DietVision : An App for Image-based Food Identification, Volume, and Nutrition Estimation

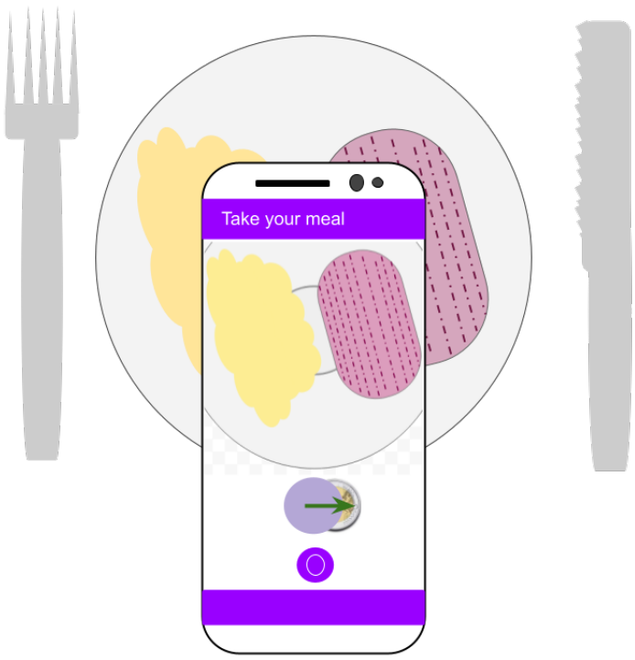
Michael Hofmann
MSc student
Télécom Strasbourg
misoko.hofmann@gmail.com

Léopold Maillard
MSc student
INSA Rouen
leopold.ma@hotmail.fr

Jessica Ramaux
MSc student
Télécom Strasbourg
jessica.ramaux13@gmail.com

Dennis Shasha
Professor
New York University
shasha@cims.nyu.edu

New York University Computer Science Technical Report 998
August 2021



Keywords— Food volume estimation, image segmentation, deep learning, stereo vision, fiducial marker, user feedback, Flutter app.

Contents

1 Abstract

DietVision is a mobile app that provides an estimate of the nutritional content of a meal from images. The software provides the following functions: (i) food detection that performs classification and assigns it to a major food group; (ii) volume estimation using two images at different angles of a plate along with a coin as a fiducial marker; and (iii) user feedback to correct errors in steps (i) and (ii). The app also provides features such as a detailed meal history, dietary habits statistics and personalized user preferences. It works on various smartphones using the Flutter Framework.

2 Example of Use

After launching the app, the user traverses menu pages until asked to take two pictures of the presented food along with a fiducial marker such as a two Euro coin. (The marker can be changed in the preferences tab.)

2.1 Taking the First Picture

The first picture is a top-view one and is used to detect food as well as to compute the surface it occupies, as on Figure 9. The fiducial marker is used as a reference object to get the right scale. Specifically, a coin is used since it has a fixed size which eliminates the issue of potential variations in size that other reference objects have, such as a plate or a glass.

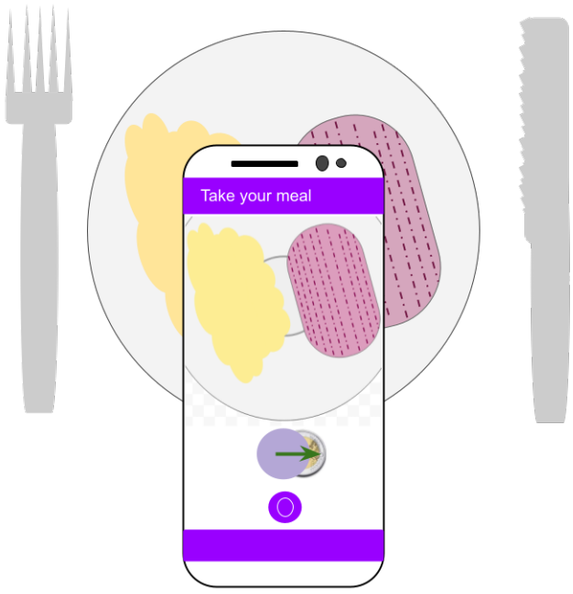
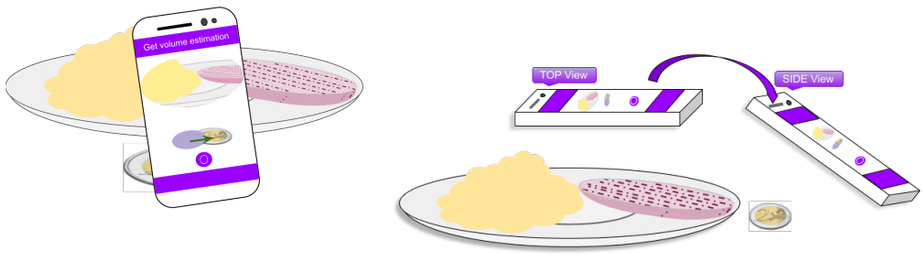


Figure 1: Top view of a meal to get the surface and the coin-food distance estimation for each detected class.

