, given its processing and storage limitations. Therefore, the image of the tray is transferred to a local Ubuntu server [7], which is running on a virtual machine.

On this server, a Python script was created with the purpose of observing the destination directory of the images and then running the detection and classification model. First, it tries to detect the presence of a bowl in the captured image, using the Resnet-50 and Faster R-CNN computer vision models. Then, from the image of that bowl, it performs feature extraction and calculates the volume of soup remaining (i.e., wasted). This server has a MySQL database [8], implemented in a Docker container [9], which will store the data collected in the previous stage, i.e., the volume wasted and a corresponding date.

To enable the integration and consultation of the information collected, as well as management and support for decision making, a web application based on microservices was developed, made up of individual containers that communicate with each other. All these services, from the database to the microservices, are implemented in Docker containers. The Frontend microservice is responsible for the user interface, allowing the stored data to be consulted. The Backend microservice interacts directly with the MySQL

database. The Eureka microservice acts as a microservice discovery service and provides communication between them.

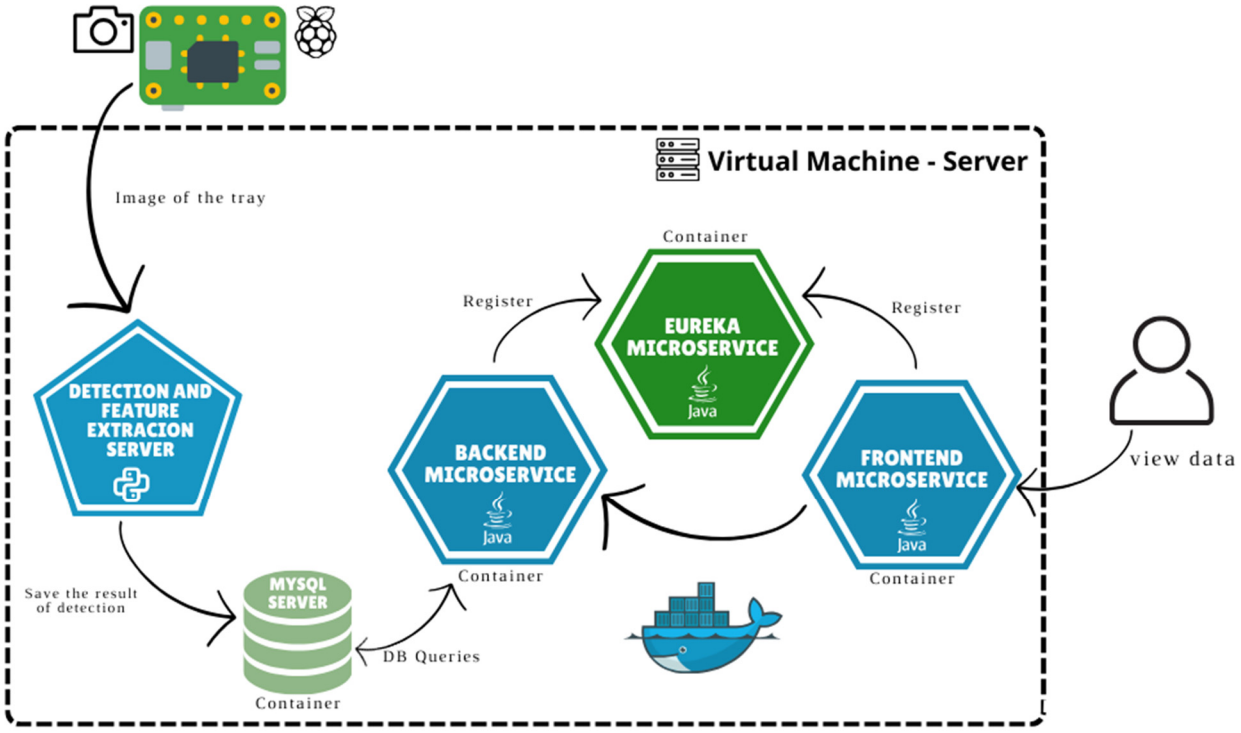


Figure 1. Prototype architecture.

Choosing an architecture based on microservices is a future-oriented decision, as it offers several important advantages. Microservices increase resilience, ensuring that component failures do not affect the entire system. They facilitate scalability, allowing each service to grow as needed. They improve testing efficiency by allowing each service to be checked in isolation. In addition, they enable the use of multiple technologies, increasing adaptability and continuous innovation [10].

2.2. Hardware Component

This subsection describes the characteristics of the hardware components used in the implemented solution.

2.2.1. Raspberry Pi and Peripherals

The Raspberry Pi 5 [6] is a highly versatile computing device designed to provide high performance in a variety of applications. It is equipped with a quad-core Broadcom BCM2712 Arm Cortex-A76 processor operating at 2.4 GHz. For easy integration into existing networks and systems, it includes dual-band 802.11 ac Wi-Fi, Bluetooth 5.0/Bluetooth Low Energy (BLE) and Gigabit Ethernet with PoE+ support. It is expandable via a PCIe 2.0 ×1 interface and enables the connection of fast peripherals, such as Node Version Manager (NVM) drives and solid-state drives (SSDs). With 4 GB or 8 GB LPDDR4X-4267 Synchronous dynamic random-access memory (SDRAM) options and a high-speed microSD card slot, this solution used a 32 GB Kingston microSD [11] with a speed of 100 MB/s.

The camera used was a Raspberry Pi Camera Module 3 [12]. This is a compact camera designed for the Raspberry Pi, equipped with a 12-megapixel Sony IMX708 sensor and an I2C-controlled focus actuator. This module features a large sensor with a diagonal of 7.4 mm and a diagonal viewing angle of 75 degrees. In addition, it supports a High Dynamic

Range (HDR) mode to ensure superior image quality and features ultra-fast autofocus, along with a library of software commands for precise control.

A Waveshare ST188 infrared reflection sensor [13] was used to detect the tray. It consists of an infrared reflector–transmitter with an LM393 voltage comparator. It has an adjustable sensitivity and a signal output indicator. Its specifications include a power output of 3.0 V to 5.3 V, dimensions of 25 mm by 15.9 mm, and mounting holes of 2.0 mm.

Figure 2 illustrates the integration of the Raspberry Pi 5 and its peripherals.

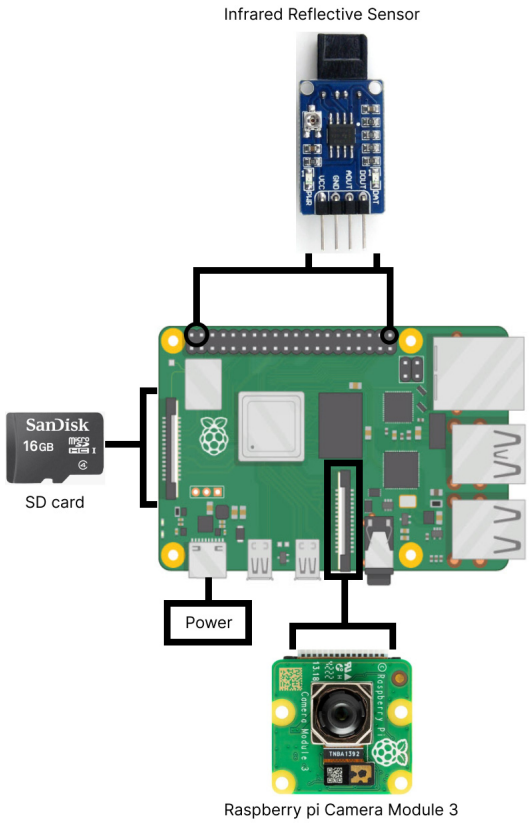


Figure 2. Integration of hardware components.

The steps taken to integrate the hardware components are described next. First, you need to insert the SD card into the appropriate slot on the device. Next, the camera is connected to the Raspberry Pi by carefully inserting the flexible cable into the designated connector, ensuring that the contacts are facing the HDMI port, as illustrated in [14] and [15]. The infrared sensor is then connected, using the Digital Output (DO) to detect the presence or absence of reflection and adjusting the sensitivity as necessary. The sensor is connected to the Raspberry Pi's GPIO4, with the sensor's 5 V supply pin connected to the Raspberry Pi's 5 V pin, and the sensor's Ground pin to the Raspberry Pi's Ground pin, which in Figure 3 correspond to the numbers 7 (GPIO4), 4 (5 V), and 6 (Ground), respectively [16]. In addition, the sensor's potentiometer is adjusted to calibrate the sensitivity to infrared reflection. These guidelines were obtained from the technical support forums in [17–19]. Finally, the power cable is connected to supply power to the system.

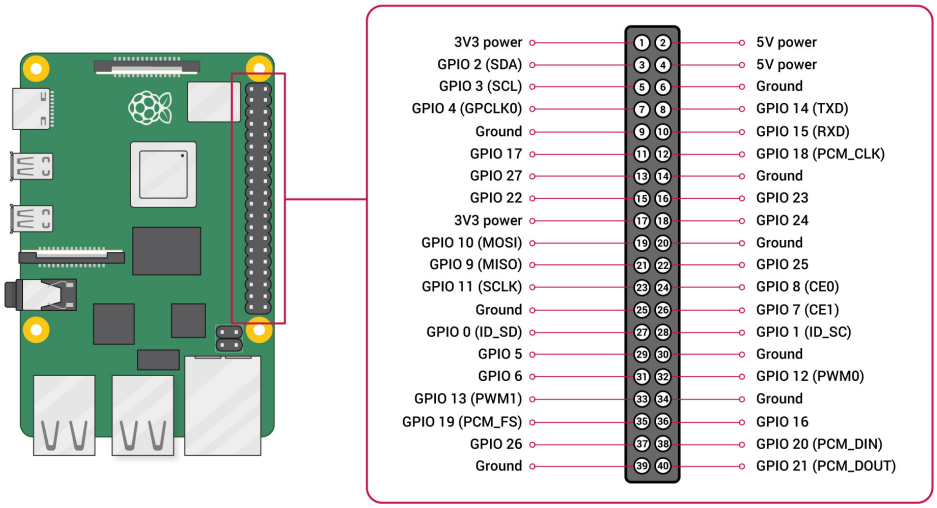


Figure 3. Raspberry Pi GPIO pin diagram. Source: [16].

Figure 4 shows the integration of the Raspberry Pi Camera Module 3 and the infrared sensor with the Raspberry Pi 5.

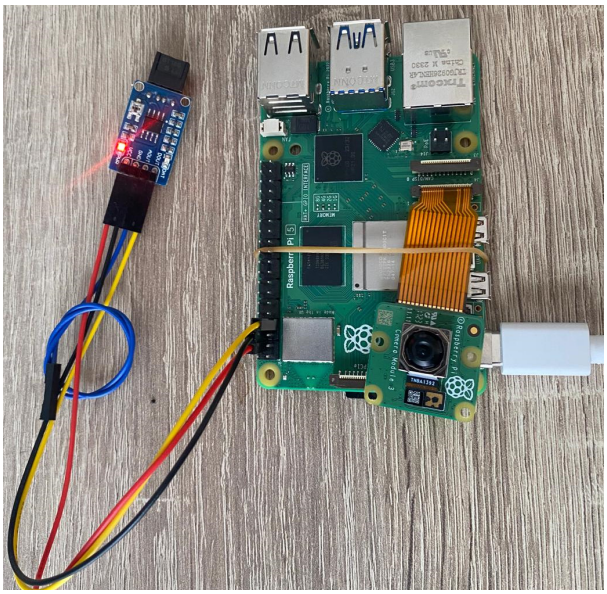


Figure 4. Sensor integration on the Raspberry Pi 5.

Figure 5 shows the test bench and the physical component of the prototype, simulating the project's application scenario. This scenario is an institutional canteen, where meals are served on standardized trays, typically consisting of a cup, plate, and soup bowl. At the end of the meal, the trays with the leftover food are delivered to a collection window connected to the kitchen, and at this point, the images are captured by the prototype.

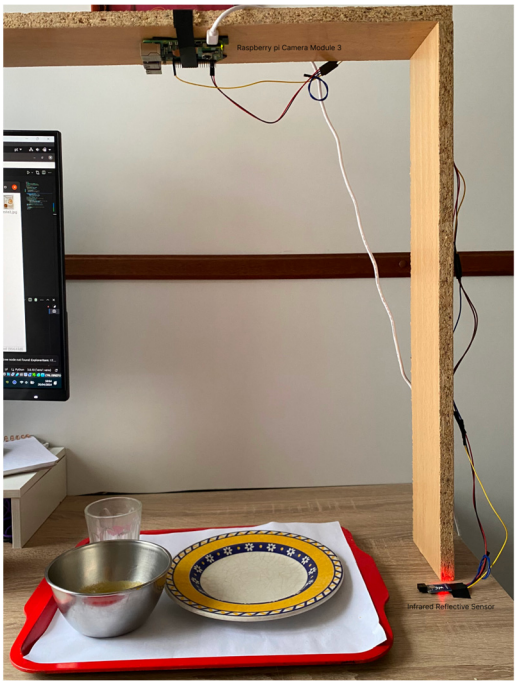


Figure 5. Prototype test bench.

