UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

# Computer Vision

## Final Project Report - Food Recognition and leftover estimation

Authors: Nicola Calzone, Riccardo Vendramin, Andrea Labelli

Professor: Stefano Ghidoni

20/07/2023

# 1 Introduction

The project consisted in the localization and classification of foods in a canteen consumer's food tray, knowing the possible thirteen possible foods, and comparing two images of the same tray before and after the meal, to measure the amount of leftover food for every food. Our approach exclude every use of Machine Learning and Deep Learning and it is based only pure image processing. We exploited several algorithms from literature, as Hough circle detection, SIFT descriptor, low level filtering, mixed with C++ object programming. Our program firstly creates foods templates, storing all the prior information of the foods, as label, id and some templates image, then in order to achieved the desired task, it goes through several steps: it detects dishes, it looks for every possible food in the tray, detecting, labeling and segmenting what it finds, then it compares the original image with the leftover one, looking for a match of the dishes and then of each food in the dish.
The output at this point is a couple of segmented areas, one related to the tray before lunch and the other after, identified by the food id.
At this point we compute the metrics, comparing bounding boxes and segmented area with the ground truth and computing the ratio of the leftover for every food.

# 2 Report structure

The report is composed of five main sections:

- Prior operations
- Dish Detection
- Classification
- Leftover matching
- Metrics

# 3 Prior operations

Before executing the algorithms we store the prior information of the possible foods in a struct foodTemplate, where for every food we save its id, its label, and a vector of template images, which will be used to compute matches.

- ```
  struct foodTemplate
  {
      std::vector<cv::Mat> foodTemplates;
      std::string label;
      int id;
  };
  ```

  All the food are saved into an array `"templates"`, and it will be crucial for the food classification task.

# 4 Dish Detection

Dish detection is performed following this idea: looks for circles in the image, discard the ones which cannot be a dish, and use the resulting circles as masks to generate one Mat object for every dish, and after a post processing step it stores them in an array of Mat objects called dishes, useful for every other further operation.

- Circle detection: it is performed using Hough Circles, after a Gaussian blur filter, on the input image of the tray before the meal. We use the function parameters to discard circles which are too small.
- Circle acceptance: since Hough doesn't assure to detect only the circles corresponding to the dishes, we use some basic geometry to discard "impossible dishes": before accepting a circle we check if the distance between its center and every already accepted circles' center is smaller than the biggest radius of the two circles. This technique let us to discard all the concentric circles.
- Dish image generation: simply using the accepted circles as masks, we can create a new image for every circle copying the original one only inside the circle.
- Post processing: in order to simplify food detection and food segmentation, before storing the dishes' images we remove all the empty dish parts, which correspond to the white areas. So we apply to each dish a color threshold to remove the dish areas. This is performed removing from the image the pixels in a tight gray range, for every possible gray range in the BGR color space. This cannot be performed with a single threshold because being in the BGR color space, setting a too large range would remove every color.

# 5 Classification

The classification part takes care of identifying the foods in the tray between the known ones. In order to achieve classification, the algorithm for every foods scans the dishes looking for a match. This can be found in two ways: for some foods it exploits colors, for others SIFT descriptors and KNN matching. Every food, once it has been found, is framed in a bounding box, is segmented, and it is labeled with the food label and its id. The output of this algorithm is an array of Dish objects: the idea is that a tray could have several dishes and every dishes could have several foods. So every Dish object stored the image of the dish, and an array of FoodData which represent the information we have about a particular food on that dish. foodData is a struct used to store information of a food, in particular, label, id, bounding box, segmented area.