The coin is placed right in front of the dish (between the user and the dish) and the smartphone must be directly above it, in order to get the most correct top-view. The user has to make the coin match as closely as possible with the purple circle displayed at the bottom of the camera preview screen. This may seem laborious at first glance, but is in practice a fun challenge. Once the phone position is adjusted, the user takes the picture. After a few seconds, a list of detected food items is displayed along with their associated surface in square centimeters.

2.2 Taking the Second Picture

The purpose of the second picture is to estimate the volume of each food class detected in the first picture, as on Figure 2. The thickness of each food item must be obtained and to do so, the picture must be taken at a 45° angle. The coin should not be moved. This time, the user will match the coin with a purple ellipse (that corresponds to a circle viewed at an angle of 45°).



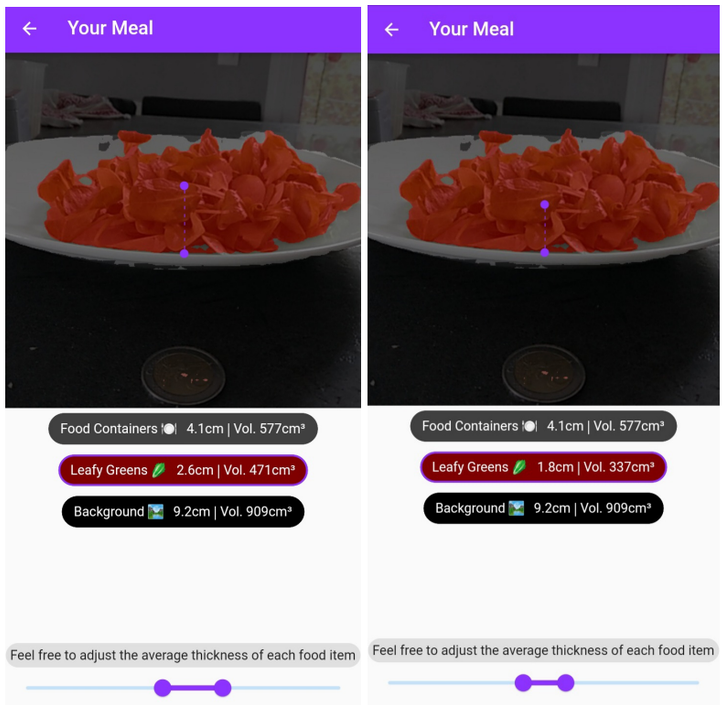
(a) 45 degree picture

(b) Example of taking a second picture with a 45 degree angle

Figure 2: The user may change the position of the phone between the first/second picture

After this second picture has been taken, the food items list will be displayed again, with this time their volume estimation in cubic centimeters.

2.3 Using Feedback to improve accuracy



(a) Original picture of a side view salad (b) Best accuracy on a side view salad

Figure 3: User's feedback to improve the accuracy of the thickness detection.

When the volume estimation results are displayed, the user is given the possibility to adjust them. Two categories of feedback can thus be performed to manually improve the final results :

- Food item classification : in the event that the machine learning model misclassifies a food item, the user is able to adjust the selection by tapping on it and selecting the appropriate class. A food class can also be deleted from the selection. Note that there is currently no option to manually add a missing item.
- Food item thickness : the estimated thickness of each food item can be visually adjusted thanks to a slider which moves on the picture the points that correspond to the 'minimum' and the 'maximum' position of the thickness. Note that we aim here at estimating the **mean** thickness of a food. If an item does not have a consistent thickness (e.g. a mashed potatoes portion), the system will ask the user to determine a mean value that will lead to an accurate volume estimation.

As we can see on Figure 3, the thickness is overestimated and so the volume will be incorrect. In this example, there are 21 grams in the plate. On Figure 3(a) a volume of 471 cm^3 corresponds to 28.26 grams of salad according to our database. However, with the user's feedback, we have a new volume of 337 cm^3 on the Figure 3(b) which corresponds to 20.8 grams. We can see that a little adjustment results in a difference of 5 grams. This feedback improves the accuracy of the app.