2.2.2. Server

As described above, the Raspberry Pi communicates with a local server. In this implementation, a virtual machine created with Oracle VM VirtualBox [20] was used to act as the server. The virtual machine runs the Ubuntu 20.04 operating system. The hardware components configured for this virtual machine include 9037 MB of RAM, 4 CPUs, and 55 GB of storage on a virtual SATA hard disk, located in the *server-project.vdi* file. These hardware components are virtual, configurable as required, and allow the server to run efficiently.

2.3. Software Component

This subsection details the various software elements developed and configured to implement the prototype, starting with creating the dataset, training and validating the models, configuring the IoT devices, local server functionalities, and developing a web application to visualize the results.

2.3.1. Dataset

To create the dataset used to train the model, images were taken of the trays in the canteen at the Superior School of Technology of the Polytechnic Institute of Castelo Branco, Portugal [21]. The dataset consists of 270 images of trays. It should be noted that the images are very similar to each other, with only the position of the bowl on the tray and the amount of soup in the bowl varying. However, the aim is to detect only this specific type of bowl, since this dataset was built to work in institutional canteens that have bowls of this type, such as the canteen at the Superior School of Technology. This is to avoid, for example, dessert bowls wrongly identified, as the purpose of this project is to detect soup bowls to calculate their waste. Figure 6 shows three examples of collected images that are included in the dataset created.

Figure 6. Examples of images that compose the dataset.

The Roboflow platform [22] was used to build the dataset. Initially, a project was created by selecting the "Classification" type. The collected images were then uploaded. The "Manual Labeling" option was used to manually annotate the images. In the annotation process, the "bowl" class was created. After this stage, "Start Annotate" was selected and, using a "Bounding Box", an outline was drawn around the bowl of soup, as illustrated in Figure 7. This procedure was repeated for all the images captured.

Figure 7. Image annotation in Roboflow.

Once all the images had been annotated, the dataset was exported using the "Export Dataset" option. Before exporting, data augmentation techniques such as "flip", "rotation", "blur", and "noise" were added, as well as pre-processing including "resize" and "auto-orient" to increase the number and variety of images.

Figure 8 shows the composition of the dataset. The Roboflow platform manages the composition and division of the dataset. The images have been uploaded and annotated, and the platform decides how many should be used for training, testing. and validation. In this case, 252 images are included for training, 11 images for validation and 7 images for testing. Once exported, this dataset will be used to train the CNN model, as described below. This dataset is available for download at [23].

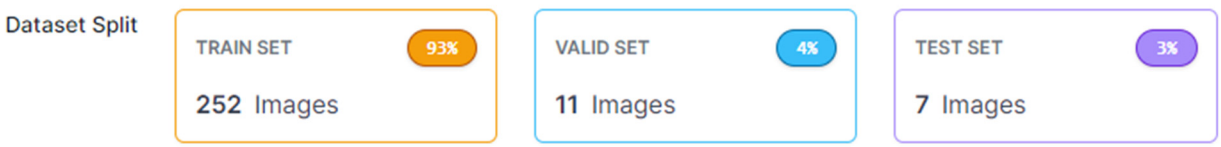


Figure 8. Dataset split.

2.3.2. Resnet-50 and Faster R-CNN Models

CNNs, or convolutional neural networks, are specifically designed to process grid-structured data such as images and have excelled in computer vision tasks such as image classification, detection, and segmentation. Models based on CNNs are made up of convolutional, pooling, and fully connected layers, which extract and represent features from the input image [24]. In the architecture of a CNN, shown in Figure 9, the convolutional layer is responsible for extracting features from the image using trainable filters, which identify patterns such as edges, textures, and shapes. The resulting feature maps are passed through the ReLU layer, which introduces non-linearity, allowing the network to learn complex tasks [24–26].

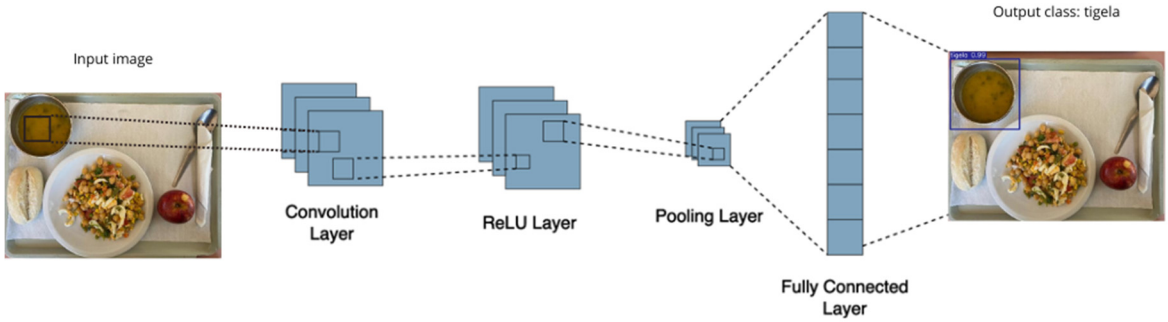


Figure 9. Typical architecture of a CNN.

Following on from the work in [5] by the same authors of this paper and the conclusions observed, it was decided to combine two models: Resnet-50 and Faster R-CNN. These two models combine two different approaches in the field of computer vision and deep learning for object detection. Resnet-50 is a deep neural network architecture known as a "residual network", designed to overcome training challenges in deep networks [27]. Faster R-CNN describes an approach that divides the image into regions and uses a CNN to identify objects in those regions. By uniting these two approaches, as illustrated in Figure 10, it is possible to take advantage of Resnet-50's ability to extract complex features from images, while Faster R-CNN handles object detection.

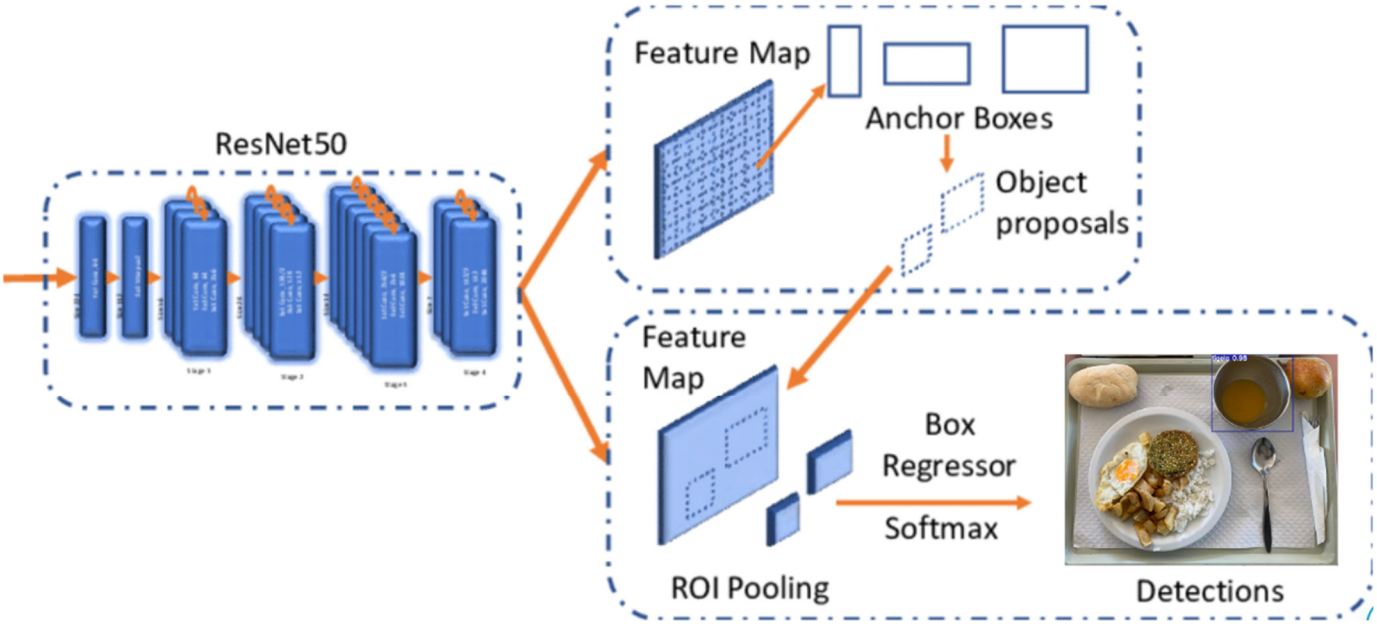


Figure 10. Architecture of the Resnet-50 and Faster R-CNN models.

The model was trained on the Google Colab platform [28], using a notebook available at [29]. The notebook, with Python code, was adapted to allow the dataset prepared for this purpose to be imported, using the Roboflow API [22]. The Google Colab platform offers free hardware resources, such as Graphics Processing Units (GPUs) and Tensory Processing Units (TPUs), support for various languages, and integration with Google Drive [30] and GitHub [31]. The machine provided by the platform in the free plan has the following characteristics: NVIDIA T4 graphics card with 16 GB of VRAM and 13 GB of RAM for the system.

In addition to training the model, this notebook also allows validation of the model, which consists of analyzing the validation images of the dataset to assess performance. Figure 11 shows an example image of the validation results carried out directly on the notebook, showing that the model accurately identifies the location of the bowl.

Figure 11. Validation results with images from the dataset.

The model was trained several times with different versions of the dataset, and the number of training epochs was adapted to avoid overfitting. Overfitting represents a significant challenge when training CNNs. It occurs when a model is over-trained to the point where it memorizes specific details of the training data. As a result, the model performs excellently on the training data, but fails when dealing with new data [32].

The model used was trained for 19 epochs. The decision to stop training was made based on several factors. When it is found that the model's performance on the dataset is not improving, i.e., it is stagnating or getting worse, indicating an increase in errors, training is stopped [33]. In this scenario, where all the bowls are the same and therefore the images of the trays are very similar, extending training for too many epochs would not improve the model's performance and could even make it worse.

The performance metrics used to evaluate the model were mean average precision (*mAP*) and loss. Average precision (*AP*) is a metric commonly used in the context of binary classification and information retrieval to summarize the precision–retrieval curve. It provides a single value that represents the quality of retrieval results classified to a specific class or category, especially in tasks such as object detection [34]. The calculation is carried out according to Equation (1).

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] \times Precisions(k) \quad (1)$$

The *mAP* is obtained by averaging the accuracies at different recall levels (evaluation of the model's effectiveness in detection) [35]. This metric is particularly important because it is not influenced by the size of the objects, which allows for an effective

comparison between models. It is calculated based on Equation (2), where the average of the *AP* of the classes considered is obtained.

$$mAP = \frac{1}{k}\sum_{i}^{k} AP_i \quad (2)$$

Finally, the loss during training reflects how the model is adjusting to the dataset and is expected to decrease as the epochs progress [36]. However, a very low loss value does not guarantee good performance on new data, as it may indicate an excessive fit to the training data. The formula for calculating the loss is shown in Equation (3).

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] \times Precisions(k) \quad (3)$$

Figure 12 shows that, in the first few epochs, the *mAP* is significantly high. This is because we are working in a very controlled environment, where all the images in the dataset come from a single canteen, resulting in great similarity between them. Consequently, the results are extremely positive, and from epoch 5 onwards, the *mAP* is 1.0.

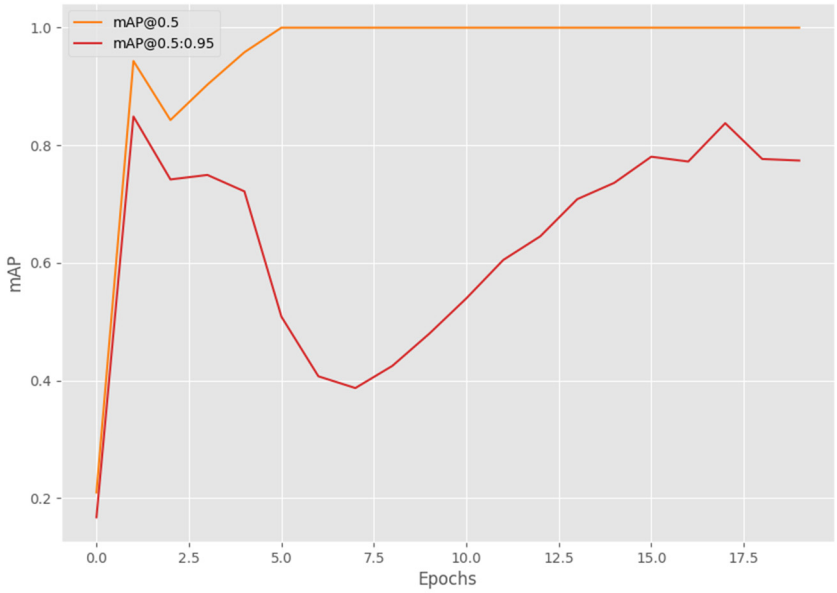


Figure 12. mAP results in Resnet-50 + Faster R-CNN training.

