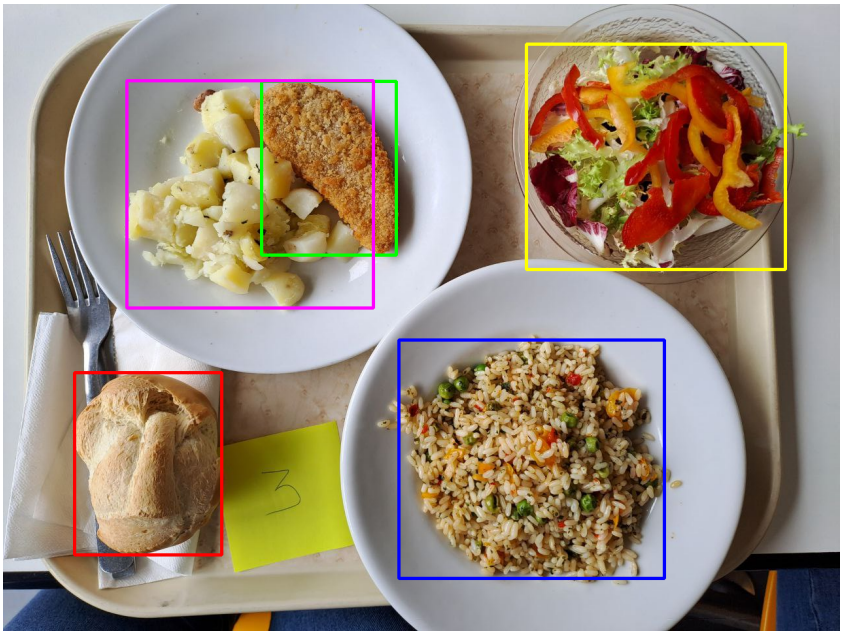
UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DEPARTMENT OF COMPUTER ENGINEERING

FOOD RECOGNITION AND LEFTOVER ESTIMATION



COMPUTER VISION PROJECT

DAVIDE DELRIO, TRENTI STEFANO, MATTEO VILLANI

2022 – 2023

1 Introduction and member contributions

The aim of this project was to develop a computer vision system to analyze a food tray from a canteen at the beginning and at the end of a meal to estimate the amount of leftovers for each type of food. The system will have to recognize the different foods on the plates and eventual side dishes outside of the plates for a combination of foods and tray compositions.
The following contributions for the project were previously agreed by the members of the group.
Delrio Davide(130 hours):

- Food and Leftover Recognition
- All functions in Detector_lib.cpp/hpp, Plate_rec.cpp/hpp

Trenti Stefano(130 hours):

- Food Localization and Segmentation
- Data structure
- Main program algorithm
- All functions in Utilities.cpp/.h, Classes.h

Villani Matteo(90 hours):

- Performance measure and analysis
- All functions in mAP.cpp, mAP.h

2 Data structure(Stefano)

To help structuring the program three new object types were created to store all the information. These objects are structured as the problem in question where there is a tray containing two plates and optionally two other food items (bread, salas). We then defined a new object type `tray` that has two objects of type `plate` (firstCourse and mainCourse) and two objects of type `food` (bread and salad). Additionally it has a trayImage of type `cv::Mat` representing the image of this tray and two `bool` (hasBread and hasSalad) that are true when bread or salad are present in the tray.

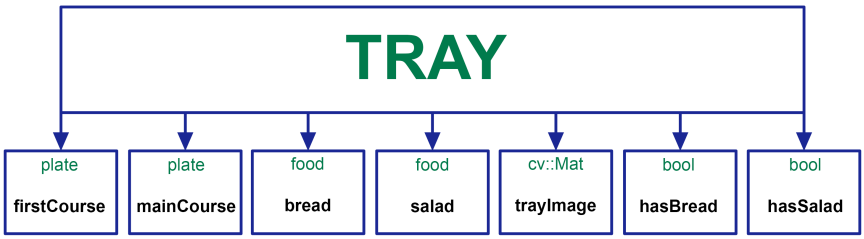


Figure 1: Tray object structure

The object of type `plate` represent each plate with an image `cv::Mat` plateMask containing a mask that is white only for the pixels of this plate, a `cv::Mat` plateImage which is an image with only the pixels from the plate and everything else black, a `bool` value that tells if the plate is empty and finally a `vector<food>` containing all the items of type `food` in this plate.

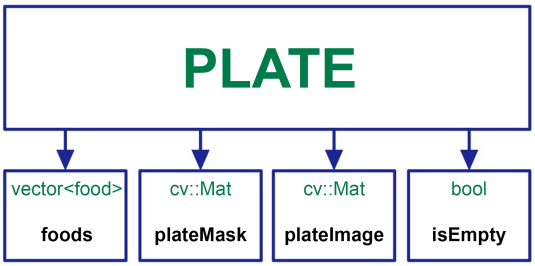


Figure 2: Plate object structure

Finally the object of type `food` is one that has three objects: a `cv::Mat` foodMask containing the binary mask of this food, an `int` called ID which corresponds to the food item as in the `food_categories.txt` provided file and finally a `std::map <int , vector<int>>` bbox that stores the bounding box parameters in the same way as presented in the project PDF so (x,y,width,height) and is a map accessed through the ID of the food.