2.4 Meal nutritional values assessment

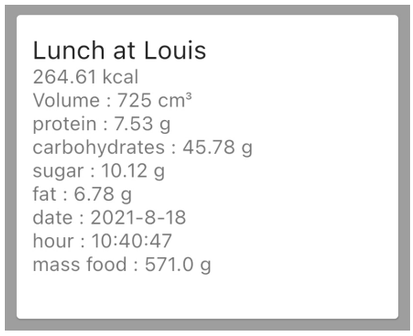


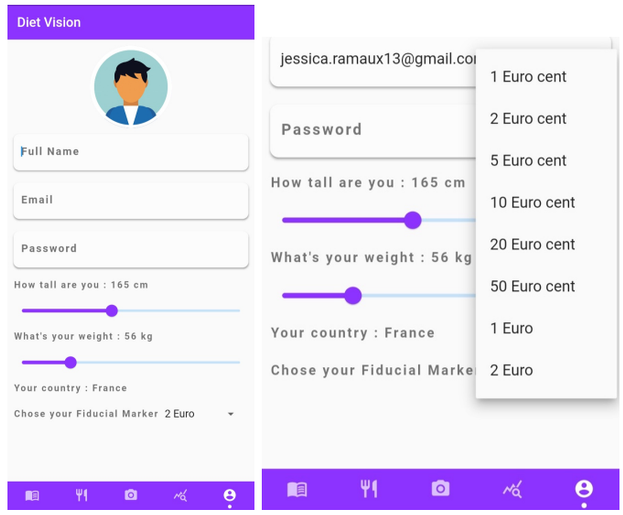
Figure 4: Saved meal entry

2.5 User's profile

To customize the experience, the user can add useful information to track the user's eating habits : name, size, weight, etc. These data can be filled on the *Profile* tab, as we can see on Figure 16(a). In addition, the country in which the user is located is displayed as well as a collection of coins 16(b) related to its currency.

2.6 Beverage in a meal

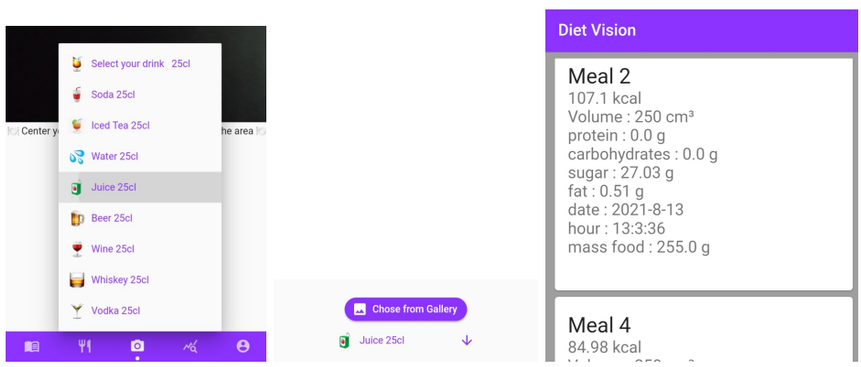
In order to take drinks into account, we added a dropdown button with a predefined list of drinks (Figure 6(a)). By selecting a drink, the user gets the nutritional information of the meal (Figure 6(c)) plus the values associated with the drink, as on Figure 6(b). To calculate these nutritional values, we consider a fixed volume (25 cl) which corresponds to an average glass of liquids. In an upcoming iteration of the app, the user should be able to adjust the drink volume.



(a) Overview of the user's profile.

(b) List of coins of a local currency (Here in France, the currency is euro)

Figure 5: User's preferences tab.



(a) List of Drink with a 25 cl volume

(b) Selected drink

(c) List of Meal in History screen for a beverage of 25 cl

Figure 6: Example of the drink button display and the drink history.

3 Architecture

3.1 Overview

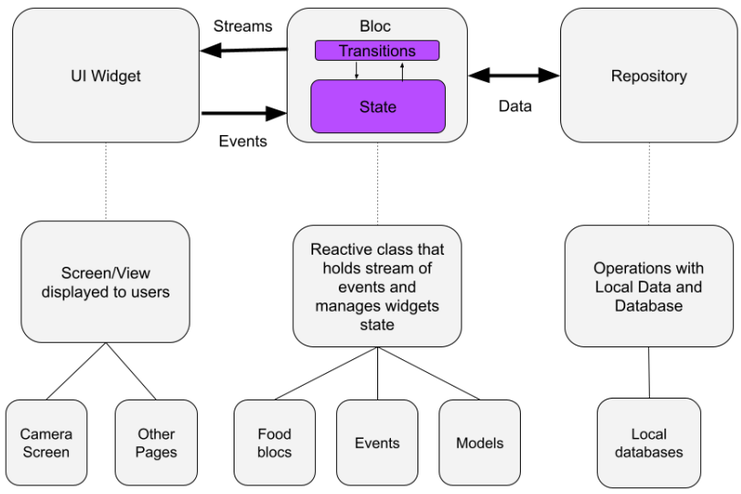


Figure 7: Flutter Flow Chart of the App.

GitHub repository : `https://github.com/leopoldmaillard/diet-vision-flutter`

The app was organized following the architectural pattern shown on the above diagram. Each of its components can be described as follow :

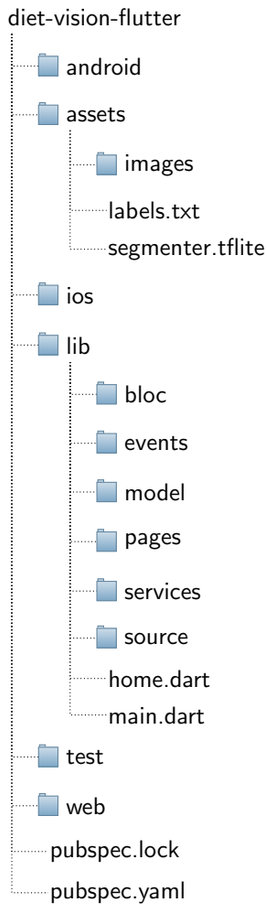
1. **UI Widget :** All the possible UI screens or "views" of the user interface will be handled in this part and be provided by the bloc.