Regarding loss, it can be seen in Figure 13 that its initial value is not excessively high. Over the epochs, it gradually decreases, stabilizing at a value of 0.0359.

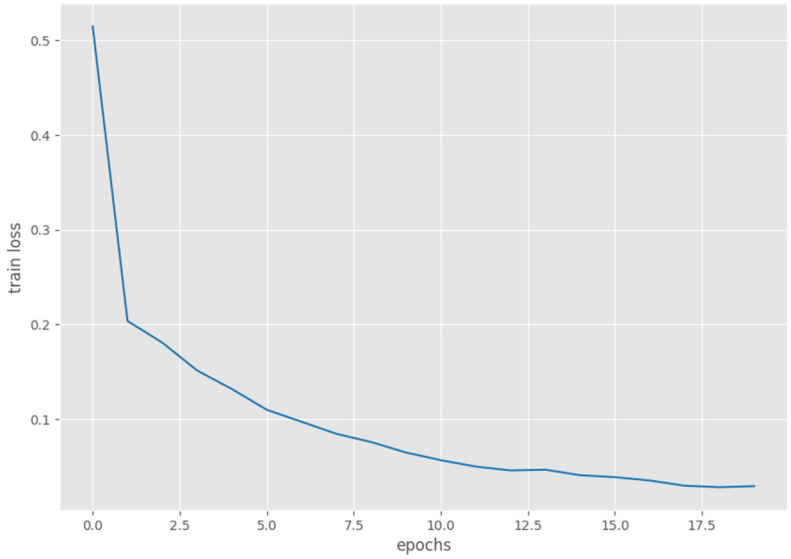


Figure 13. Loss results in Resnet-50 + Faster R-CNN training.

2.3.3. Software Configurations on IoT Devices

The first task carried out on the Raspberry Pi was to install the operating system, in this case the Raspberry Pi OS available at [37]. As described above, its peripherals were installed, i.e., the Raspberry Pi Camera Module 3 [12] and the ST188 infrared sensor [13].

Next, a Python script was developed to enable communication between the Raspberry and the server and the integration of the infrared sensor to trigger the action of capturing the image as soon as a tray is detected. Figure 14 shows the flowchart illustrating the process carried out by this script to capture and transfer the images. The process begins with the initialization of the script, where the camera and infrared sensor are configured for the operation.

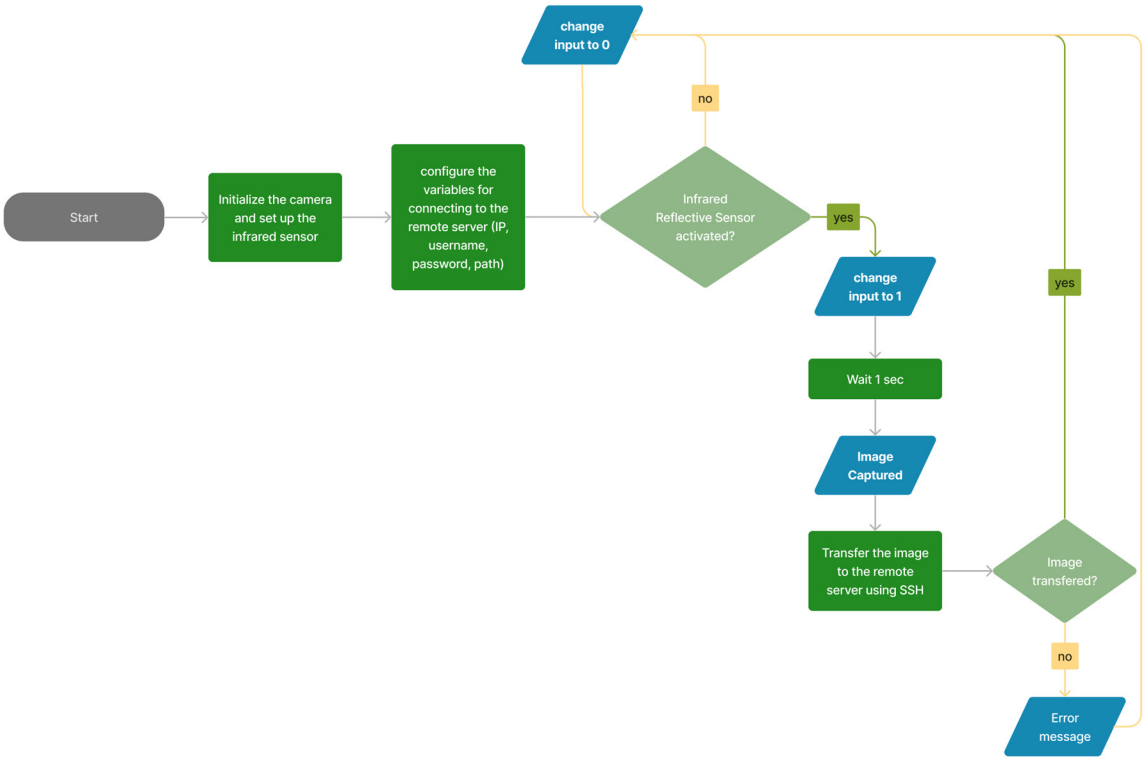


Figure 14. Flowchart of the Python script for capturing and transferring images on the Raspberry Pi.

The Python Picam2 library [38] was used to configure the camera. The size of the images was set to 1920 px by 1080 px. When the script is executed, a preview of the camera starts, allowing real-time visualization of what is being captured. As for the infrared sensor settings, the Python GPIOZero library [39] was used to manipulate the digital pins.

Next, the variables needed to connect to the remote server were defined, including the IP address, the server's username and password, and the path of the destination directory on the server, where the captured images will be stored.

Initially, the script checks whether the infrared sensor is active. If the sensor is not active, the input is 0 and the script continues to check until the sensor is active. When the sensor is active, the input is changed to 1 and the script waits 1 s before capturing the image, to prevent it from being blurred. So that repeated images of the same board are not captured, a new image is only captured if the previous input value is different from 1. In other words, if a tray remains in front of the sensor for a long time, only one image will be captured. When the input is changed to 0, indicating that the previous tray has been removed, and the sensor is activated again due to the presence of a new tray, a new image will be captured.

Once the image has been captured, it is transferred to the local server using the Python Paramiko library [40,41]. In the test scenario, the server and the Raspberry Pi are on the same local network. So, communication between the two is performed via SSH and using private IP addresses.

Afterwards, the script checks whether the image transfer was successful. The image is never saved locally on the Raspberry to avoid storage problems. If the image is successfully transferred, the input is reset to 0. If the transfer is not successful, an error message is displayed and the input is also reset to 0, returning to the initial step of checking whether the sensor is active.

This process is repeated continuously, allowing images to be captured and transferred whenever the infrared sensor detects the presence of a tray.

2.3.4. Ubuntu Local Server

The local server is hosted on a virtual machine and performs various functions. Initially, it was created to receive the images from the Raspberry and run the detection and classification models as well as feature extraction. However, for prototyping purposes, it also has the function of hosting both the database and the web application, using Docker [9].

To automate the process, a Python v3.12.3 script was created involving all the functions needed to carry out the detection and classification, extract the features, calculate the volume, and insert the data into the database. Figure 15 shows a flowchart illustrating this process.

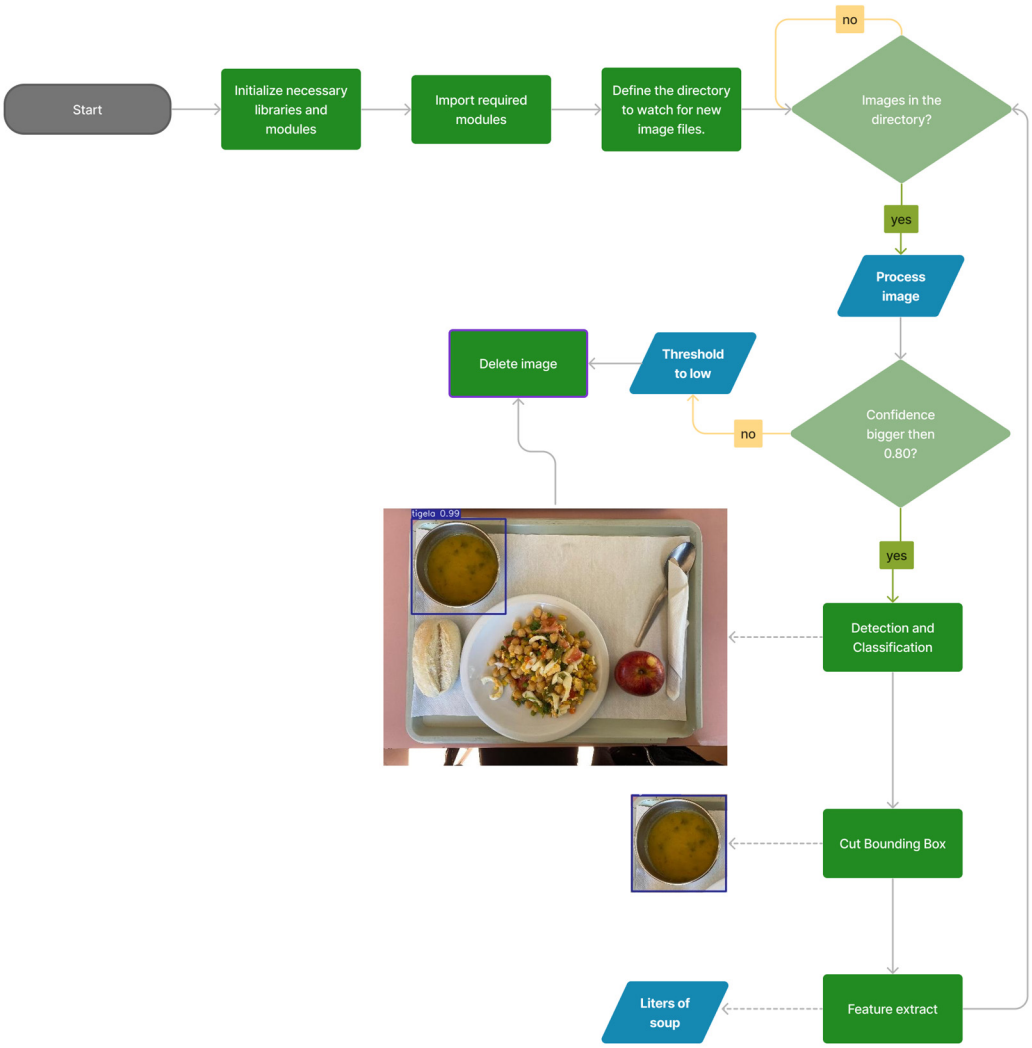


Figure 15. Flowchart of the main Python script for processing the images on the server.

The directory where the images coming from the Raspberry are stored is under watchdog, which means that when an image arrives, it is immediately processed. This functionality has been implemented using the Python library Watchdog Observer [42].

The first stage of the processing involves detecting and classifying bowls in the images. To do this, we adapted a project available online from the same author as the notebook used to train the model [29]. This detection script, explained in the flowchart in Figure 16, uses Python libraries such as Pytorch [43], Numpy [44], Cv2 [45], OS [46], YAML [47], and Matplotlib [48].

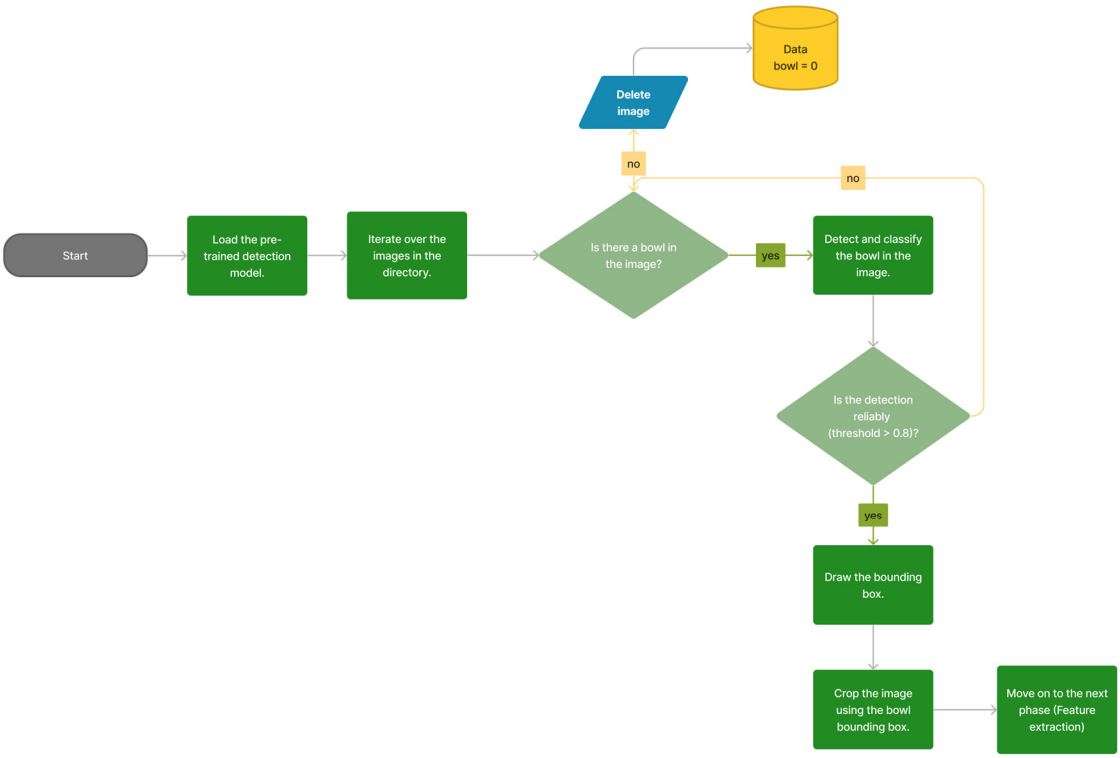


Figure 16. Flowchart of the Python script for detecting and classifying bowls in images.