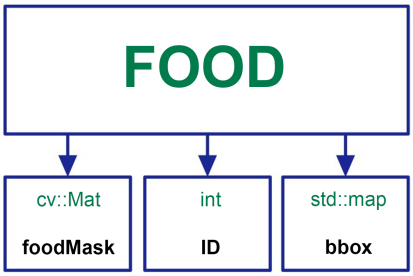


Figure 3: Food object

This data structure is very useful as it let us populate the `tray` object with all the useful information for our analysis such as recognized food ID's, food masks and bounding boxes from food segmentation all in a single object that can be passed to different functions preserving all the information about the previously analyzed tray image. These objects and their definitions are all in the `Classes.h file` inside the Utilities folder.

3 Main algorithm structure and Food Segmentation(Stefano)

Now that we have a data structure to work on we can begin explaining the main steps and procedure of the algorithm. The program starts by asking the user to input a path for the tray image that you want to analyze, it then starts the analysis of this image though the function `generateTray()`.

Figure 4: Example: input image for tray 4

This function detects the two plates in the tray with the function `getPlateMask()` and create two sub images containing only the pixels in each plate with the function `squareMask()`.

Figure 5: Sub image of plate 0

Figure 6: Sub image of plate 1

For each plate sub image it detects the food present in the plate with the function `processImages()` which returns for each plate an object `plate` initialized with foods having the ID's of the food item present in the plate which is then saved into either the `tray.mainCourse plate` if the ID correspond to main courses food items (fish, pork ...) or in the `tray.firstCourse`. For example for the sub image of plate 0 it returns a `plate` with two `food` objects one with ID 11 (basil potatoes) and the other with ID 7 (fish cutlet) which means that it is a mainCourse. Knowing now the ID's of the foods present in the plate we can segment the plate image to obtain the masks and bounding boxes for all `food` objects in the `plate` object. We start by segmenting the first plate which can contain different types of pasta or rice, this is achieved by the function `segFirst()` which takes in input the sub image of the plate corresponding to the first course previously saved and returns a binary mask containing only the pixel of the detected first course food. It does so by transforming the image to HSV format and applying a threshold which removes the pixels with a low saturation as these correspond to the plate pixels. We can do so because the first course is a single food item and since we are in a canteen we can assume to always have these white plates and the lighting to be controlled. We can see the result of this segmentation in the figure below.

Figure 7: Segmented rice mask applied to the tray image

The we can segment all the foods present in the main course plate. To do so we first segment the sides that are present in the plate with the function `segSidesFirst()`, in this case the only side present are basil potatoes. This function segments the sides present on the plate, gather their masks and removes the corresponding pixels from the plate image so that the following function `segMainSecond()` can ignore them and concentrate on the pixels that are left which are corresponding to the pixels of the plate, of the main foods (fish,pork,rabbit,seafood salad) or noise leftover from the side segmentation. To segment the sides the `segSides()` function is called which in turns call the functions `segBeans()` (which segments the beans and returns the corresponding mask) and `segPotatoes` (which does the same for potatoes). In this case the result from the `segPotatoes` is shown below with the resulting plate image where the segmented pixel corresponding to the potatoes are missing.

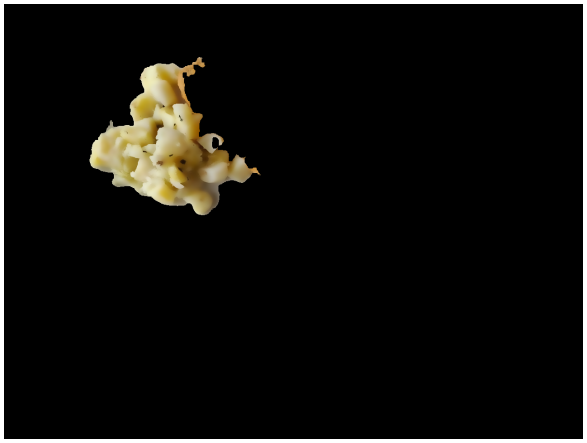


Figure 8: Segmented potato mask applied to tray image

Figure 9: resulting sub figure with no potatoes

The `segPotatoes()` function works similarly as the `segFirst()` function as it converts the image to HSV and thresholds it with Hue, Saturation and Value values corresponding to potatoes. After obtaining the modified sub image the main food can be segmented with the function `segMain()` which knowing the ID of the food decides how to threshold the modified sub image in the HSV color space giving us the following result. All the HSV threshold values were obtained with a script called HSVColorPicker.cpp created for this purpose.

Figure 10: Segmented fish mask applied to the tray image

In the case of beans the function `segBeans()` is far more complicated, it uses 3 different sub functions to do the following steps:

1. removes pixel that are certainly not beans with the `beanSelector()` function that applies a threshold
2. equalize the histogram of the tray image and apply the previously obtained mask
3. applies a second threshold which aims at removing all the edges between beans as to obtain a small blob for each bean with the function `eqbBeanSelector()`
4. it then creates a mask from these cleaning up the blobs that are too small or too big by analyzing all the connected components
5. it applies the mask of each single connected component to the histogram equalized image and then averages the pixel value of non zero pixel and removes the components that have an average value too far from one corresponding to beans
6. it finally uses dilation operations to enlarge these blobs to cover the beans edges.