2. **BLoC :** Deals with events every time the user interacts with the app, responds by computing the associated task and finally refreshes the updated screen. Its main goal is to keep the meal history database updated when a new one just has been recorded, as well as the tab where it is displayed.

3. **Repository :** Handles local databases used in the app and their storage.

3.2 Project file structure

A Flutter project has the ability to generate a cross-platform app from a single codebase. Our file structure reflects this implementation choice, its relevant components are described below and some of them will be detailed in the following parts of this report :



3.3 Flutter Layered Architecture

Focusing on developing a scalable app, we decided to structure the project using a layered architecture, as described below :

- bloc : Deals with the Stream of events and States.
- events : A collection of events corresponding to the different user interactions.
- model : Data Structure stored in the database that corresponds to a single meal.
- pages : Regroups all UI screens/views of the App.
- services : Database definition and manipulation.
- sources : Storage repository (mainly JSON files for drinks, nutritional values etc.).

The *main.dart* file is the starting point for the app execution and will invoke the other components.

3.4 Configuration and Flutter packages requirement

The Dart programming language offers a great variety of packages and some of them are extensively used in the DietVision app. An exhaustive list of these dependencies can be found within the project's *pubspec.yaml* file and includes, in particular :

- TensorFlow Lite plugin : A plugin for accessing TensorFlow Lite API that supports lightweight image segmentation models.
- Flutter Camera Plugin : A plugin for accessing the device cameras, getting a real-time camera preview and taking pictures.

- Geolocator plugin : A plugin for accessing user's location. It is used here to pre-select user preferences based on the current location.
- Shared Preferences : A plugin to save key-value data on disk in order to persist user preferences.
- SQFLite : A plugin for a relational database management system embedded into the end program.

When running the app for the first time, the user will have to grant it camera and location access. Optionally, image gallery access should be granted to use the app on pictures that are already stored on the device. The project's *ios/Runner/Info.plist* file has been modified to reflect these requirements.

3.5 Local databases : nutritional values and fiducial markers

During the development of this project, various databases have been created to communicate with the app during runtime. They are stored in JSON files, in particular :

- A food nutrition database

Each food class that can be identified by the app has an associated entry in the database, here is an example for the "Poultry" class :

```
  {
    "id": '7',
    "name": "Protein | Poultry",
    "cal": 271,
    "fat": 13.95,
    "protein": 19.22,
    "carbohydrates": 17.25,
    "sugar": 0.4,
    "glucide": "N/A",
    "vm": 0.90
  },
```

A food group has associated nutritional values such as its kilo-calories (Calories), carbohydrate, lipid, and protein per 100g. It also has a density value (g/ml) that is required to compute nutritional values of a given meal after its volume estimation. To get these values, we've simply averaged the nutritional values of a few food items in each group.[6]. More precisely, we used the table of [4] which contains more than 14 000 food meals. For each entry, we get all the nutritional information for 100g, along with the name of the food category such as "Meats", "Vegetables" or "Sweets" which can match the output categories of our segmentation model. We first tested our nutritional database with an average of a few ingredients per category to obtain a single value for each category.

- A drink nutrition database

Similarly, nutritional values have been gathered for liquids. Here is a sample of the resulting database.

```
{"name": "Coffee", "cal": "1", "mv": "1"},
{"name": "Coke", "cal": "42", "mv": "1.02"},
{"name": "IcedTea", "cal": "27", "mv": "1.03"},
{"name": "Water", "cal": "0", "mv": "1"},
{"name": "Beer", "cal": "43", "mv": "1"},
{"name": "Wine", "cal": "83", "mv": "0.99"},
```

- Access to a coin used in a country derived from its geolocation

When the user connects to the application, he/she can choose to allow geolocation to obtain his/her position, and therefore the country he/she is in. In the database we stored the name, the code of the Country which follow the ISO 3166-1 alpha-2 norm, and the money used in the country. We made this list of country for the American dollar, Canadian dollar, Euro and English pound. The list can be completed as needed.

```
{"id": '9', "coin": "euro", "country": "France", "code": "FR"},
{"id": '37', "coin": "american_dollar", "country": "United States",
    "code": "US"},},
```

- A fiducial marker database

To make sure the app is convenient and accessible to as many people as possible, the user is able to select the fiducial marker among an extensive collection of coins of the user's geographical area. Each coin has an entry that indicates its diameter and the currency to which it belongs.

```
{
    "id": '29',
    "coin": "euro",
    "value": "2 Euro",
    "diameter_inch": 1.01,
    "diameter_mm": 25.75,
},
```