The "load pre-trained detection model" step iterates over the images in the directory, detecting and classifying bowls in each one. If the bowl is detected with a threshold higher than 0.8, the image is cropped by the bounding box and the next step is carried out: "feature extraction". If no bowl is detected, the image is deleted. The value set for the threshold must be considered acceptable, meaning that the model will only truly consider a bowl if it has a confidence of over 80%. In other words, if the model detects a bowl, but with a confidence of 10%, it will not be considered, and the bounding boxes will not be drawn.

The next stage is "feature extraction". This is where the volume of wasted soup is calculated, i.e., the volume that remains in the bowl when the image of the tray is captured. The Python libraries Cv2, Numpy, Copy [49], and Math [50] were used to implement this functionality.

The feature extraction process, illustrated in Figure 17, begins with loading the image and converting it to grayscale. A Gaussian blur [51] is then applied to reduce noise and facilitate edge detection. The "HoughCircles function" is used to detect circles and is performed using the Hough algorithm [52], which identifies the circles in the image. Once the circles representing the bowl and the soup have been detected, masks are applied to filter out the orange areas of the image, which represent the soup. The largest circle detected corresponds to the radius of the bowl, while the smallest circle represents the soup in the bowl.

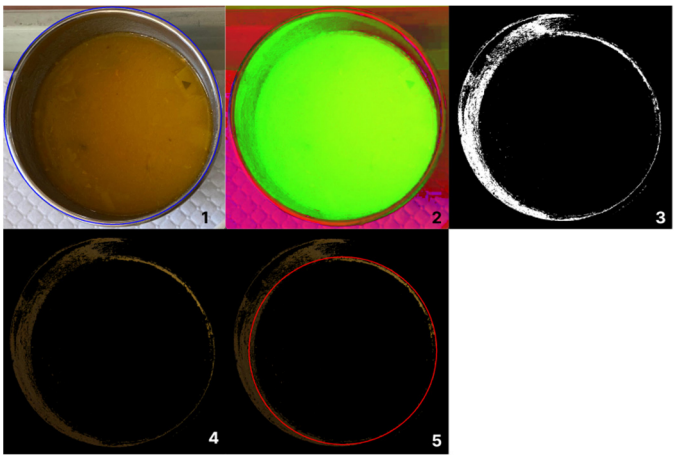


Figure 17. Image processing process for detecting circles.

The "calculate soup volume function" used to calculate the volume of the soup uses the radius of the bowl and the soup calculated earlier. First, it converts the pixel values to centimeters and adjusts the smaller radius of the soup according to the scale of the bowl. Next, the volume is calculated using the formula for the volume of a truncated cone [53], as shown in Equation (4). The function returns the total number of liters (i.e., milliliters) of soup in the bowl.

$$V = \frac{1}{3} * \pi * h * (r^2 + r * R + R^2), \quad (4)$$

where:

- R is the radius of the base of the original cone (bottom surface);
- r is the radius of the top surface;
- h is the height of our truncated cone.

Although this method provides an estimate of the soup's volume, it is important to recognize its limitations and consider possible error factors, such as inaccuracies in detecting circles and converting pixels to centimeters. Since the bowl does not effectively represent a truncated cone, due to the walls of the bowl not being straight, a margin of error must also be considered when calculating the volume of the soup. However, this process offers an automated and sufficient approach to calculating the volume of soup in the bowl. The flowchart in Figure 18 represents the feature extraction process described.

It should be noted that, to improve the structure and clarity of the code, each phase was implemented in a separate Python script and then called in a main script in the order described above. This main script is executed in a continuous cycle along with the observation of the directory. When an image arrives in the folder, the processes within the cycle are executed.

Finally, there is one last phase, which consists of the communication between these scripts and the MySQL database, to store the data from the volume calculation. Therefore, another script was created to connect to the database and insert the data into it. This script is called after the volume has been calculated. It should be noted that, for statistical purposes, data were also stored for trays that did not contain bowls, to compare the number of consumers who do or do not eat soup. If there is no bowl on the tray, the volume of wasted soup entered is zero.

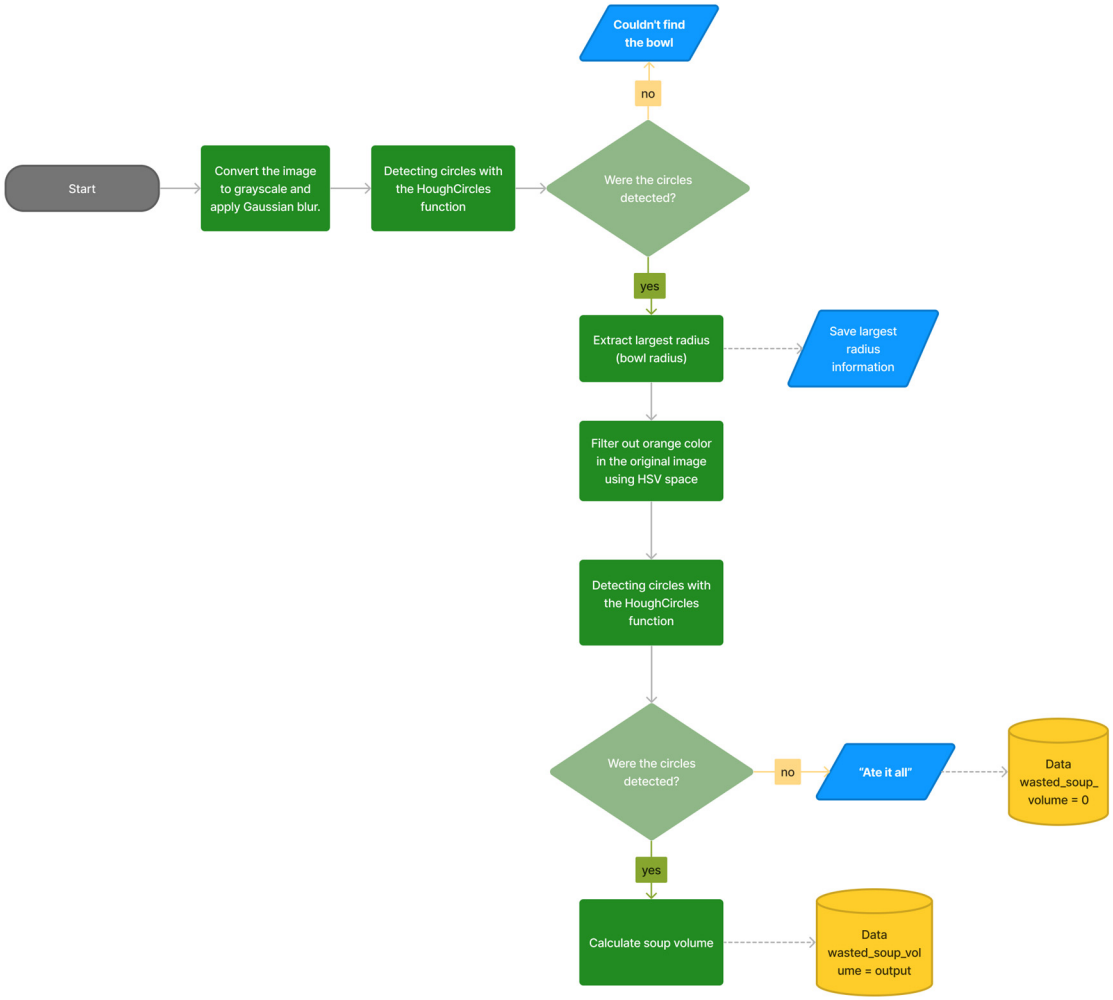


Figure 18. Flowchart of the Python script for feature extraction.

2.3.5. Database and Web Application

The database, as mentioned above, was implemented in a Docker container using a MySQL image (version 8.4.0) [54] available on Docker Hub [55]. The database consists of two tables: meal and menu. The meal table stores the data relating to each image processed and contains the fields shown in Figure 19. The menu table stores information entered by software users via the web application interface, such as the daily menus, including the type of soup and the quantity produced per day. This option for the end user to enter data is optional. The fields in the menu table can be seen in Figure 19.

It should be noted that some of the data in the database is fictitious, for demonstration purposes. In other words, since the prototype has not been put into production, the data entered the database come from the tests that have been carried out on the prototype, but other data have also been entered manually to populate the database. A Python script was developed which inserts data into the two tables with randomly generated values appropriate to the scenario.

As mentioned above, the web application developed to visualize the data collected is based on microservices. The application was developed using the Java programming language [56] and is based on the Spring Boot web framework [57]. It consists of three microservices: Frontend, Backend, and Eureka Server.

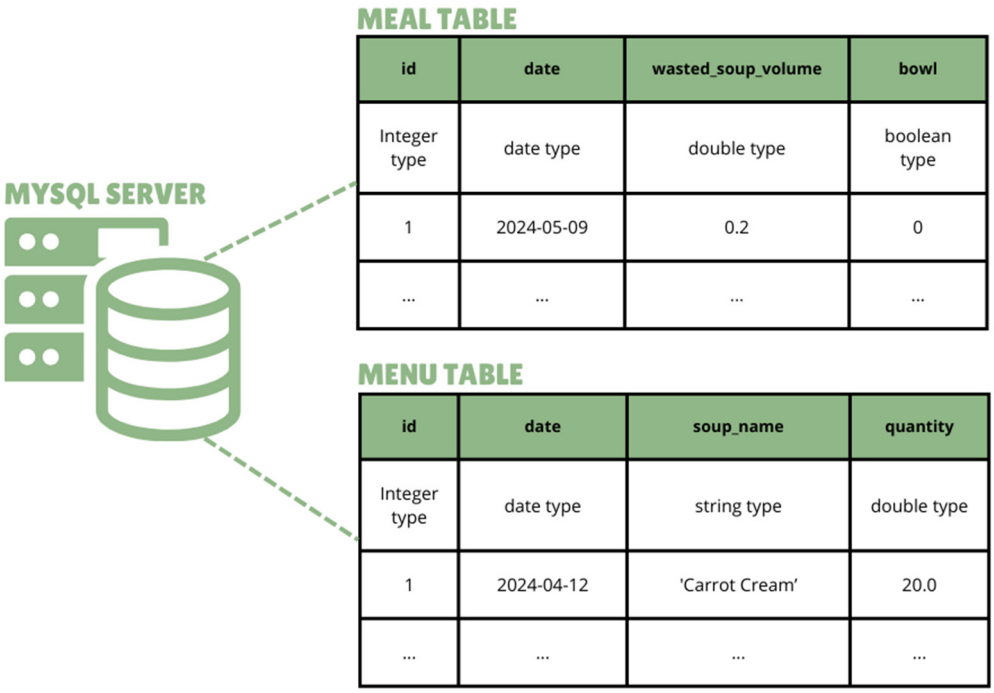


Figure 19. Composition of the MySQL database and its tables.

The Frontend microservice is responsible for presenting and managing the user interface (UI) and making requests to the Backend microservice. This microservice in turn is responsible for dealing with the database. The Eureka server is a microservice with various functionalities, such as load balancer, for example. However, for this project, its main function is the discovery of microservices [58]. In other words, it allows the various microservices to register, facilitating communication between them. Figure 20 shows the architecture of the application, as explained above.