## 5.1 How the algorithm works

- Main strategy: the algorithm scans all foods contained in the templates array mentioned above. For every food, it looks for the best dish, which is the dish where the food is found and validate it through several checks to avoid impossible situations.
- Compute the best dish: for specific food as for pasta, and potatoes, it is computed looking if the average color (after filtering out the yellow parts, to delete pasta in case there is) is in a specific range. For other food
  ```

types, it is computed looking for the dish with most good matches. The best dish is selected though an index "max_key", so that dishes[max_key] is the best dish for a specific food. If the food is not detected, max_key = -1.

- Exceptions: bread and salad are exceptions to the above strategy: we looks not in a dish but in the whole tray. This is achieved using specific color ranges (as for pasta and potatoes) but not in dish image. For the bread the algorithm looks for it outside every dish, and for salad it looks for it not in every dish but only in the one with smaller radius.

- Food bounding, segmenting and labeling: for specific food types as pasta, potatoes, salad and bread, bounding is achieved by putting a rectangle wrapping all the non-black pixels in the best dish or in the thresholded image, because these foods have a dish on their own. The same image used for bounding is the segmented area itself. For other foods, bounding is done by computing the good matches between the template images and the best dish, and creating a rectangle around all of them. Segmenting in this case is performed using k-means segmentation and taking all the non-black area in k is equals to two (background and food) and looking for the segment which contains more matches if k is bigger. Labeling is achieved by using the label and the id of the food we are looking at each iteration, stored in the templates array.

# 6 Leftover matching

The leftover matching part takes care of matching each plate inside of the tray in the original image with the corresponding leftover in the leftover image.
The images analyzed (both for original and leftover cases) are already preprocessed and have been through the removal of dish.
Now, since the leftover pictures present some problematics (e.g. there are less dishes than in the original tray, or a dish is completely empty) we used 4 methodologies, each of one bonds an original dish with a leftover dish, which must be different from all the other found bondings. Then, the picture which was predicted the most by the 4 methodologies is picked. How the final matching is picked is described in the next paragraph.
At the very end, we have a vector of `Couple` structs, each of which containing an original and a leftover dish. This vector is then given to a module called `createFinalPairs()` which has the responsibility to bond every segment of a food, found in the original dish to a segment found in the leftover dish. It constructs a vector of `SegmentCouple` structures which is given in output to the `main.cpp` ready to be used in the `metrics` module.

## 6.1 Matching Algorithm

The responsible function for the matching task is `jointPredictions()` which returns a vector of Couple of dishes, one original and one leftover.
Couples of dishes from the original tray on the leftover one are carried out checking four comparison metrics:

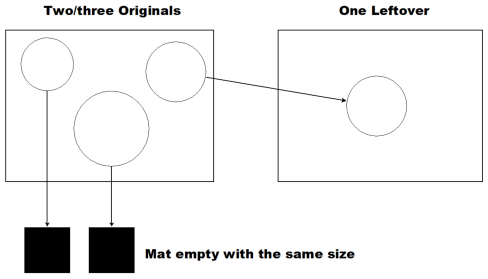
1. `coupleClosestElements()`: performs dish area comparison. It takes the circle areas of the original and leftover dishes and computes the couple with minimum distance in terms of the areas of the circles of dishes.

2. `coupleCIELABColors()`: computes original and leftover dishes with most similar CIELAB representations. Distance can be computed with DeltaE method or with norm method as for the BGR average method. This depends on a flag variable passed to the function and it seems to perform slightly better with Delta E method [Schuessler, 2022]