The result through the various steps of the `segBeans()` function for tray 8 are shown in the figures below.

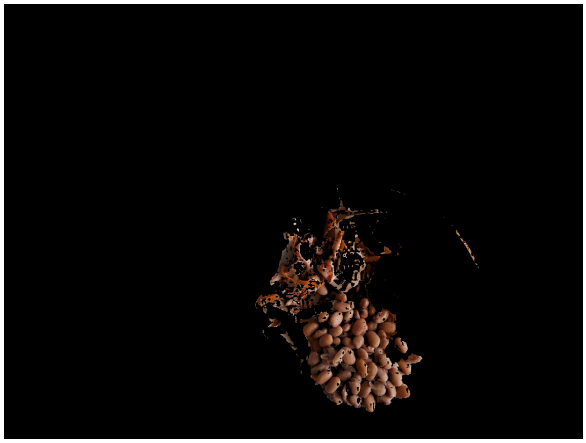


Figure 11: Result after step 1

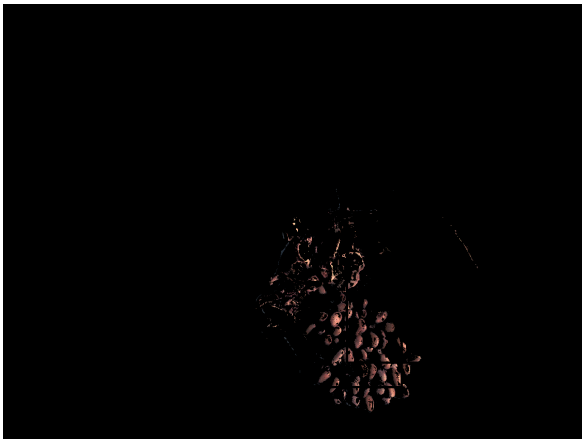


Figure 12: Result after step 3

Figure 13: Result after step 4

Figure 14: Result after step 6

Many different methods to segment foods where tested through the development of this project such as kmeans clusterization, meanshift, region growing with edge detection but the one chosen here yielded the best and most consistent results .

At last here it is shown the result for food segmentation on the main course of the tray 8.