- A History Meal database

To enable the user to track his or her food habits, a Meal History database records every meal the user has validated while using the app. These entries can be deleted by the user.

```
{
    int id;
    String nameFood;
    String nutriscore;
    int volEstim;
    double volumicMass;
    double mass;
    double kal;
    double protein;
    double carbohydrates;
    double sugar;
    double fat ;
},
```

4 Algorithms for food detection, classification and volume estimation

To estimate volume from pictures in a few seconds on a smartphone, our system computes the area of each food item from a top-level picture and then the thickness (and therefore the volume) from an angled picture. The volume is calculated from the simple formula :

$$Volume = Surface \times Thickness \tag{1}$$

4.1 Food segmentation

The first key component of the DietVision app is its semantic food segmentation model, whose goal is to separate the different food items on a plate by classifying each pixel in one of 26 major food groups.

The integrated model is the Seefood Mobile Segmenter that has been released by a Google Team. It adopts a DeepLab-V3 architecture[1] with a MobileNet-V2 backbone[5] that offers state-of-the-art performance on mobile semantic segmentation. It takes as input a 3-channel RGB image of size 513×513 scaled to $[0-1]$ and returns the associated 513×513 segmentation mask where each pixel takes a class index of the following labelmap :

```
id,name
0,background
1,vegetables | leafy_greens
2,vegetables | stem_vegetables
3,vegetables | non-starchy_roots
4,vegetables | other
5,fruits
6,protein | meat
7,protein | poultry
8,protein | seafood
9,protein | eggs
10,protein | beans/nuts
11,starches/grains | baked_goods
12,starches/grains | rice/grains/cereals
13,starches/grains | noodles/pasta
14,starches/grains | starchy_vegetables
15,starches/grains | other
16,soups/stews
17,herbs/spices
18,dairy
19,snacks
20,sweets/desserts
21,beverages
22,fats/oils/sauces
23,food_containers
24,dining_tools
25,other_food
```

The model's authors specify that it has been trained on a Google-internal food ingredient parsing dataset and that the labelmap was constructed based on USDA guidelines and lists of popular dishes on Google Lens.

The TensorFlow Lite version of this model is used and its .tflite file can be found in the project's *assets/* directory along with the *labelmap.txt* file. It can be directly downloaded from a dedicated page on the TensorFlow Hub repository[2].

4.2 Surface estimation

The segmentation model yields an estimate of the surface area of each food type.

4.2.1 Retrieve segmentation results

In Dart, the output of the segmenter is a raw list of ARGB pixels that corresponds to the segmentation mask. Each class of the labelmap has an associated ARGB value that must be defined in a data structure mapping a class name to its custom colour.

Then, by iterating on each pixel value of the output, the system derives the identified food items of the input picture as well as their associated number of pixels.

4.2.2 Fiducial marker

After obtaining the number of pixels per food group, the system computes the actual area thanks to the fiducial marker. The calculation is simple:

$$class\_surface(cm^2) = \frac{class\_pixels \times coin\_surface}{coin\_pixels} \tag{2}$$

The output is a data structure mapping each represented food class with its surface.

4.2.3 Food classes distance

The top-view picture allows us to know precisely the relative position of each food in the plate and its distance to the coin.

Algorithm 1: Get the distance between the coin and each food item

Input : List of all pixels (pixelsList) of the detected class
Output: Distance (in cm) between the middle of the coin and the lowest pixel detected for the class.

$length \leftarrow$ length of the pixels list of the selected class;
1) Non-empty list verification
if $length == null$ **then**
  return 0;
end
2) Get useful values to make the calculation
$listY \leftarrow$ new list of pixels of the selected class;
3) Complete a list of pixels with row index
for $i \leftarrow 0$ **to** $length$ **do**
  $listY \leftarrow$ add index row of the pixel;
end
$xMax \leftarrow$ maximum of the list listY which correspond to the lowest segmented pixel of the selected class;
$distancePixels \leftarrow (513 - 513/16 - xMax)$: distance (in pixels) between the middle of the coin and the lowest pixel of the class;
$distanceCoinFood \leftarrow$ value of distancePixels converted (in cm);
return (distanceCoinFood);

We use a simple cross product to obtain the distance (in cm) from a distance (in pixels):

$$class\_distance(cm) = \frac{class\_distance\_pixels \times coin\_diameter}{coin\_diameter\_pixels} \tag{3}$$

This distance value will be useful when estimating the volume using a second picture (see section 4.3.3).

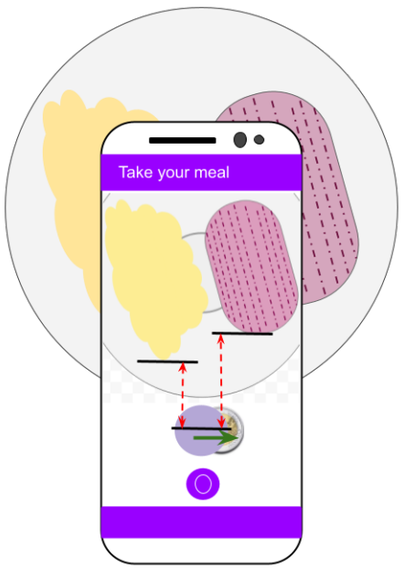


Figure 8: Measurement of distance between each food item and the fiducial coin.