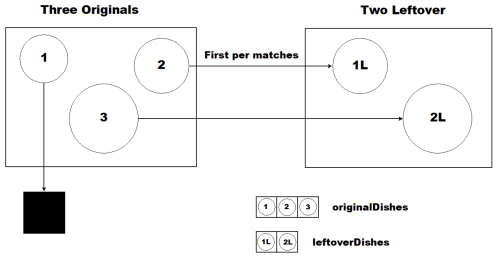
3. `coupleMaxMatches()` : checks which is the greater number of matches between the first, second or third dish (the last one if it is present) with the respective leftovers.

4. `coupleMinAverageColor()`: computes the minimum distance between average colors of the original and the leftover in the BGR space.

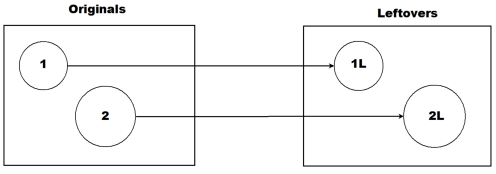
Finally, `jointPredictions()` takes the final pairs computed by each of the functions before, which act as if they are "voting" for the true final matches such that:
- all original dishes have the most predicted leftover dish coupled
- if a leftover has already been assigned it cannot be given anymore, even if it was predicted again on majority, so - in case this happens - the original dish will be coupled with the second-most preferred leftover.
- if there is only one leftover and there are more original dishes this is what happens:




- if there are 2 leftovers and there are 3 original dishes this is what happens:



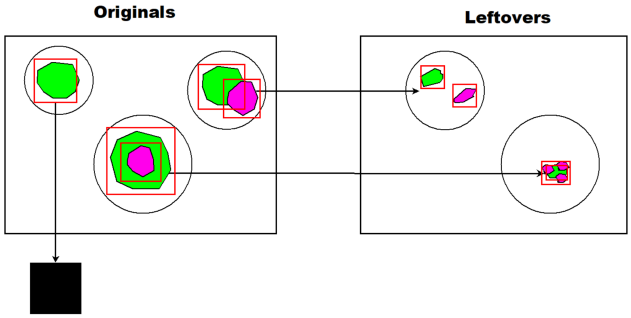

- finally, if there is the same number of leftovers and original dishes, this is what happens:




## 6.2 Last step before metrics

The food on original dishes is associated with its respective leftover thanks to the Bounding boxes; an image is provided

as an example:




# 7 Metrics

## 7.1 mAP

The mean Average Precision metric calculated at IoU threshold 0.5 is computed with the intersection of the GroundTruth BoundingBox with the predicted one, for each food item.

## 7.2 mIoU

The mean Intersection over Union stands for the average of the IoU computed for each segmented food item compared with the ground truth area of the mask provided for each food.
This stands for the average precision of each specific class, then is divided by the number of classes.

## 7.3 Ratio "after"/"before" images

The function computes the difference between the ratio of food leftover estimated and the ratio of the food leftover computed with ground truth data, for every food.

## 7.4 Performances of Classification

In this subsection the performances of classification are showed, one image for each tray; for a clearer view the images are shown at the end of the report.

## 7.5 Performances of Leftover Matching

In this subsection the performances of leftover matching are showed, one image for each segment of the corresponding tray; for a clearer view the images are shown at the end of the report.

# 8 How to run the code

In order to run the project the needed version of OpenCV is the 4.7.0. Moreover the complete code has only been tested on Ubuntu-18.04 wsl, as there may be problems with OpenCV version 4.6.0 on Windows 10.

# 9 Code

In this section, the structure of the project with its most important modules is provided, explaining each of them how they are composed and providing a brief explanation of what the most important functions do.
The code is also available on our GitHub page.

## 9.1 *Detect and Classification* module

:

- `detectAndCompute`:
  This function, for all the templates used, and for all the dishes detected, computes the food on dish that has the most number of matches with the template used, using `SIFT`.
  It is the main function of the detection and classification task. It looks for each food in all the dishes (received as input) performing several steps to assert if there is a given food in the tray and in which dish.
  It is a void function but it operates on an array of Dish objects, given in input.
  It iterates over each `foodTemplate` in the templates vector, representing a specific food type. For each `foodTemplate`, it searches for matches in each of the dishes images.