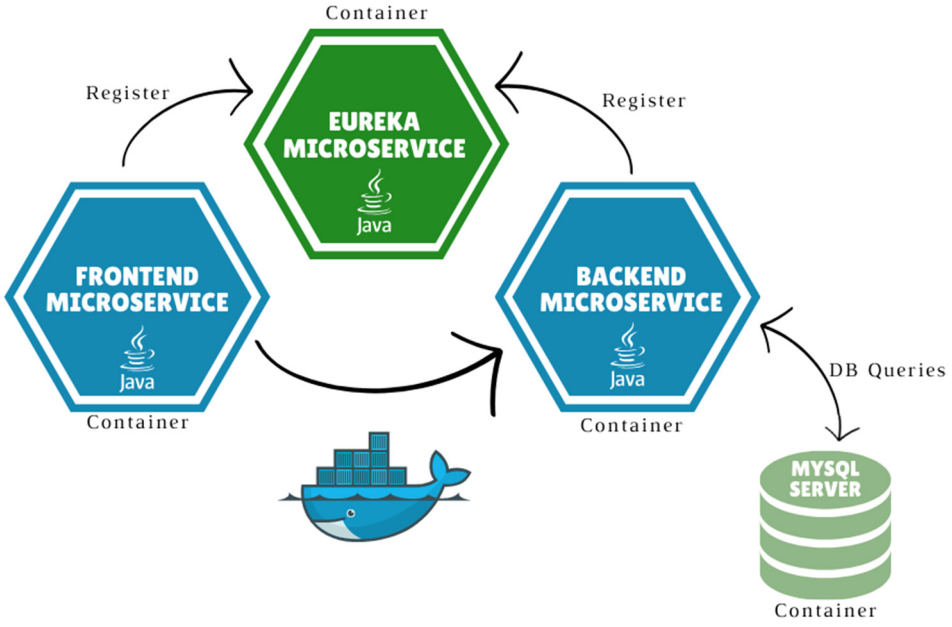


Figure 20. Web application architecture.

The Docker images of these microservices were created once they had been developed and pushed to the Docker Hub, so that the entire application could be functionally accessible, i.e., by accessing the server's IP and the port corresponding to the application.

The application's functionalities are described in the use case diagram shown in Figure 21. This diagram illustrates the interactions between the actors and the system, highlighting the various actions that can be carried out in the application.

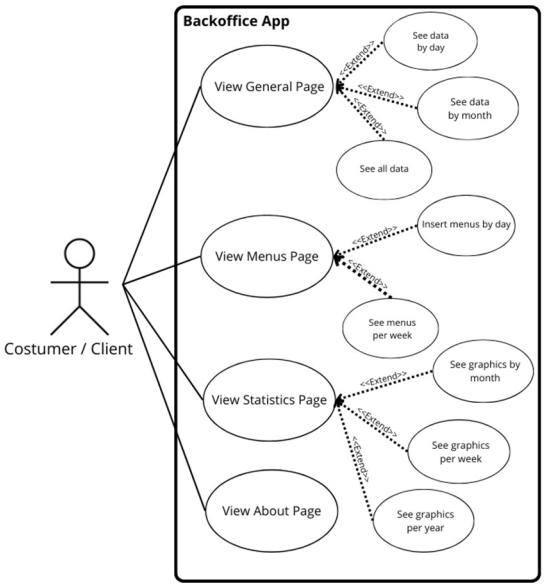


Figure 21. Use case diagram.

In this case, the main functionalities are viewing the home page; viewing the menus page; viewing the statistics page; and viewing the page containing information about the application. Within these main functionalities, there are other underlying ones. On the home page, data can be viewed by date, month, or all available data. On the menus page, you can enter daily menus and view them on a weekly basis. On the statistics page, data are presented graphically by month, week, or year.

Below is a set of screenshots of the web application, to give a detailed overview of its functionalities. Figure 22 shows the "General Page", where the user can select the data to view by specific date or month and view all the available data.

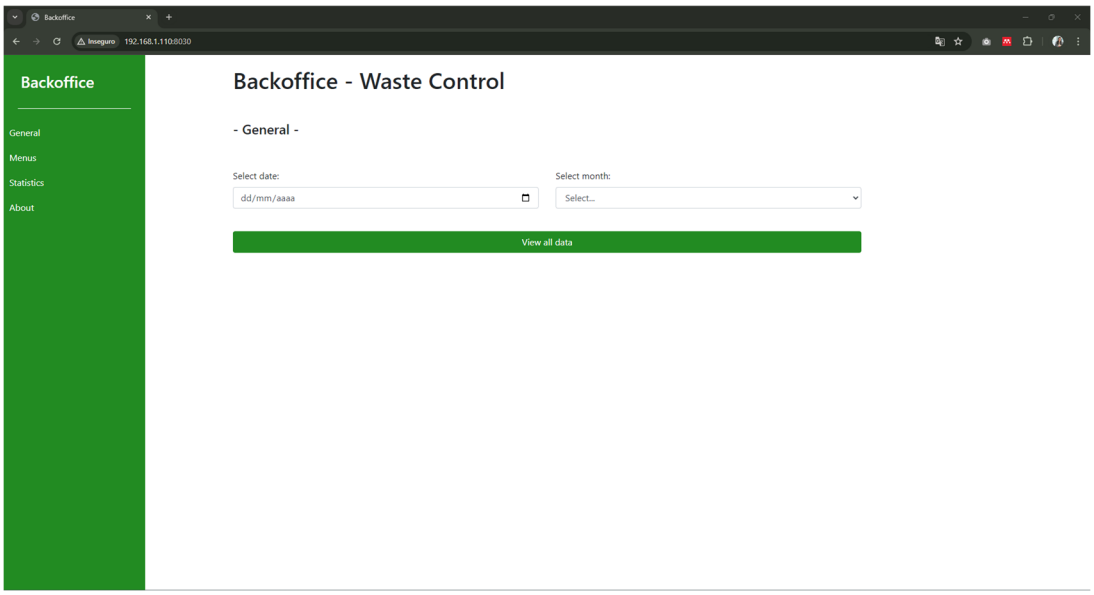


Figure 22. App Backoffice: general page.

On this page, the data are presented in tables, as can be seen in Figures 23 and 24. When viewing by date, it is possible to obtain information such as the volume of soup wasted that day and the volume produced (if entered by the user). You can also see the total number of meals served and compare those served with and without soup.

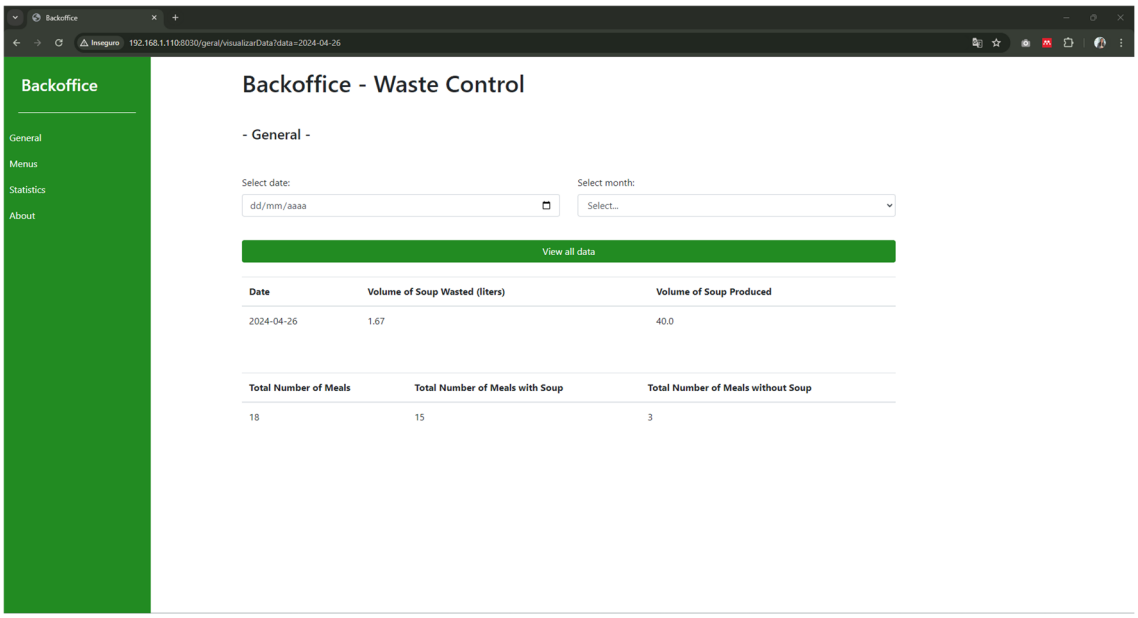


Figure 23. App Backoffice: general page: date.

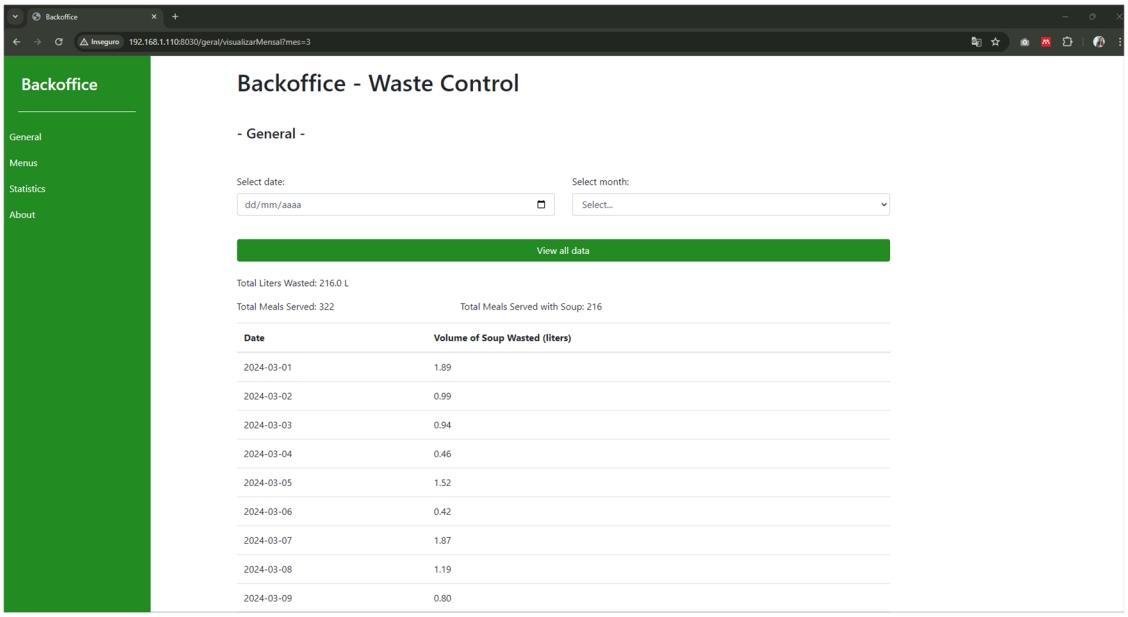


Figure 24. App Backoffice: general page: month.

If the user chooses the monthly query, they can view the volume of soup wasted over the various days of the selected month, as well as the total number of meals served that month. In addition, it is possible to view the total amount of soup wasted (in liters) and produced during that period, as shown in Figure 24.

On the "Menus" page available in the left-hand navigation bar, it is possible not only to enter the daily menu, as shown in Figure 25, but also to view the weekly menus, as shown in Figure 26.

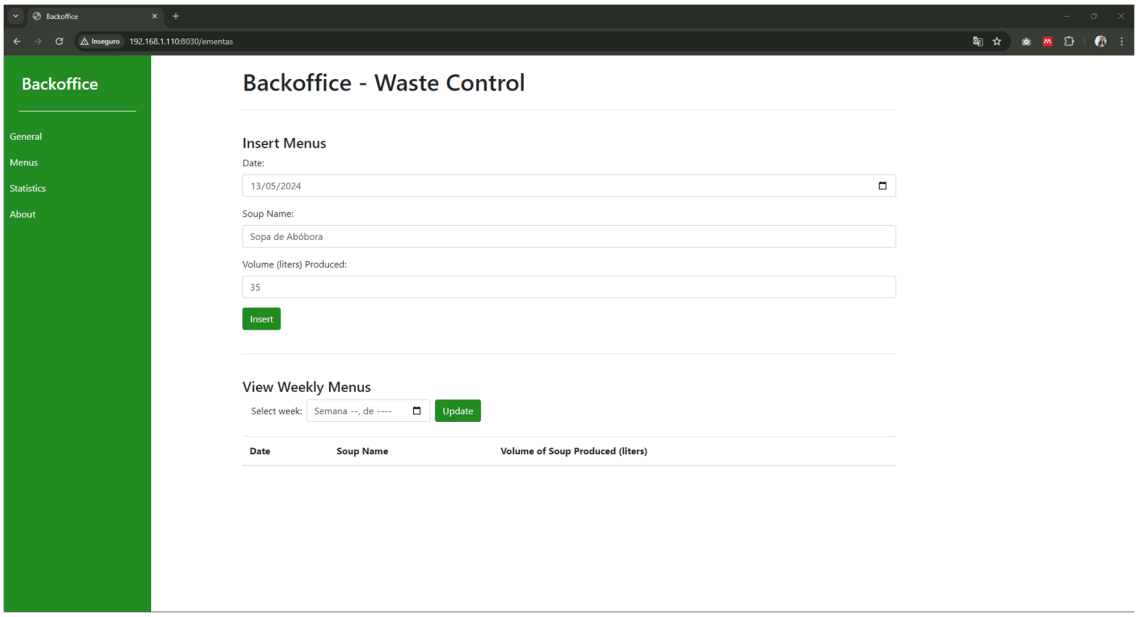


Figure 25. App Backoffice: menu page: insert menu.