4.3 Volume estimation

Once the surface of each food item is computed, the next step is to get each item's average thickness. One obvious option would be to take a second picture with a 90° angle with the fiducial marker placed vertically.

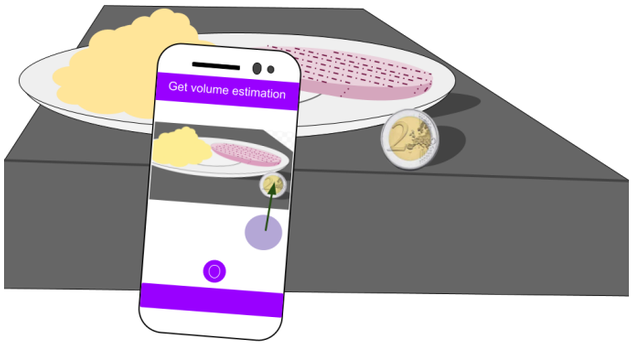


Figure 9: Graph with 90°

However, this is inconvenient because the user would have to hold the coin and take a picture while matching a 90° angle which does not feel natural at all. In addition, the food segmentation model doesn't work well on pictures of food with such extreme angles so accuracy was poor.

Instead, the system asks the user to take a picture with the coin on the able at a 45° angle from the table plane. The main challenge for the user is to ensure that the angle is correct.

Figure 10: 45 degree point of view picture

4.3.1 Fiducial marker with a known angle

When we take a picture with an angle between the camera and the object of interest, rounded items like a coin will have an elliptical shape instead of a circular one. Moreover, items in the foreground will appear to be bigger than those in the background etc.

To address these distortions, we considered the elliptical shape that corresponds to a coin viewed from a known angle of 45°. To match this area while taking the second picture, the user will be obliged to get this exact camera angle. Such an ellipse can be obtained from the following formula:

$$minor\_axis = (major\_axis) \times sin(\frac{\pi}{4}) \tag{4}$$

4.3.2 Thickness computing

Considering a picture with a known angle, we determined how to compute the thickness of the different food items. The major concern is that, in the second picture where we see the segmentation mask of a 45° point of view, the visible pixels represent both pixels from the surface of the first picture as well as "new" pixels that correspond to the thickness. The challenge was to isolate these item's thickness pixels from the pixels that have already been seen on the top-view picture.

Overview with a Graph

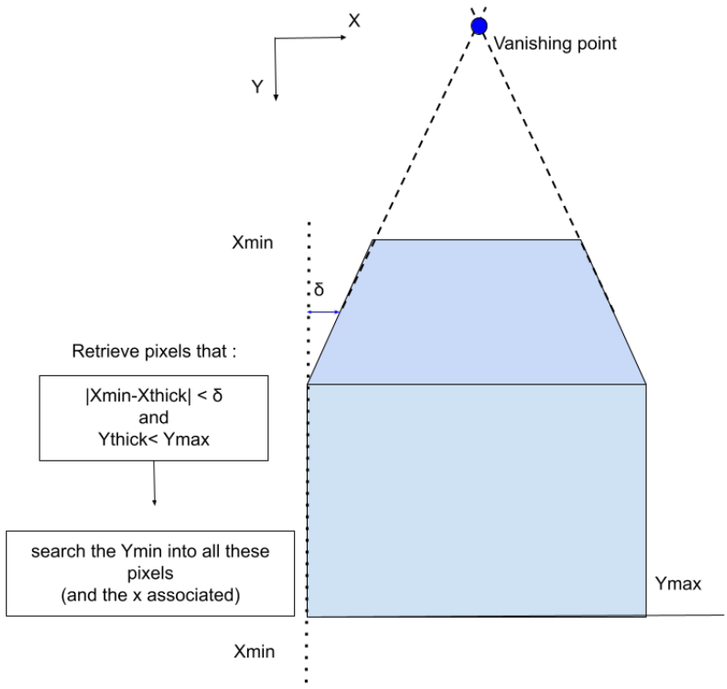


Figure 11: Thickness Estimation

To differentiate pixels that represent the thickness from those that represent the top surface, we consider the food as parallelepiped. When we take a picture of a three dimensional item, it has a *vanishing point* (that is commonly used by designers to indicate the depth of a parallelepiped). Finding the bottom point of the parallelepiped is easy since we have segmentation of the food.

However finding the top pixel of the food thickness is more of a challenge since the food on a plate is obviously not a perfect parallelepiped. By considering the food as a parallelepiped, we considered finding one of those pixels at the top left corner because this is where the depth edges and the height edges converge (we could have considered the top right corner as well).
Looking at it in another way, this point must be in the y-axis, upside the bottom on the lowest point, and not too far from the y coordinates of the left-most food pixel, δ at most. Otherwise, it is certainly a pixel of the top view picture.