  It computes the number of matches for each dish and stores the results in the `dishesMatches` vector.
  It then selects the best matching dish based on the computed matches and some additional criteria, like the color range, for specific food types. Depending on the food type, different functions are called to bound the detected food items within bounding boxes and store relevant information in the `foodData` vector.
  - input: an image, a vector of Mat objects, a vector of integers, a vector of Vec3f, a reference to a vector of `FoodData`, a vector of `foodTemplate`;
  - output: an image with the detected food item;

- `computeBestDish`:
  - It initializes an integer variable `bestdishid` to -1. This variable will be used to store the index of the best matching dish. If no dish is found it will be returned -1, and managed it from the main algorithm.
  The function is called for every food, and iterates through each dish image. The best dish is computed in two ways, depending of the type of food: for some foods the choice is based on the average color, for the other on the number of good matches computed by Sift algorithm.
  In particular, identified the food with the id, if the average color of the dish is in a determined range that food is in that dish, so the function returns the index corresponding to the dish.
  In the other case, it looks for the dish with the highest number of good matches (stored in an array on int passed as parameter of the function) with the current food, and if the number is higher than a threshold, the food is found in that dish, and the dish index will be returned.
  - input: a foodTemplate, a vector of Mat objects, a vector of integers;
  - output: an integer;

- `boundPasta`:
  After some checks about the possibility of pasta presence in the dish, since if a dish contains pasta it cannot contain any other food, all the food in the dish is bounded.
  - input: a Mat object, a vector of strings, two vectors of integers, an integer;
  - output: a reference to the final output image, and a reference to the vector of `FoodData` struct;

- `boundBread`:
  Since the bread is not in a dish but directly on the tray, we use the already found dishes as masks to detect the tray region of the image. In this region it looks for bread though a color filtering.
  It creates a binary mask using the `cv::inRange` function to threshold the HSV image and extract regions with bread-like colors. The specific range is set between two HSV values: (12, 75, 180) and (25, 130, 255).
  After some post processing steps as erode and dilate, it obtains the mask of bread, and it use it to bound it and classify it though id and label.
  - input: a Mat object, a vector of Mat objects;
  - output: a reference to the final output image, and a reference to the vector of `FoodData` struct;

- `boundSalad`:
  This function iterates through the vector of `acceptedcircles`, which contains circles detected in the input image corresponding to dishes to find the circle with the smallest radius, and keep also its index. After a HSV conversion, two binary masks, maskRed and maskGreen, are created using the `cv::inRange` function.
  Masks are use to detect regions in those ranges, and if the result is not an array of zeros it means it found salad.
  So the function create the bounding box, the segmented area and classify the dish as salad.
  - input: a Mat object, a vector of Vec3f;
  - output: a reference to the final output image, and a reference to the vector of `FoodData` struct;

- `boundPotatoes`:
  Potatoes are bounded exploiting again their color: the given dish is thresholded in the HSV color space and the result of this filtering is bounded, if not empty.
  Then as usual, it is classified and segmented.
  - input: a Mat object, a vector of integers, an integer;