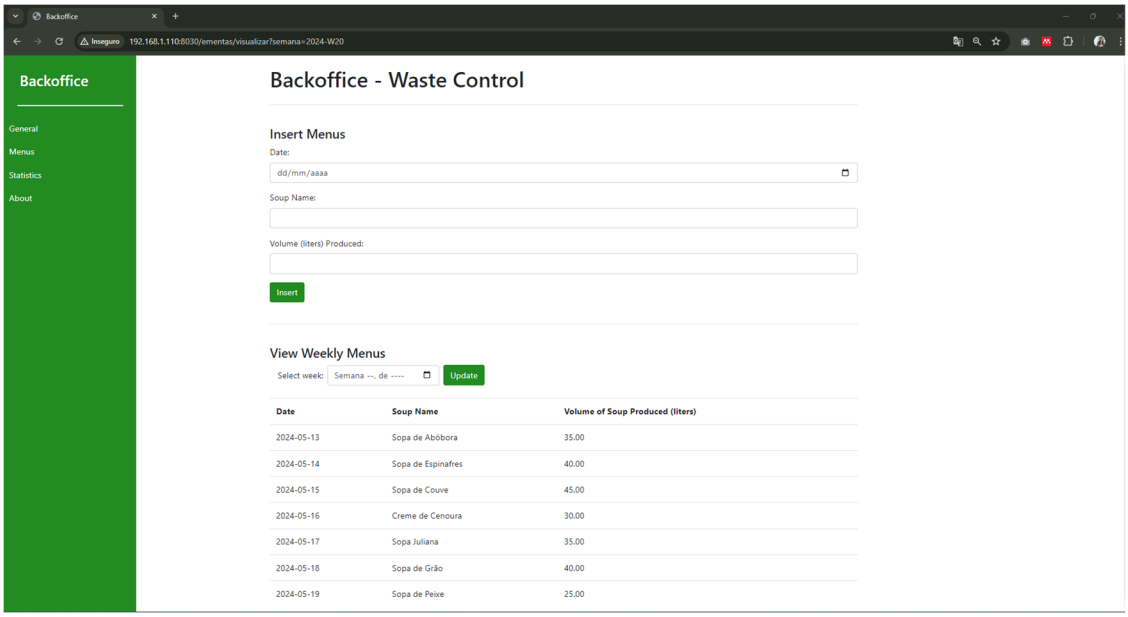


Figure 26. App Backoffice: Menus Page: view menu.

The "Statistics" page, as its name suggests, allows you to consult statistical data, based on the information stored in the database. As on the home page, you can choose how you want to view the data. In the "Week" option, shown in Figure 27, you can see the data collected on the days of the week selected by the user, relating to the number of trays that include soup or not, as well as the daily volume of soup wasted.

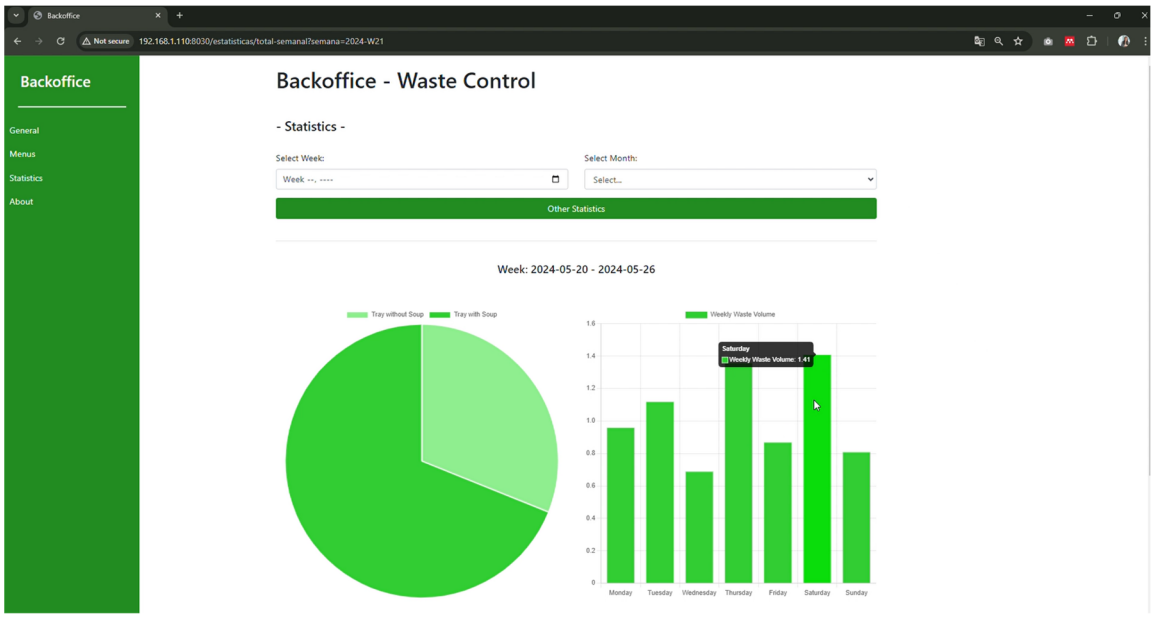


Figure 27. App Backoffice: statistics page: week. .

If the user chooses to view the monthly statistical data, shown in Figure 28, the circular graph is like the one shown in the weekly view, but with data for the month. The bar graph corresponds to the volume of soup wasted per week over the selected month, while the line graph allows you to analyze the volume of soup wasted per day during that month.

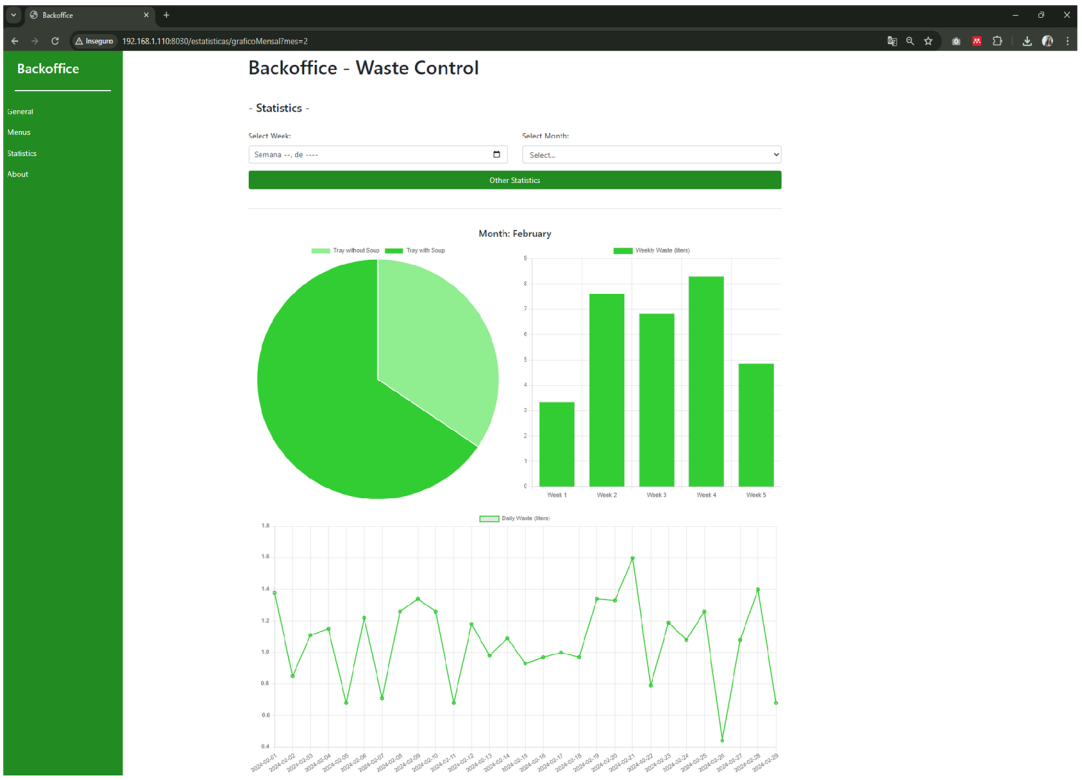


Figure 28. App Backoffice: statistics page: month.

Also, on the "Statistics" page, data can be viewed by year. Figure 29 shows the two types of graphs displayed when the user chooses this option. In these graphs, it is possible

to analyze the volume of soup wasted per month in the year 2024. It is also possible to compare the total number of meals served with and without soup per month.

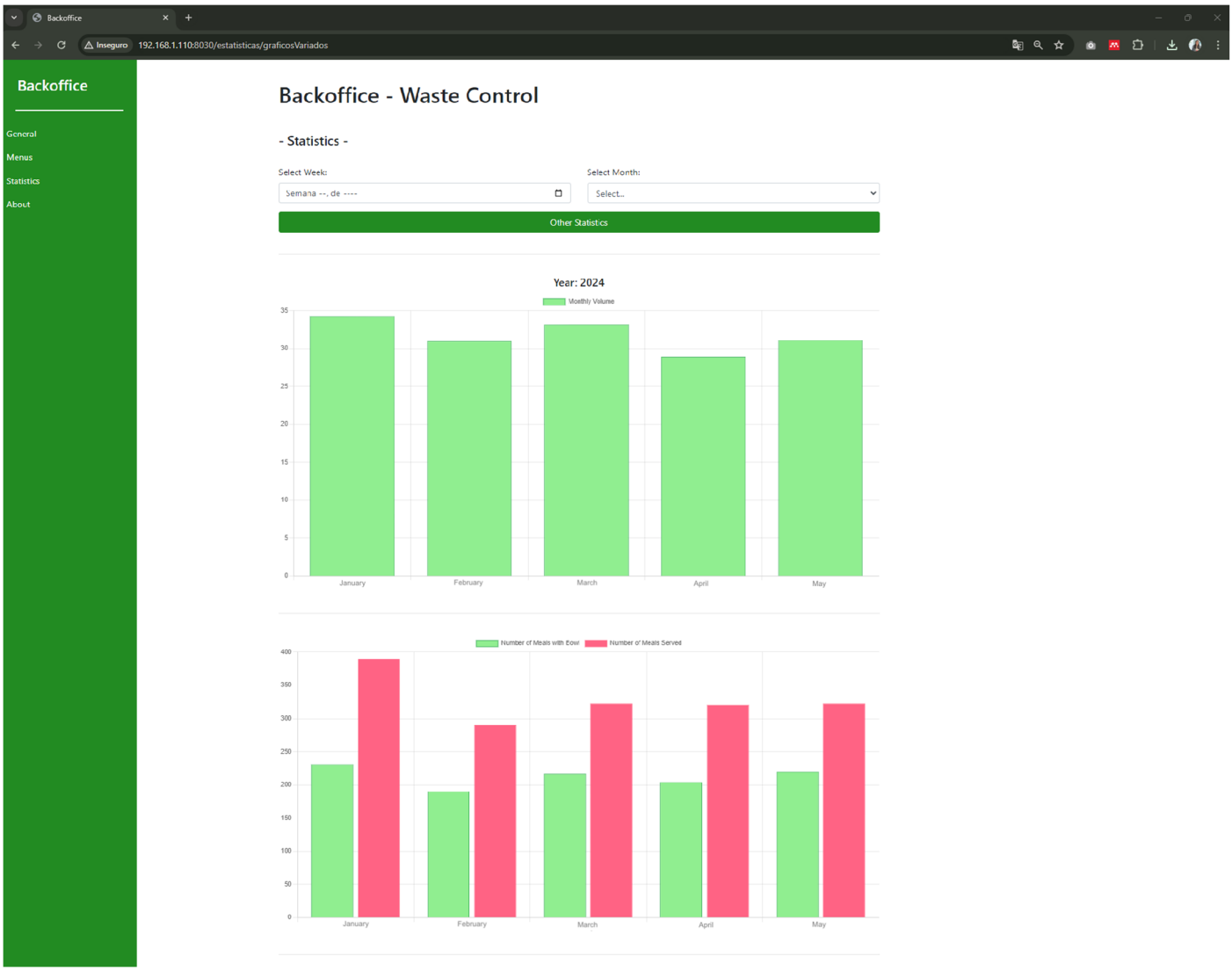


Figure 29. App Backoffice: statistics page: year.

Finally, Figure 30 shows the "About" page, which contains a brief description of the web application and the project developed.