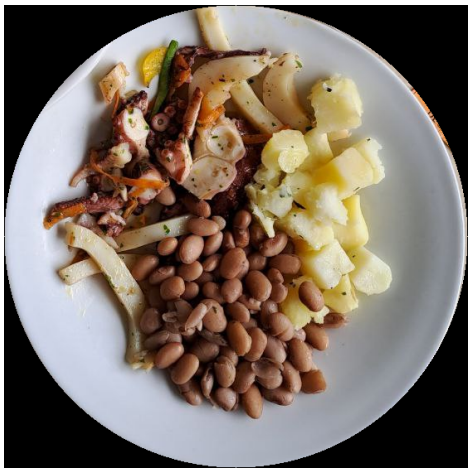


Figure 15: main course image of tray 8

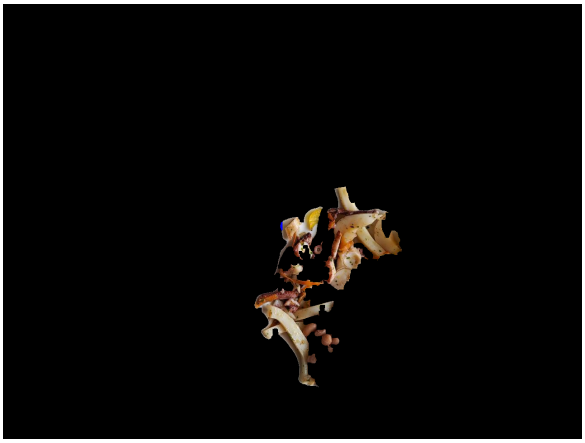


Figure 16: seafood salad segmentation

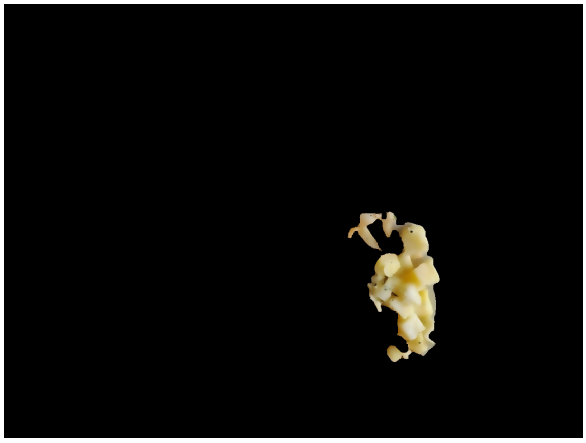


Figure 17: potatoes segmentation

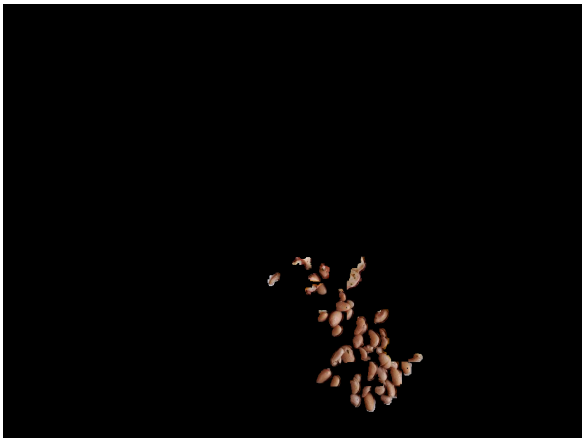


Figure 18: beans segmentation

After segmenting the foods in the plate we can now look at side dishes such as salad or bread. Salad is segmented with the function `getSaladMask()` which locates the salad bowl with the hough circles function like for the plates (we can assume that the salad bowl will always be the same for a canteen) and yields the following result.

Figure 19: salad bowl obtained with the `getSaladMask()` function

This image is then looked over by the function `is_this_salad()` which returns a boolean value of true when it detects that there is salad in the image or false otherwise. If there is salad in the image we segment this obtained image to only include the pixel corresponding to salad with the `cutSalad()` function which once again thresholds the HSV image for values of high saturation yielding the following result.

Figure 20: salad segmentation

It is worth noting that all mask obtained through this type of segmentation are then "cleaned" by removing small independent blobs and by filling the holes inside it that may occur. The segmentation of the bread posed a much greater challenge since the color of the bread is very similar to the color of the tray on which it is placed on. The function `segBread()` uses 5 different sub functions and does the following steps:

1. filters the image and threshold it to a particular color of the bread as to obtain with certainty groups pixels that are in the bread
2. keeps only the largest connected component from which it obtains the rough center poin
3. from this point it starts region growing with the `floodFill()` function obtaining a mask with however a small spill caused by the color similarity
4. from this mask we create a sub image which is then evaluated with the function `is_there_bread()` which returns a true bool if it recognize bread
5. the sub image is then quantized in colors for a specific range of colors like in the function kmeans with the function `BreadQuantize()`
6. a range of colors corresponding to bread is then selected and after some blob removal with `breadCleanup()` the final mask is obtained

The result for the various steps of this function are shown in the figures below.

Figure 21: mask obtained by region growing

Figure 22: bread localization after thresolding

Figure 23: color quantized and thresholded bread

Figure 24: Final bread mask

Finally after segmenting all the different food items with all the different functions all the various images and masks are stored in the `tray` object on their corresponding plates and/or foods. Also for every food item the bounding box is created and stored. We can see all the obtained bounding boxes printed on the initial tray image in the figure below where the bounding boxes have different color for first course food, main course food, main course sides, salad and bread.

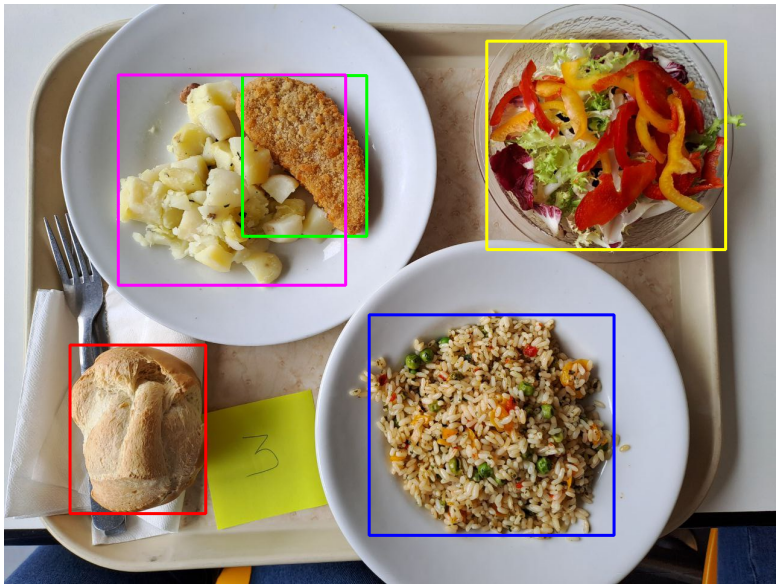


Figure 25: Tray 4 image with all found bounding boxes for every food item

Also the masks obtained for the various food items are shown in the figure below where the greylevel of each food mask correspond to the food ID.