At this point we have an estimation in pixels of the thickness. This algorithm is quite robust but is unfortunately not precise all the time. It also must take into account the scale, since an object far in the picture will look smaller than it actually is. These issues are discussed below.

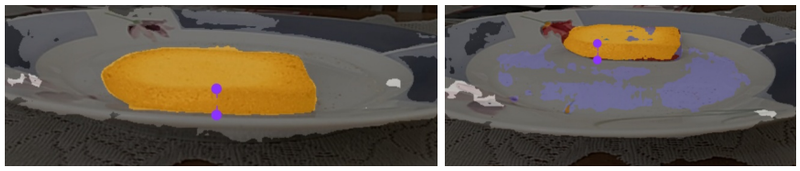
Algorithm 2: Algorithm to get the thickness precise estimation

Input : List of all pixels (pixelsList) recognized for each class, Name of the class (name).
Output: Thickness in pixel calculated for the chosen class, and register the points (Ymin and Ymax) chosen for this calculation.

$length \leftarrow$ length of list of pixel in the selected class;
1) Verification of non-empty list
if $length == null$ **then**
  return 0;
end
2) Obtain useful variable to make the calculation
$listX \leftarrow$ new list of pixels in the selected class;
$listY \leftarrow$ new list of pixels in the selected class;
$listPossiblePixels \leftarrow$ new list of possible pixels on top of the food bottom thickness in the selected class;
$limitDownPlate \leftarrow$ give the lower limit of the plate in the picture (above the ellipsoid shape for the coin);
3) Complete a list of pixels with index of column/row
for $i \leftarrow 0$ **to** $length$ 1 **do**
  $listX \leftarrow$ add index column of the pixel;
  $listY \leftarrow$ add index row of the pixel;
end
4) Algorithm of thickness
$YBottomThickness \leftarrow$ the lowest pixel in the plate for the chosen class;
$Xmin \leftarrow$ the minmium column of the list of pixel in the plate i.e the pixel the most at left for this selected class on the picture;
4.1) Find the best pixel for the top of the item food
for $i \leftarrow 0$ **to** $length$ 1 **do**
  $\delta \leftarrow$ distance between the column at the pixel chosen and the column of the leftmost pixel defined before: (listX[i]-Xmin).abs);
  if $(listY[i] < YBottomThickness)$ *and* $(delta < 5)$ **then**
    $listPossiblePixels \leftarrow$ add this pixel in the possible list of pixel for the y top of the thickness;
  end
end
5) Find the precise thickness
$length2 \leftarrow$ length of list of possible pixels for the top pixel of the thickness;
if $length2 \neq null$ **then**
  $YTopThickness \leftarrow$ the pixel corresponding to the top of the food to obtain an estimate of the thickness i.e the minimum *Dennis asks: should this be maximum?* of the list of possible pixels;
end
6) Create a list of the index (X/Y)
$minMaxList \leftarrow$keep min and max pixel index used for the thickness calculation;
return (YTopThickness−YBottomThickness);

4.3.3 Accurate & scaled thickness considering perspective

One of the challenges related to the parallax effect that appears on the side-view picture was the scale variation of a food item. If we take the same item placed in the background (respectively, in the foreground) of a scene, it will appear smaller (respectively, larger). Indeed, its thickness will represent fewer (respectively, more) pixels for a same item, as we can see on Figure 12(b) (respectively 12(a)).



(a) Bread in foreground (b) Bread in background

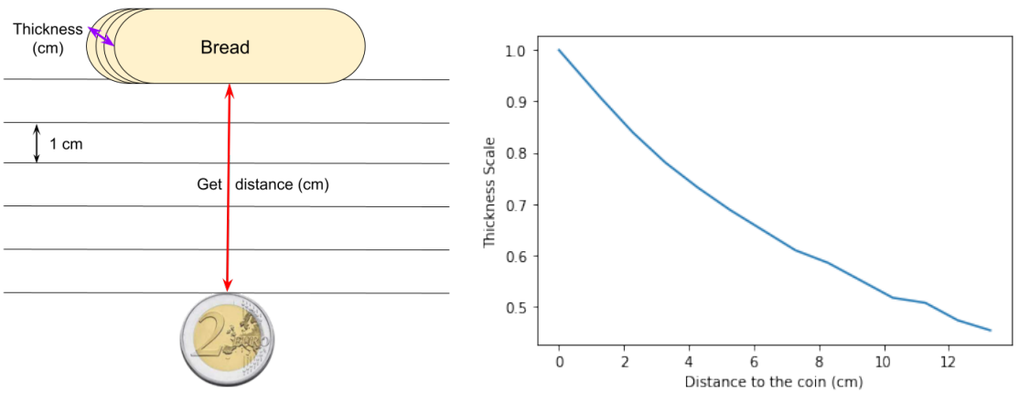
Figure 12: A given item will fill more pixels if it is in the foreground.