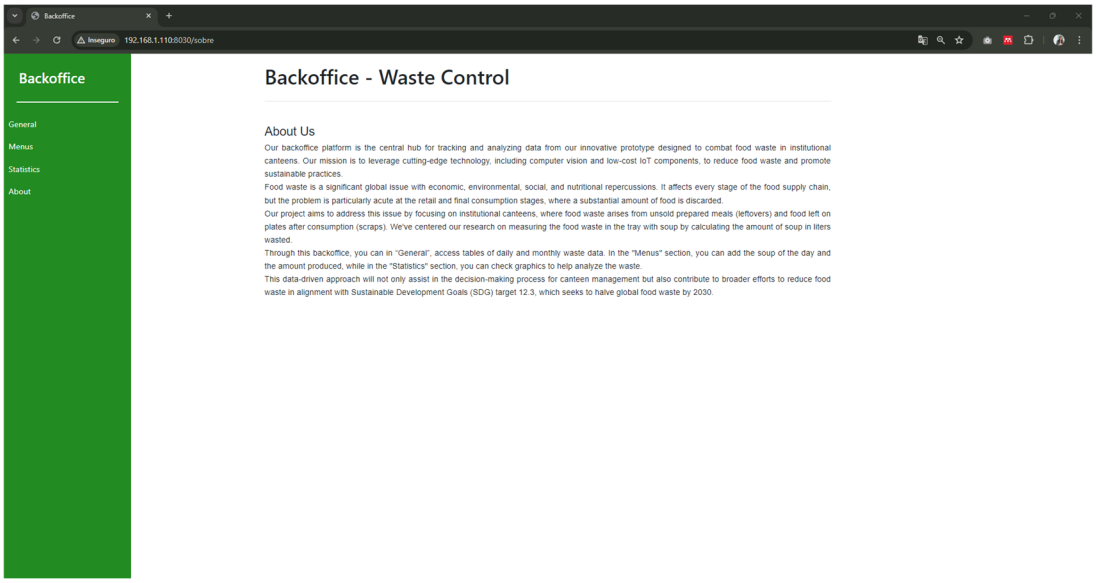


Figure 30. App Backoffice: about page.

3. Validation Tests

This section presents the validation tests carried out on the prototype. Figure 31 shows the operation of the infrared sensor connected to the Raspberry Pi, which consists of two red lights. When only one is on, this indicates that the tray has not been detected, and the input is 1. As soon as the tray is detected, both lights are on, and the sensor's input becomes 0.

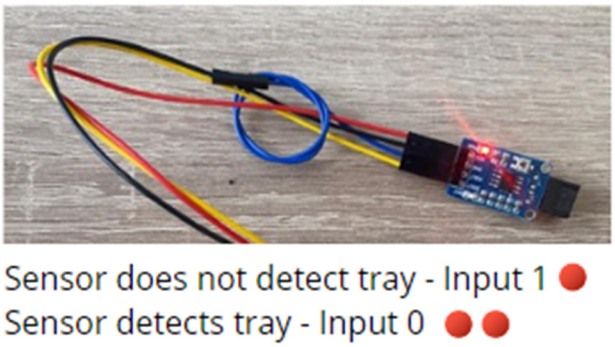
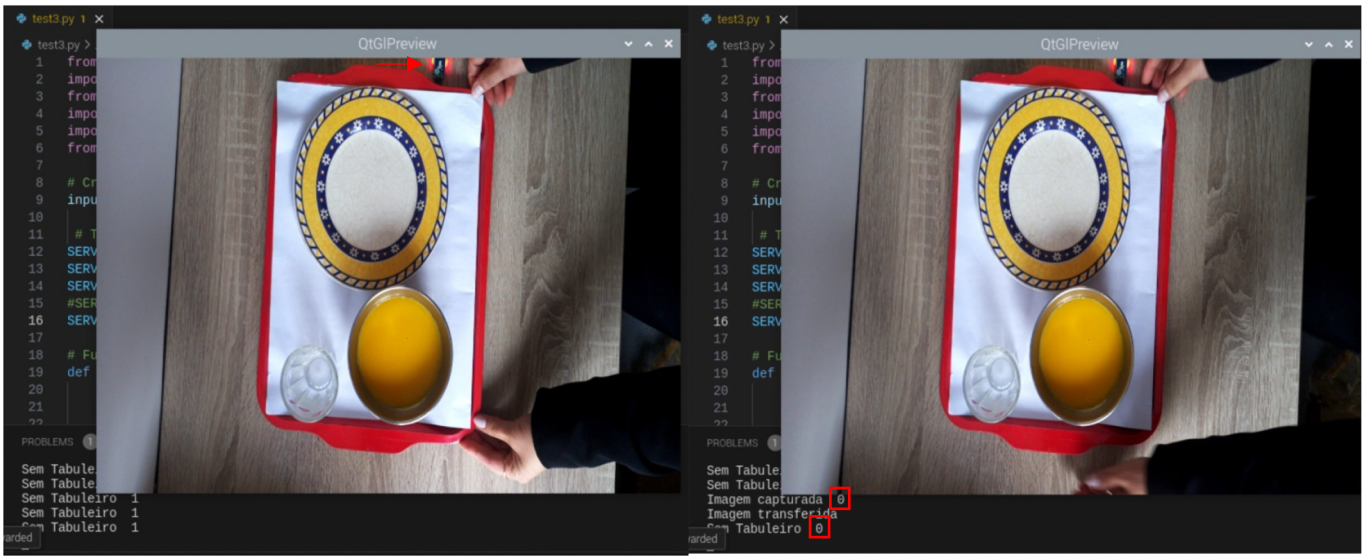


Figure 31. Infrared sensor operation.

Figure 32 shows the moment when the sensor identifies the tray, evidenced by both red lights coming on. To avoid capturing a blurry image, this is not done immediately, as shown in Figure 32a. Thus, the image is captured after 1 s, then transferred to the server, and the sensor input is changed to 0, indicating that the tray is still present. This process is illustrated in Figure 32b, which shows the successful capture of the image and the change of the input to 0.



(**a**) (**b**)

Figure 32. Image capture and transfer software running on the Raspberry Pi. (**a**) When the tray has not yet been detected; (**b**) after the tray has been detected and the input changed to 0.

As far as the server is concerned, Figure 33 shows that the captured image has been successfully stored in the "images" directory after capture and transfer. The connection to the database is then established. Then, the directory under observation detects the new image and starts processing it for classification and object recognition tasks.

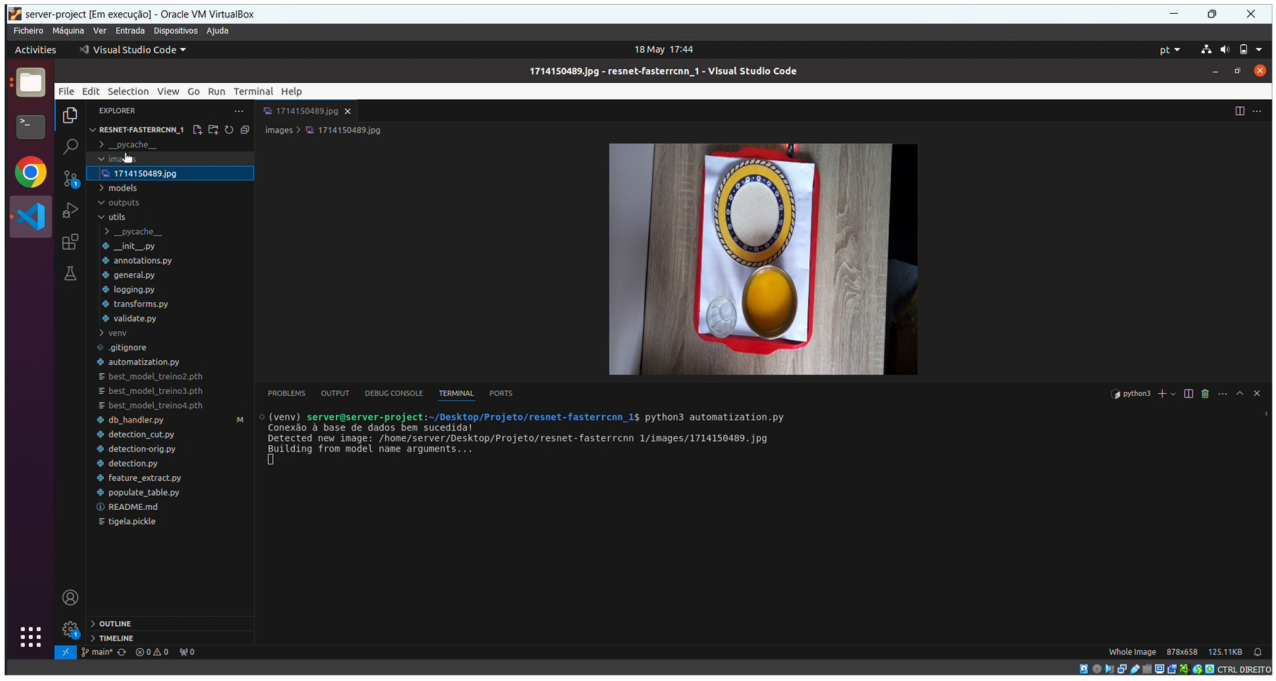


Figure 33. Image processing begins on the server.

Once detection is complete, the original image is deleted from the "images" directory. The image with the bounding boxes drawn is saved in the "outputs" directory, as shown in Figure 34. For feature extraction, the bowl is isolated and cut out by the boundaries of the bounding boxes. Then, its volume is calculated, as shown in the same figure.

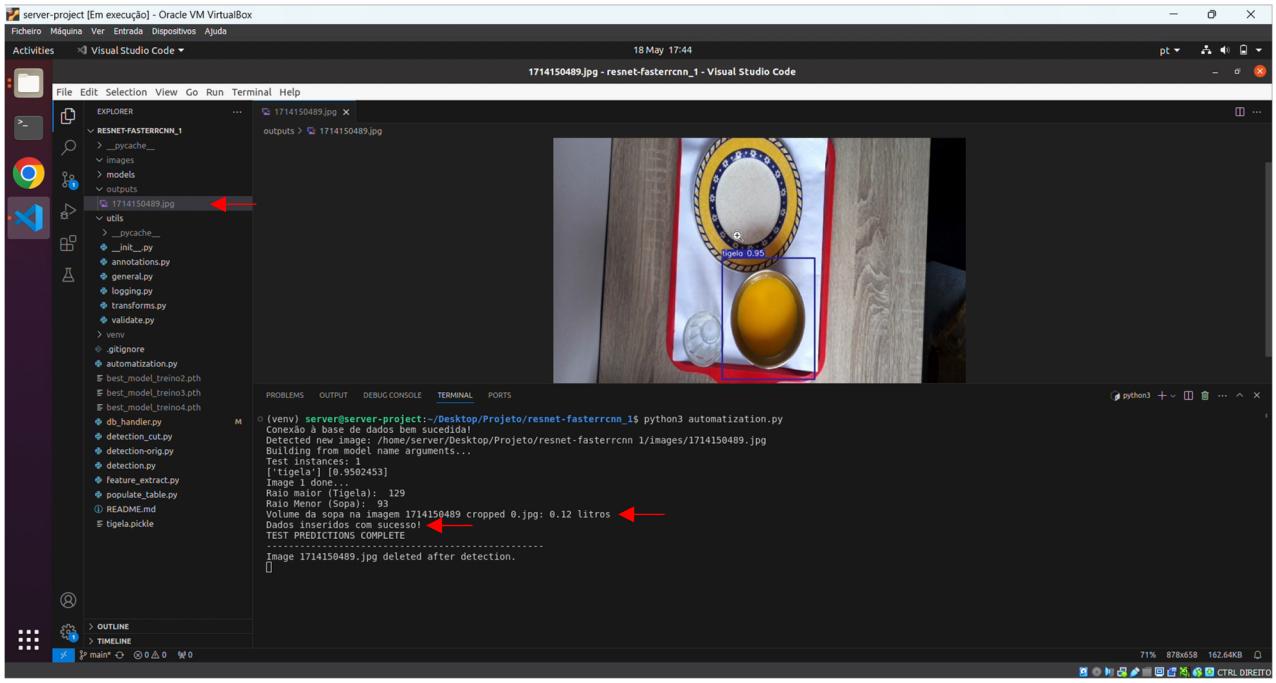


Figure 34. Detecting the bowl in the image and calculating the volume of wasted soup.

In the validation test carried out, the bowl contained a remaining volume of 0.12 L of soup. To compare the calculated value with the real value, the soup in the bowl was measured and the quantity obtained was 0.10 L. As explained in Section 2.3.4, this small discrepancy is due to the bowl not being a perfect truncated cone shape. Thus, there is an inherent margin of error in volume calculations. Finally, these data are sent and stored in the database.

Once these processes are finished, the cycle starts all over again and the system waits for a new image to arrive at the server to start processing once more.

4. Conclusions

Reducing food waste is not just an option, but an ethical responsibility with a temporal urgency. Every meal recovered helps to tackle important challenges, with significant consequences in the economic, environmental, and social spheres. Understanding the urgent need to solve these problems is fundamental, especially considering the large amount of food lost along the supply chain, accentuated by the habits of retailers and consumers.

This work has explored the use of computer vision and artificial intelligence to identify and analyze patterns that can guide strategies to effectively reduce food losses. The main aim of this paper was to propose, implement, test, and validate a prototype that makes it possible to identify and quantify soup waste in an institutional canteen, contributing to the development of strategies that lead to a reduction in its waste.