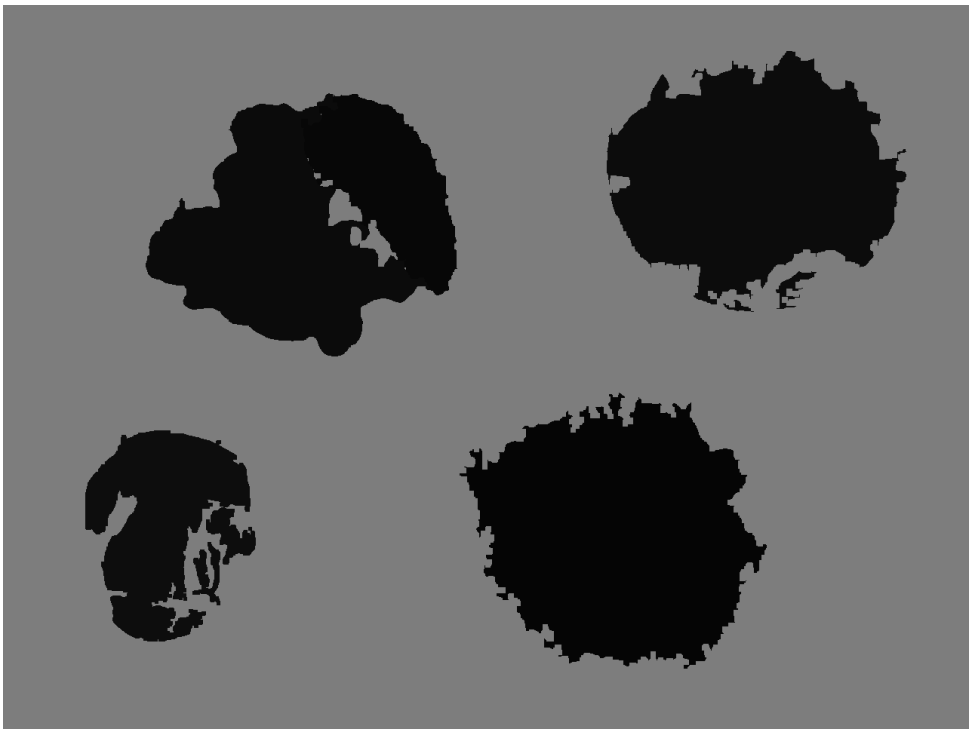


Figure 26: All masks found for tray 4

4 Food Recognition(Davide)

In this project it was developed a system of food recognition in each plate. The goal is that of identifying the type of food present in a determined part of the plate by taking advantage of two libraries of functions. The first library, named `detector_lib.hpp`, contains the main functions for the food recognition of the different types of food. The second library, `plate_rec.hpp`, handles the processing of images from the plate and to return the types of food recognized. **Description of the function libraries** `detector_lib.hpp` The library

`detector_lib.hpp` contains different functions for the recognition of foods inside the images of the plates. The main functions are:

- **map_label**: this function maps a char letter with a numerical value that represent the type of food. This representation is internal to the library and must be remapped.
- **load_images_from_folder**: This function loads the reference images from the specified path and associates each image with the respective numerical label.
- **sift_object_detection**: This function uses the SIFT (Scale-Invariant Feature Transform) algorithm to detect matches between the keypoints of the plate image and the reference images. The number of good matches and the total number of matches for each type of food is calculated, 2 variants of this function are also available: sift_object_detection_label_selected and sift_object_detection_reference_img, whose operation is the same but the first one searches only for a precise set of labels, while the second does not use the standard database but a series of images passed as an argument.

- **search_for_salad**$_b read : This function is used to locate the bread or salad within the passed image.$**search_for** $This function is used to identify the food inside the dish using the sift_object_detection and sift_object_detecti$

`plate_rec.hpp:` The plate_rec.hpp library deals with the processing of the plate images and the recognition of the foods that are present. The main functions are:

- **load_images_from_folder_l**: This function loads reference images from the specified path.
- **get_center_color**:This function calculates the average color of the center region of the plate image, used to estimate which foods might be on the plate.
- **processImages**: This function uses the detector1_lib.hpp library and the color estimates to determine what food is on the plate. Based on the predominant color, specific searches are carried out to recognize specific types of food.
- **is_this_salad** and **is_this_bread**: These functions detect whether the food on the plate is salad or bread, using case-specific reference images.
- **process_leftover**: This function identifies the dishes (leftovers) in the tray based on those already found previously. More specifically, it identifies which of the passed images is the first course.

Operation of the food recognition system The food recognition system works as follows:

- Once the processImages call has taken place, the central color is checked for each image passed.
- Plausible foods to be searched for using the search_for function will be selected based on the predominant color.
- Based on the IDs found, an object of class "plate" will be filled with the public IDs of the corresponding food, this will be the output.

The is_this_salad and is_this_bread functions are used only in the specific case where we need to detect whether the food is salad or bread. These 2 functions perform the following steps:

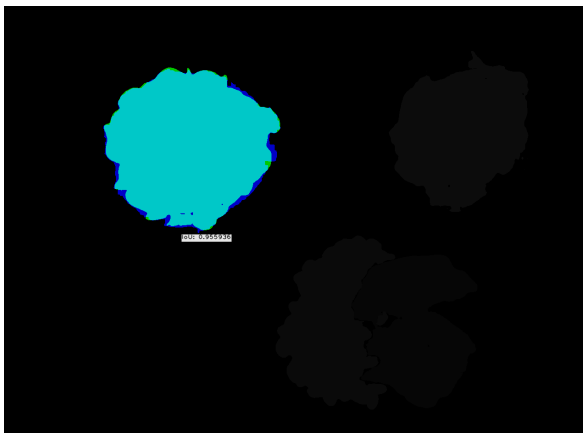
- Starts by running the SIFT algorithm with the corresponding database of images for the bread or the salad
- In the case that correspondence was found the functions return a boolean value, true if there was correspondence and false otherwise