  - output: a reference to the final output image, and a reference to the vector of `FoodData` struct;

- `getCorrectSegment`:
  This function returns a Mat object containing the mask of the given food, when there is no other way to compute it, as in the case of more foods in a dish with no color detection implemented.
  It goes though the following steps: it receives as input the results of the segmentation, which are two masks, one for cluster.
  To assign the correct mask to the food, it uses the masks on the original images, creating two new images with only one segment not black.
  Then using sift on both images and counting the matches with the food templates, it computes which of the two clusters contains more good matches and choose it as the segment of the given food and returns it.

## 9.2 `Leftover` module

:

- `Leftover::matchLeftovers`:
  This function is used in order to bond a leftover dish with an original tray dish. This happens by coupling 4 methodologies and this function works as an handler of the class Leftover, which has the responsibility to call all the sub-functions in order to collect the vector of Couple (original-leftover) "predicted" by every methodology. It then gives these vectors to the `jointPredictions()` module which outputs the final result vector of Couple structs. Finally, it creates a vector of k-means segmented couples, stored using a vector of `SegmentCouple` struct, which is the output needed in order to use the metrics on the obtained results.

- `Leftover::jointPredictions`:
  This function is responsible for combining and refining the matching results obtained from different matching methods. The goal is to determine the best matching pairs of original dishes and leftover dishes based on multiple criteria:

  - dish area;
  - average BGR color distance;
  - number of matches by applying SIFT;
  - CIELAB color distance with Delta E metric or simple norm.

  The function returns a `finalPairs` vector containing the best-matched pairs of original dishes and leftover dishes.

- `Leftover::createCouple`:
  This function has the responsibility to create the vector of segments that must be used in metrics module. It performs k-means segmentation by selecting a k equals to the number of dishes found in the original dish by the classification module. This then implies a k-means segmentation on both the original and the leftover dish.

- `Leftover::breadFinder`:
  This is the function that find bread into the dish. It is only responsible of drawing a bounding box around bread, if found, also in the leftover dish.

## 9.3 *metrics* module

:

- `parseGroundTruth`:
  This function takes as input the string of the filename to extract the ID of the food considered and the coordinates x, y, and the width and height of the groundtruth Bounding box provided; a `FoodData` struct is then created to store all these informations in the `Rect` variable of the struct; then the struct is pushed into the vector of `FoodData` structs;
  - input: by reference, a string;
  - output: the vector of `FoodData` structs;

- `getiou`:
  The function returns the IoU (Intersection over Union) parameter between the groundtruth and the predicted Bounding boxes;
  - input: by reference, takes the coordinates values of x, y, and the values of width and height of the two Bounding boxes;
  - output: a float;

- `getious`:
  This function returns the Intersection over Union for each ground truth and associated predicted coordinates of the `Rect` variable from each `FoodData` struct.
  - input: by reference, the vectors of `FoodData` structs.
  - output: a vector of float;

- `getConfusionVector`:
  This function, based on the result of the IoU computed by the previous functions, returns a string, representing the correctness of prediction, that is pushed in a vector of strings. If the value of IoU is 0, the sample is taken as "false negative" ("FN"); If the value of IoU is less or equal to the IoUThreshold value, setted to 0.5, the sample is taken as "false positive" ("FP"); Otherwise, the sample is taken as "true positive" ("TP");
  - input: a vector of float;
  - output: a vector of strings;

- `getCumulativeTP`:
  This function increments the number of "true positive" predictions into the array of cumulative "true positive" predictions;
  - input: by reference, a vector of strings;
  - output: a vector of integers;

- `getCumulativeFP`:
  This function increments the number of "false positive" predictions into the array of cumulative "false positive" predictions;
  - input: by reference, a vector of strings;
  - output: a vector of integers;

- `getPrecision`:
  This function computes the precision value by dividing the cumulativeTP over the sum between cumulativeTP and cumulativeFP; then, the precision value is stored in a vector of float, that stands for all the precision values;
  - input: two vector of integers;
  - output: a vector of float;

- `getRecall`:
  This function computes the recall value by dividing the cumulativeTP over the sum between cumulativeTP and cumulativeFN; then, the precision value is stored in a vector of float, that stands for all the recall values;
  - input: two vector of integers;
  - output: a vector of float;

- `getAP`:
  This function calculates the average precision taking the values of precision of a given class and the number of times that specific class appears; so then, it returns the precision of a single class;
  - input: by reference, two vector of floats;
  - output: a float;

- `getmAP`:
  This function computes the mean average precision taking the values of precision of a single class and then computes the average between all of them;
  - input: by reference, a vector of float;
  - output: a float;

# 10 Undertaken and abandoned paths

We have addressed various avenues for food segmentation, in order to find out the best one to solve our problem.
At the very beginning we were working on single regions produced by `MSER` algorithm for Blob detection combined with the average color of the blobs. Some of this code is still present in the ImageProcessor class of the delivered final code, but is now unused since it produced tons of small rectangles which needed to be analysed and classified as one of the 13 food classes. It was inefficient (beans produced hundreds of blobs, for example) and could have been tricky to work on it. We have left in the code because we thought it could be useful at least during the segmentation part but at the end we created a function - computeBox() - that performs the computation of the bounding boxes in a better way.
For what concerns segmentation we have also tried to use `Watershed` and `Mean Shift`, since K-Means needed a K to be picked.
But mean shift algorithm for segmentation is not included in OpenCV library and if the algorithm is already non efficient, our version was certainly not going to be state-of-the-art, so we discarded this option and tried watershed but results were not as good as K-Means by picking the right K.
At the end we picked K-Means because we had the intuition of using the number of found ids inside of a dish as K parameter.
Mean Shift is not used for segmentation but `Pyramidal Mean Shift Filtering` - present in OpenCV library - is still used as preprocessing step before K-Means segmentation.
Another abandoned path is the one concerning `GrabCut()` function for foreground (food) segmentation and extraction. At the end we decided to implement from scratch a similar function that only works with simple thresholding of whites (the dish) and we called it `removeDish()` as said before.