The main contributions of the work presented in this paper are as follows: the creation of a dataset adapted to the scenario and food in question [23]; the training and validation of the ResNet-50 model combined with the Faster R-CNN model for the detection; the study of the extraction of characteristics from images; the definition of an architecture, as well as the hardware and software components of a low-cost prototype, based on Internet of Things and computer vision (CNNs); the description of the prototype's implementation and configuration process; the development of a web application based on microservices to visualize the data collected; and finally, the validation and proof-of-concept tests carried out on the prototype.

The prototype proved to be effective, as it achieved the objectives proposed in its design. It is considered an economically viable and functional option. Together with the

application, it can be used to support decision making and contribute to more efficient management, with the aim of reducing food waste.

Based on our results, we identify several opportunities for future work. To replicate this solution, it is necessary to adapt it to the environment in which it will be used. This includes creating a dataset suitable for the dishes that are sold, training the CNN with the dataset created, and adapting the hardware to the space in which it is used, for example, by adjusting the location of the sensor and camera.

As for feature extraction, this also needs to be adapted to the implementation context. In the case of this solution, it was developed to detect circles and the color of soup to obtain an estimate of the soup's volume. It is important to recognize its limitations and consider possible error factors, such as inaccuracies in the detection of circles and the conversion of pixels to centimeters. This form of feature extraction can easily be replicated for other foods served in bowls, such as desserts, but would need to be adapted to other types of food served in other formats, such as plates.

When training the Resnet-50 and Faster R-CNN model, the performance metrics used to evaluate it show extremely positive results since the model is working in a very controlled environment, where all the images in the dataset come from a single canteen, resulting in great similarity between them. To improve the model's performance, it would be advantageous to diversify the training environment by including images from several different canteens or food environments, which would help the model to generalize better to different scenarios. Additionally, increasing the diversity of the dataset, by incorporating a variety of lighting conditions, viewing angles, and food arrangements on the trays, would also be beneficial.

Furthermore, the web application developed to visualize the data collected could be adapted to the needs of users, offering better support for decision making. This can include customizations to the interface, more advanced analysis tools, and integration with other existing systems or processes in the canteen, for even more efficient food waste management.

Author Contributions: Conceptualization, A.C. and C.A.; methodology, A.C. and C.A.; test, J.M.L.P.C. and V.N.G.J.S.; formal analysis, J.M.L.P.C. and V.N.G.J.S.; investigation, A.C. and C.A.; writing—original draft preparation, A.C. and C.A.; writing—review and editing, J.M.L.P.C. and V.N.G.J.S.; supervision, J.M.L.P.C. and V.N.G.J.S.; funding acquisition, J.M.L.P.C. and V.N.G.J.S. All authors have read and agreed to the published version of the manuscript.

Funding: J.M.L.P.C. and V.N.G.J.S. acknowledge that this work is funded by FCT/MCTES through national funds, and when applicable, co-funded by EU funds under the project UIDB/50008/2020.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Gustavsson, J.; Cederberg, C.; Sonesson, U.; van Otterdijk, R.; Meybeck, A. *Global Food Losses and Food Waste: Extent, Causes and Prevention*; Food and Agriculture Organisation of the United Nations: Rome, Italy, 2011; Volume 365.
2. Instituto Nacional de Estatística. *Desperdício Alimentar-Resultados e Perspetivas*; Instituto Nacional de Estatística: Lisbon, Portugal, July 2022.
3. Tribunal de Contas Europeu. *Relatório Especial-Luta Contra o Desperdício Alimentar: Uma Oportunidade para a UE Melhorar a Eficiência dos Recursos na Cadeia de Abastecimento Alimentar*; Tribunal de Contas Europeu: Luxembourg, 2016.
4. United Nations. Ensure Sustainable Consumption and Production Patterns. Available online: https://sdgs.un.org/goals/goal12 (accessed on 5 October 2023).
5. Correia, A.; Aidos, C.; Caldeia, J.M.L.P.; Soares, V.N.G.J. Using Computer Vision for Reducing Food Waste in an Institutional Canteen. *Waste, Submitted, pending review*.
6. Raspberry Pi. Available online: https://www.raspberrypi.com/ (accessed on 21 May 2024).

7. Enterprise Open Source and Linux|Ubuntu. Available online: https://ubuntu.com/ (accessed on 21 May 2024).
8. MySQL. Available online: https://www.mysql.com/ (accessed on 21 May 2024).
9. Docker Desktop: The #1 Containerization Tool for Developers|Docker. Available online: https://www.docker.com/products/docker-desktop/ (accessed on 17 May 2024).
10. Microsserviços e o Impacto na Escalabilidade de Aplicações. Available online: https://imaginedone.com.br/artigos/inovacao-e-tecnologia/microsservicos/ (accessed on 21 May 2024).
11. Canvas Select Plus microSD Card, A1, Class 10 UHS-I, 64GB to 512GB-Kingston Technology. Available online: https://www.kingston.com/en/memory-cards/canvas-select-plus-microsd-card (accessed on 18 May 2024).
12. Buy a Raspberry Pi Camera Module 3–Raspberry Pi. Available online: https://www.raspberrypi.com/products/camera-module-3/ (accessed on 18 May 2024).
13. Infrared Reflective Sensor-Waveshare Wiki. Available online: https://www.waveshare.com/wiki/Infrared_Reflective_Sensor (accessed on 18 May 2024).
14. Camera-Raspberry Pi Documentation. Available online: https://www.raspberrypi.com/documentation/accessories/camera.html (accessed on 9 May 2024).
15. How To Use Dual Cameras on the Raspberry Pi 5|Tom's Hardware. Available online: https://www.tomshardware.com/raspberry-pi/how-to-use-dual-cameras-on-the-raspberry-pi-5 (accessed on 18 May 2024).
16. Raspberry Pi hardware-Raspberry Pi Documentation. Available online: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html (accessed on 18 May 2024).
17. Raspberry Pi 5-gpiod vs. RPi.GPIO-Raspberry Pi Forums. Available online: https://forums.raspberrypi.com/viewtopic.php?t=359742 (accessed on 18 May 2024)..
18. Using a IR Reflective Sensor-Raspberry Pi Forums. Available online: https://forums.raspberrypi.com/viewtopic.php?t=181544 (accessed on 9 May 2024).
19. 2. Basic Recipes—Gpiozero 2.0.1 Documentation. Available online: https://gpiozero.readthedocs.io/en/stable/recipes.html#pin-numbering (accessed on 18 May 2024).
20. Oracle VM VirtualBox. Available online: https://www.virtualbox.org/ (accessed on 18 May 2024).
21. Instituto Politécnico de Castelo Branco. Available online: https://www.ipcb.pt/ (accessed on 21 May 2024).
22. Roboflow: Computer Vision Tools for Developers and Enterprises. Available online: https://roboflow.com/ (accessed on 15 May 2024).
23. Soup-Bowl-Dataset Dataset > Overview. Available online: https://universe.roboflow.com/sopa/soup-bowl-dataset (accessed on 4 June 2024).
24. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 3rd ed.; O'Reilly Media: Sebastopol, CA, USA, 2022.
25. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 53. https://doi.org/10.1186/s40537-021-00444-8.
26. Maurício, J.; Domingues, I.; Bernardino, J. Comparing Vision Transformers and Convolutional Neural Networks for Image Classification: A Literature Review. *Appl. Sci.* **2023**, *13*, 5521. https://doi.org/10.3390/app13095521.
27. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv* **2015**, arXiv:1506.01497. Available online: http://arxiv.org/abs/1506.01497 (accessed on 21 May 2024).
28. Google Colab. Available online: https://colab.research.google.com/drive/1thXa8nF65Iywbcbm4LaZ_KDD1bkIvi9O?usp=sharing (accessed on 9 May 2024).
29. sovit-123/fasterrcnn-pytorch-training-pipeline: PyTorch Faster R-CNN Object Detection on Custom Dataset. Available online: https://github.com/sovit-123/fasterrcnn-pytorch-training-pipeline (accessed on 9 May 2024).
30. Personal Cloud Storage & File Sharing Platform-Google. Available online: https://www.google.com/drive/ (accessed on 24 May 2024).
31. GitHub: Let's build from here · GitHub. Available online: https://github.com/ (accessed on 24 May 2024).
32. Overfitting and Underfitting in Machine Learning|SuperAnnotate. Available online: https://www.superannotate.com/blog/overfitting-and-underfitting-in-machine-learning (accessed on 21 May 2024).
33. Ying, X. An Overview of Overfitting and its Solutions. *J. Phys. Conf. Ser.* **2019**, *1168*, 022022. https://doi.org/10.1088/1742-6596/1168/2/022022.
34. What is Average Precision in Object Detection & Localization Algorithms and how to calculate it?|by Aqeel Anwar|Towards Data Science. Available online: https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b (accessed on 21 May 2024).
35. mAP (mean Average Precision) for Object Detection|by Jonathan Hui|Medium. Available online: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173 (accessed on 21 May 2024).
36. Mudivedu, V. What Is the Difference between Training Loss Validation Loss and Evaluation Loss. Medium. Available online: https://medium.com/@penpencil.blr/what-is-the-difference-between-training-loss-validation-loss-and-evaluation-loss-c169ddeccd59 (accessed on 18 January 2024).
37. Raspberry Pi OS–Raspberry Pi. Available online: https://www.raspberrypi.com/software/ (accessed on 9 May 2024).

38. How To Use Picamera2 to Take Photos With Raspberry Pi|Tom's Hardware. Available online: https://www.tomshardware.com/how-to/use-picamera2-take-photos-with-raspberry-pi (accessed on 17 May 2024).
39. rpi-gpio-example/example.c at master · krinkinmu/rpi-gpio-example. Available online: https://github.com/krinkinmu/rpi-gpio-example/blob/master/example.c (accessed on 9 May 2024).
40. Paramiko- How to SSH and Transfer Files with Python|by Mokgadi Rasekgala|Medium. Available online: https://medium.com/@keagileageek/paramiko-how-to-ssh-and-file-transfers-with-python-75766179de73 (accessed on 9 May 2024).
41. python-How to Transfer a File to ssh Server in an ssh-Connection Made by Paramiko?-Stack Overflow. Available online: https://stackoverflow.com/questions/11499507/how-to-transfer-a-file-to-ssh-server-in-an-ssh-connection-made-by-paramiko (accessed on 9 May 2024).
42. Python: Event Monitoring with Watchdogs|by Pravash|Medium. Available online: https://pravash-techie.medium.com/python-event-monitoring-with-watchdogs-86125f946da6 (accessed on 17 May 2024).
43. PyTorch. PyTorch 2.1 documentation. Available online: https://pytorch.org/docs/stable/index.html (accessed on 29 December 2023).
44. NumPy -. Available online: https://numpy.org/ (accessed on 17 May 2024).
45. CV2-Master Guide OpenCV Made for Python Developers. Available online: https://konfuzio.com/en/cv2/ (accessed on 17 May 2024).
46. os—Miscellaneous Operating System Interfaces—Python 3.12.3 Documentation. Available online: https://docs.python.org/3/library/os.html (accessed on 17 May 2024).
47. The Official YAML Web Site. Available online: https://yaml.org/ (accessed on 17 May 2024).
48. Matplotlib—Visualization with Python. Available online: https://matplotlib.org/ (accessed on 17 May 2024).
49. Copy—Shallow and Deep Copy Operations—Python 3.12.3 Documentation. Available online: https://docs.python.org/3/library/copy.html (accessed on 17 May 2024).
50. Math—Mathematical Functions—Python 3.12.3 Documentation. Available online: https://docs.python.org/3/library/math.html (accessed on 17 May 2024).
51. Coding Gaussian Blur Operation in Python From Scratch|by Rohit Krishna|Medium|Medium. Available online: https://medium.com/@rohit-krishna/coding-gaussian-blur-operation-from-scratch-in-python-f5a9af0a0c0f (accessed on 21 May 2024).
52. Implementing the Hough Transform from Scratch|by Alberto Formaggio|Medium. Available online: https://medium.com/@alb.formaggio/implementing-the-hough-transform-from-scratch-09a56ba7316b (accessed on 21 May 2024).
53. Truncated Cone Volume Calculator. Available online: https://www.omnicalculator.com/math/truncated-cone-volume (accessed on 17 May 2024).
54. mysql-Official Image|Docker Hub. Available online: https://hub.docker.com/_/mysql (accessed on 17 May 2024).
55. Docker Hub Container Image Library|App Containerization. Available online: https://hub.docker.com/ (accessed on 21 May 2024).
56. Java | Oracle. Available online: https://www.java.com/en/ (accessed on 21 May 2024).
57. Spring Boot. Available online: https://spring.io/projects/spring-boot (accessed on 17 May 2024).
58. 2. Service Discovery: Eureka Server. Available online: https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-eureka-server.html (accessed on 17 May 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.