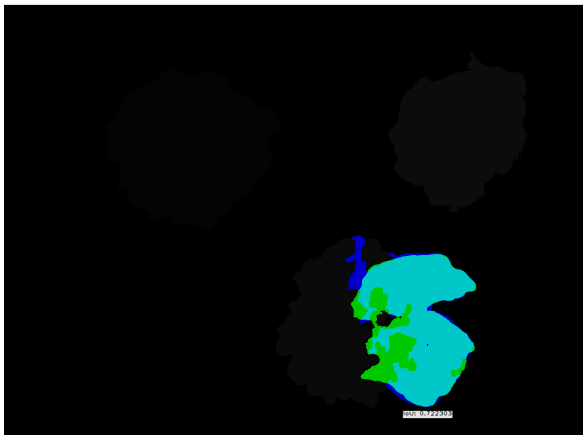
During the development of this model, several paths were taken. Initially, I tried to use a pre-trained model for food classification. Unfortunately, the pre-trained models did not produce satisfactory results for food image classification. The performance was below expectations, probably the amount of images provided was not sufficient. Then I tried to implement a "Bag of Words" but this also did not have the desired results, obtaining an error rate that was too high for the goal to be achieved. So I took a more basic approach using colors and SIFT for food classification. In tray 8 leftover 2, the process_leftover function returns an incorrect id

because the dish with the octopus has much more red compared to the first, and looking at the blue channel, it has a low enough intensity to pass. By modifying the values to exclude the octopus in this tray, I would have increased errors in other trays

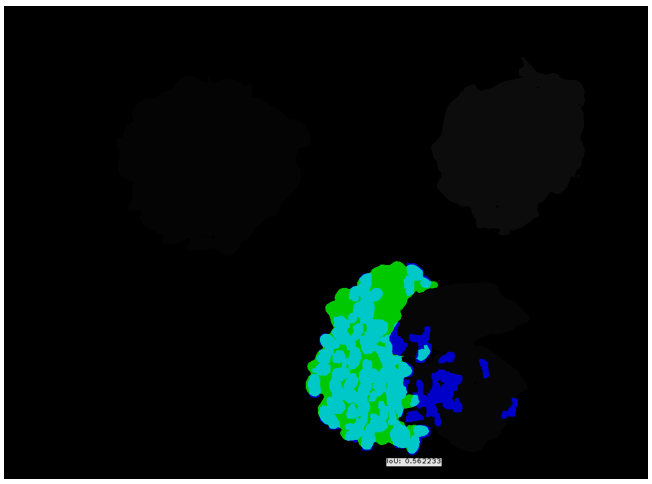
5 Performance analysis and evaluation (Matteo)

All the functions to perform performance analysis and evaluation are in the mAP.cpp, mAP.h files. MAP To compute Mean Average Precision (mAP) we analized the bounding box for each food item in the tray, the mask selected are the 'food_image_mask.png' for the ground trouth and the bounding box for each food item. The 'plot..' functions displays the current bounding box, meanwhile ComputeIntersections compute the pixels that are in common. After getting the intersection count we compute the Iou with 'getIou', and based on those values we can define the matches (TP, FP, TN) of each segment. Due to the deadline this part was not completed, we wanted to consider for each tray 3 classes: 1 considering the First Course; 1 considering the Main course and the 3rd considering the sides. For each comparison (normal, leftover 1, leftover 2) and for each class we were about to compute the Average precision, and thus the Mean Average Precision. The mean Intersection over Union (mIoU) The mean Intersection over Union was a bit challenging: for each food item for each detection we computed the Iou getting the ID value of the grayscale mask obtained by detecting all the items in the tray. After obtained the ID we got the IoU using the 'getioUFrom_masks' function. To get the mean of all the ratios computed we stored in an array all the values for each tray and then devided by the number of successful detection. Mask evaluation between ground truth and detected masks from tray image in tray6:

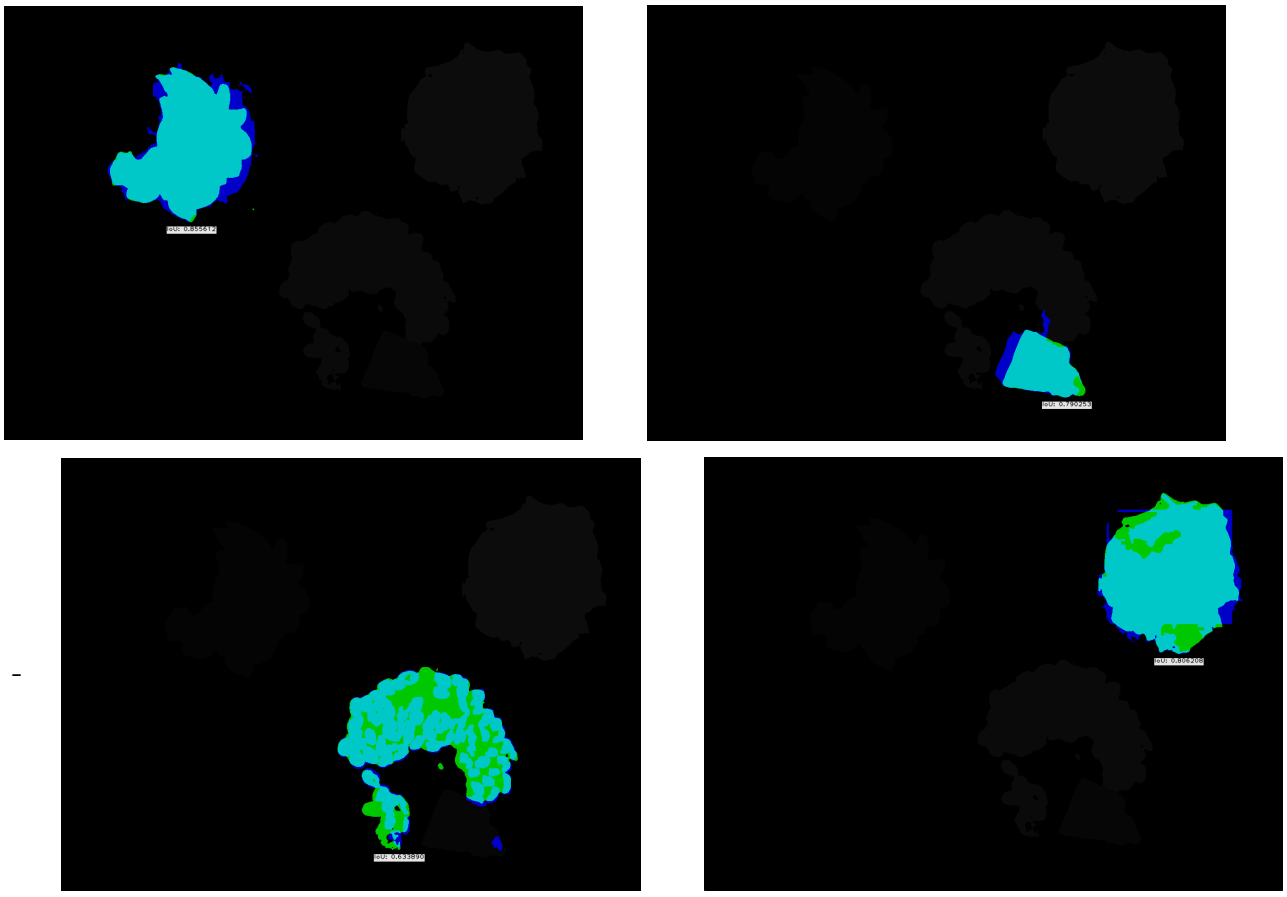




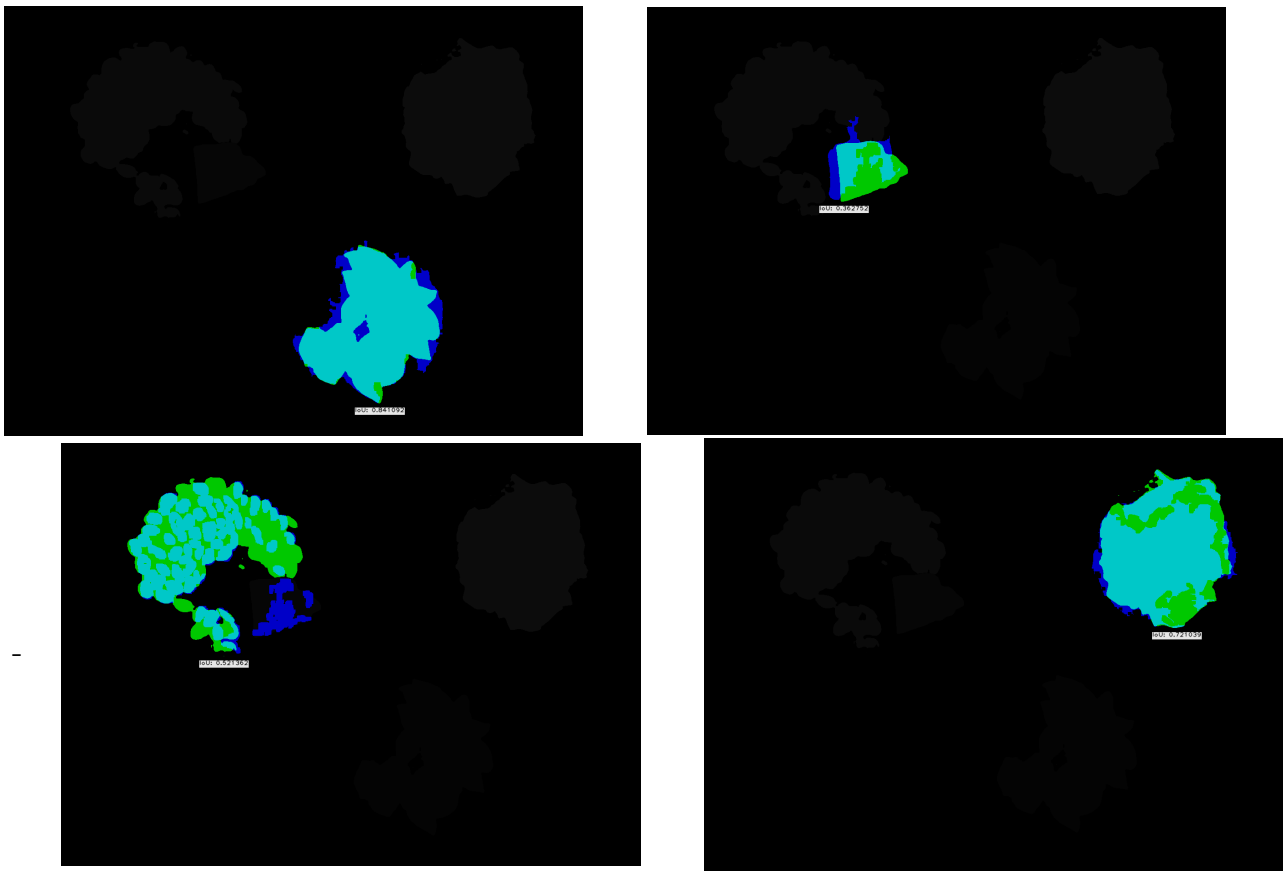
-



Mask evaluation between ground truth and detected masks from leftover1 in tray6:



Mask evaluation between ground truth and detected masks from leftover2 in tray6:



RATIO To get the ratio we used the formula R = 'pixel after'/'pixel before' , once again, for each item, for each detection. The function 'getRatio' computes the division for given 2 masks in input: the one from the ground trouth and the other detected by our algorithm. The ratio compares the pixels with the same gray level in the mask, that we called ID. All the functions necessary to compute the performance measurements are located in the mAP.h and mAP.cpp with some brief comments. For food leftover estimation is the quantity of food leftover estimated from the predicted masks and the difference between your estimated quantity of food leftover from the leftover estimate masks and the quantity computed from the ground truth. The quantity of food leftover is defined as the ratio R_i (in terms of pixels) between the segmentation mask in the image after food comsumption and the segmentation mask in the initial image. Food leftover estimation results:

FOOD LEFTOVER ESTIMATION					
tray 1	tray		tray 5:	tray	0.75223
	left1	0.75216		left1	0.65133
	left2	0.37853		left2	0.24655
tray 2	tray		tray 6:	tray	0.75625
	left1			left1	0.26594
	left2			left2	0.18563
tray 4:	tray	0.65152	tray 7:	tray	0.45895
	left1	0.59451		left1	0.09621
	left2	0.10326		left2	
tray 4:	tray	0.65152	tray 8:	tray	0.26448
	left1	0.59451		left1	0.65915
	left2	0.10326		left2	0.35954

Figure 27: Food leftover estimation results

mIoU (mean Intersection over Union) is a metric used to evaluate the performance of semantic segmentation algorithms. It calculates the average overlap between predicted and ground truth segmentation masks, providing a measure of how well the model can correctly classify pixels while accounting for localization accuracy. Higher mIoU values indicate better segmentation performance. mIoU results:

TRAY 1:	food: 0.753698	TRAY 5:	food: 0.710007
	left1 : 0.610599		left1: 0
	left2: 0.578173		left2: 0.591284
TRAY 2:	food: 0.834372	TRAY 6:	food: 0.746824
	left1: 0.641204		left1: 0.617193
	left2: 0		left2: 0.72103
TRAY 4:	food: 0.854881	TRAY 7:	food: 0.54742
	left1: 0.72978		left1:0
	left2: 0.700731		left2: 0.305773
TRAY 3:	food: 0.721546	TRAY 8:	food: 0.611172
	left1: 0		left1: 0.58926
	left2: 0		left2: 0

Figure 28: mIoU results

The best tray found was the tray 6.

6 Conclusion

In conclusion the detection and segmentation of foods in a plate was successful for the most part, there were some cases such as leftover2 from tray 4 where the detection was unsuccessful because the photos were taken too close to the tray so the plates were bigger than the other images and also were cut off. When testing with the true ID's for the various plates the segmentation yielded for the most part good results for both initial trays and leftover trays. When instead adding the uncertainty of the predicted label for the food detected there were more issues with wrongly detected and consequently segmented foods. One of the most difficult foods to detect is bread, for the SIFT algorithm a small dataset of bread images was used but the result is not as good as we hoped, maybe a better dataset could improve the overall results. Comparing the ground truth masks for the initial tray images for all trays the mask found correspond very well in terms of location with the ones from the ground truth and in terms of pixel count are also pretty close. When looking instead at the leftover images, as the food recognition sometimes fails and returns incorrect food IDs, the masks segmented in a position for the wrong food ID obviously create a wrong mask and it is noticeable in some results. More over some leftover images were not successfully analyzed as the missing detection of some plates caused by variance in the images caused the program to unexpectedly crash and we did not manage to fix the issues in time. These images are marked in red in the mIoU result figure.