# 11 Lesson Learned and problems encountered

This project thought us many lessons. First of all, how having different environments, versions and operative systems can be time consuming when merging all the code produced. A working code running on a Windows machine with 4.6.0 version and using Visual Studio, has different results if run on a Ubuntu 18.04 machine with Visual Studio Code and 4.7.0 version of OpenCV.
This implies necessity of solving these conflicts and also hours of lost time.
Furthermore, we have acquired a deeper knowledge of the use of the techniques studied in class. Also, we have wasted a lot of time implementing ineffective solutions.
Have been found also some problems to pull and push the code using Github Desktop.
We learned how, at the very beginning, it can be crucial to set a common architecture for the structures and the objects to be used and passed in different steps.
For the first period we only gave each others a common structure of what was needed in order to obtain the results, but there was not a clear architecture of the data to be used and created inside of the code.
It is also true that some ideas only come up to mind while

working, so it is hard to give a very good architecture just in the planning stage.
By making changes to improve the classification part, the accuracy percentages in leftover matching got worse as they weren't before.
To assign only a leftover dish for an original one, where there are more dishes in the leftover part, only average color metric is used but it is not so accurate as for the other metrics used.

We started solving this problem only when we decided to use this common objects:


FIGURE 1: Dish class and FoodData structure

# 12 Future developments

As main improvement for the future we would like to use better segmentation techniques. K-Means is very fast and, used together with pyrMeanShiftFiltering has been giving good results in many cases. But for what concerns complex structures and textures it has been performing not always so well.
In particular, as future developement we want to test the metrics module on the results coming from the classification and then leftover matching part.

## 12.1 Metrics part

The metrics part is not over but, for the first metric the behaviour has to be like this: having the two Bounding boxes (ground truth and predicted one) for each food on each tray, the mean Intersection over Union is calculated as the number of average precision for each class divided by the number of all classes.
For the second metric the function has to segment the food of each dish and has to be compared with the corresponding segment on the ground truth mask provided; in reference of the IoU threshold will return if the food is computed as "true positive", "false positive" or "true negative"; then it computes the average precision for that specific segmented class and finally it computes the mean average precision between all classes.
The last performance measure compute how good is the algorithm estimate of leftover food, computed as difference between the ratio of food leftover estimated and the ratio of the food leftover computed with ground truth data, for every food. The ratio is obtained dividing all the pixels of a food in the leftover food by all the pixels of the same food in the original image. To achieve this, we use the masks of the segmentation, and counting for the non-black pixels.
Thanks to the array of Dish objects, and the Segment Couple array, created in the previous parts of the program, this is not difficult since we have for every existing food the two segmented area. So, given a food, for each segmented area, we scan all the pixels and count the white ones. Then computing the ratio and the difference is trivial.
This computation is not too difficult to code, but it needs a lot of previous work: good segmentation and perfect matching of foods between original tray and leftover one.
We expect to have a not very good performance, since the segmentation is not accurate for every food, and there is a non negligible error percentage of matching foods in the leftover tray.

# 13 Results

The classification has some troubles to find rabbit on dish as for the seafood salad, and, in another tray confuses potatoes with pasta; even though, the percentage of error is acceptable.
In this section are reported, respectively, all the results of classification, few results of segmentation and few results of pair segments; for each part, the code provides the results for each tray; all the images are provided at the end of the report.

# 14 Conclusions

The task achievement was pretty hard to code, both on an high level, as algorithm design, both in the coding, and we met a lots of issues.
We try to figured it out without using machine learning techniques and our approach seems to work. The delivered project works on the detection, classification and segmentation of each food for each considered tray.
An improvement of the right matches of the leftover food classification and a better segmentation algorithm could satisfy the request and give also pretty good performances.
Even tough metrics part of the code, as long as hasn't be merged into the project, has not been tested on the project, so we cannot know how it is the true performance of our solution. In conclusion, it was challenging to try to solve this problem, and surely it helpful to have a better knowledge of the subject, its possible applications and also a better knowledge of the OpenCV library.

# 15 Hours of work and contributes

- Nicola Calzone: $\sim$ 150h, $\sim$ 1000 lines of code
  - Leftover matching
  - Image processing class
  - K-Means segmentation
  - Reordering and merging all the code
  - Report
- Riccardo Vendramin: $\sim$ 150h, $\sim$ 1000 lines of code
  - Dish detection and dish processing
  - Food templates
  - Food classification, bounding and segmentation
  - Image processing class
  - Test on Watershed, Mean Shift and MSER

  - Report

- Andrea Labelli: $\sim$ 50h, $\sim$ 300 lines of code
  - Metrics
  - Report

# 16 Resulting Images

FIGURE 2: Tray1

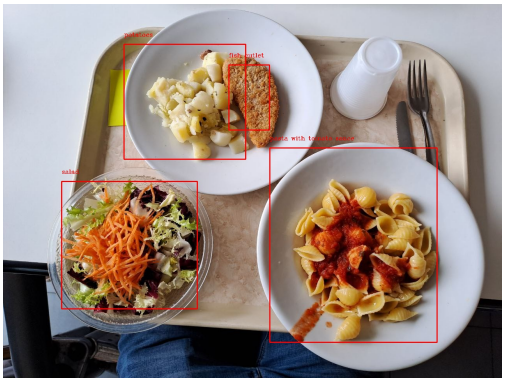


FIGURE 3: Tray2

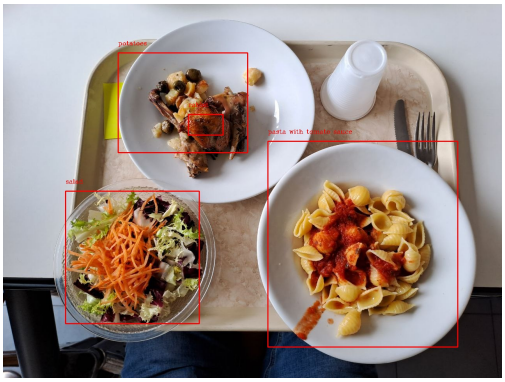


FIGURE 4: Tray3

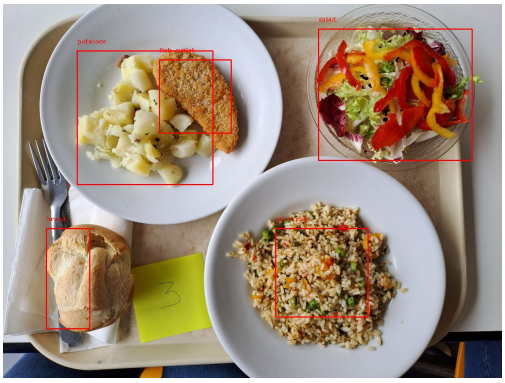



FIGURE 5: Tray4


FIGURE 6: Tray5

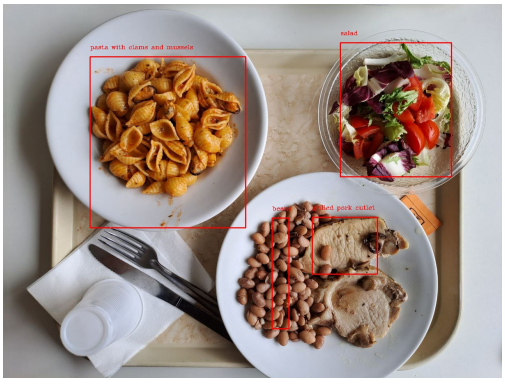



FIGURE 7: Tray6

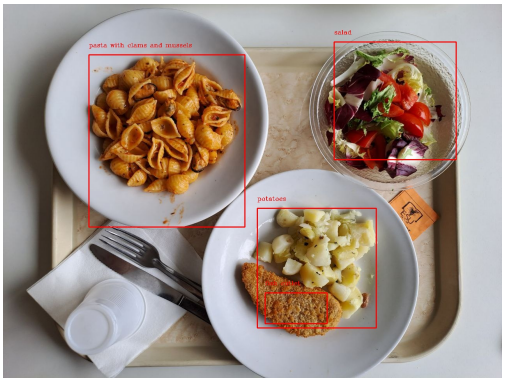



FIGURE 8: Tray7

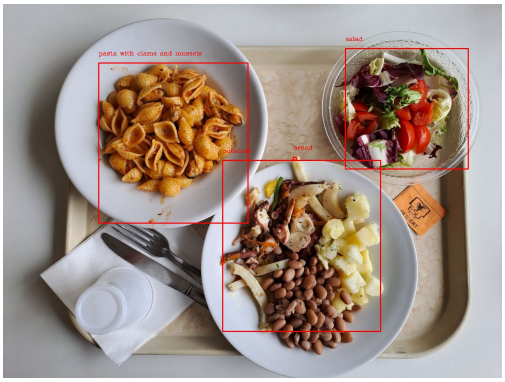



FIGURE 9: Tray8

FIGURE 10: Segment of grilled pork cutlet - Tray1

FIGURE 11: Segment of beans - Tray1

FIGURE 12: Segment of pasta with pesto - Tray1

FIGURE 13: Segment of salad - Tray2

FIGURE 14: Segment of fish cutlet - Tray2

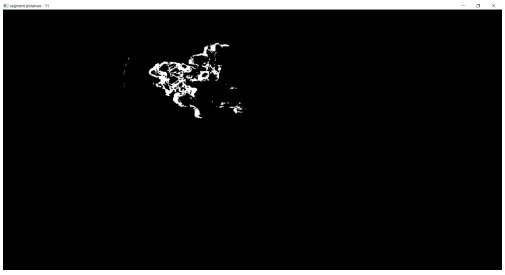


FIGURE 15: Segment of potatoes - Tray2

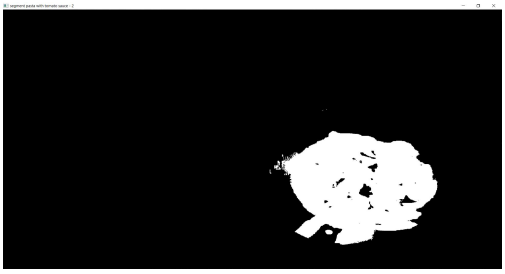


FIGURE 16: Segment of pasta with tomato sauce - Tray2

FIGURE 17: Segment of bonded original and leftover for ID 10 - Tray 1

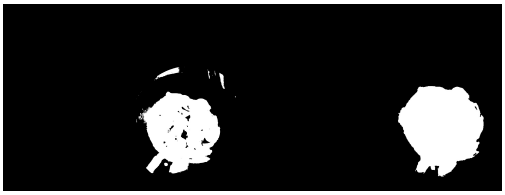


FIGURE 18: Segment of bonded original and leftover for ID 2 - Tray 2

FIGURE 19: Segment of bonded original and leftover for ID 12 - Tray 3

FIGURE 20: Segment of bonded original and leftover for ID 5 - Tray 4

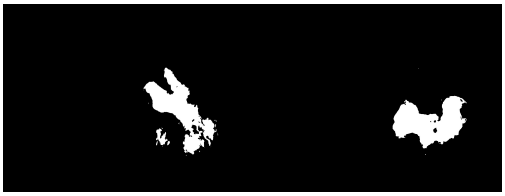


FIGURE 21: Segment of bonded original and leftover for ID 10 - Tray 4

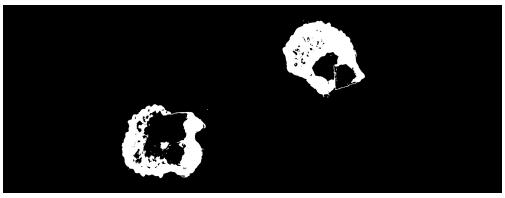


FIGURE 22: Segment of bonded original and leftover for ID 10 - Tray 4

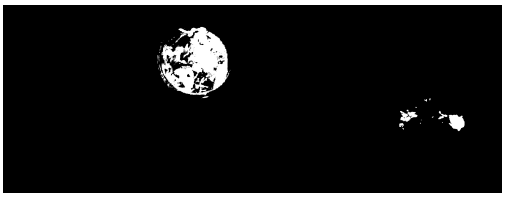


FIGURE 23: Segment of bonded original and leftover for ID 10 - Tray 4

FIGURE 24: Segment of bonded original and leftover for ID 11 - Tray 4

# Bibliography

[Schuessler, 2022] Schuessler, Z. (2022). Deltae: Cie color difference formulas in javascript.