In order to solve this problem, we used an empirical approach to see how the the number of pixels of an item's thickness evolve with its distance to the camera. We did an experiment with an iPhone Xr : the same food item (3 slices of break stacked) was taken with a 45° angle and at several known distances to the fiducial marker. Each time, we took note of the number of pixels that corresponded to the thickness of the item in order to find out how it changes with distance. We were able to plot the thickness in pixels as a function of the food/coin distance (Figure 13(b)) and fitted a logarithmic curve to the resulting points. This results in the following function :

$$realThickness = thickness \times \frac{1}{1.54 - 0.39 \times log(0.94 \times distance + 4.00)} \tag{5}$$

Where *thickness* is the thickness (in pixels) estimated by the thickness algorithm detailed earlier (Algorithm2) and *distance* is the distance in centimeters between the coin and the detected food item. This function returns the corresponding thickness in pixels for a given distance between the coin and the food item on the plate.

In principle, this function could more or less depend on the camera's focal length, but we obtained similar results after further testing with other phones such as a Xiaomi Redmi Note 8 Pro and a Samsung A20s. The thickness calculation depends on the distance calculation between the coin and the different food classes described in section 4.2.3.



(a) Set-up to measure the number of pixels of the thickness at different distances.

(b) The resulting law.

Figure 13: How to create an empirical law to resolve the issue of back/fore-ground

4.3.4 User-friendly Feedback

As stated in the example of usage, the user is able to visually adjust the result of the thickness algorithm. Indeed, it has been determined that it would be hard to rely on a end-to-end computer vision solution that would be robust to all shapes of food items, while fast and easy-to-set feedback could significantly improve the results.

A range slider is used to make the adjustment, and the screen is dynamically updated to reflect the changes in real time.

Algorithm 3: Adjust the thickness value of a food item

Input : List of the coordinates of the minimum and maximum points of the thickness of all detected classes (minMax), the currently selected class (selectedClass), the new min & max values (newValues) from the Range Slider
Output: minMax list with updated coordinates
SIMPLY ASSIGN NEW SLIDER VALUES TO THE CORRESPONDING MINMAX INDICES
$minMax[selectedClass][0] \leftarrow$ new y-coordinate min value of the thickness;
$minMax[selectedClass][2] \leftarrow$ new y-coordinate max value of the thickness;
return (minMax);

Note that $minMax$ is a global variable, change in its values will be reflected to other variables such as the volume of the selected food item.

4.4 Dietary Assessment

As stated in part 3.5, a local database has been built and each food group has its detailed nutritional values (calories, fat, protein, etc.) given per 100g portion. Since we have a volume estimation, the major entry of the database is the **density** value of a food class, from which we will be able to get all the other nutritional values of a meal, for example :

$$portion\_calories = \frac{volume(cm^3) \times density(g/cm^3)}{100} \times food\_group\_calories(kcal/100g) \quad (6)$$

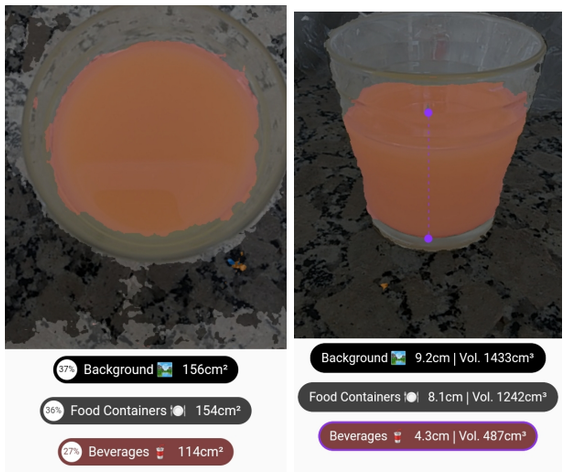
4.5 Known issues and potential improvements

4.5.1 Segmentation model

The food semantic segmentation model is at the heart of the app's mechanism. However, foods can be very diverse in terms of shapes, colors, textures, and the way they are cooked. Classifying them can thus be tricky and we believe that an improvement of the model (less segmentation artifacts, more detailed food classes) would enhance our results, especially when the angle of view is uncommon. We also noticed that segmentation results may vary depending on the light conditions. Because the model has been trained on a Google-internal dataset, it is not fine-tunable at the moment.

4.5.2 Beverage volume estimation

As stated in the Example of Use part (Beverage in a meal), we can add a beverage into a meal thanks to a list of drinks that the user can select. This system can give the nutrients precisely for a given volume (by default, 25 cl), but there is no computer vision based volume estimation built in the app at the moment, which means that all the drink information should be filled by the user. One improvement could be to allow the user to modify the volume of the drink according to what the user knows about the meal. Another option would be to use our pipeline on drinks. The segmentation model has a "Beverages" category that detects drinks that have a fairly high opacity such as orange juice or beer, however it does not work on drinks that are too clear, such as lemonade or water. A future iteration of the app could allow the user to choose the type of beverage and then perform volume estimation by taking two more pictures of the drink container.



(a) TOP View: Segmentation of an Orange juice in a glass

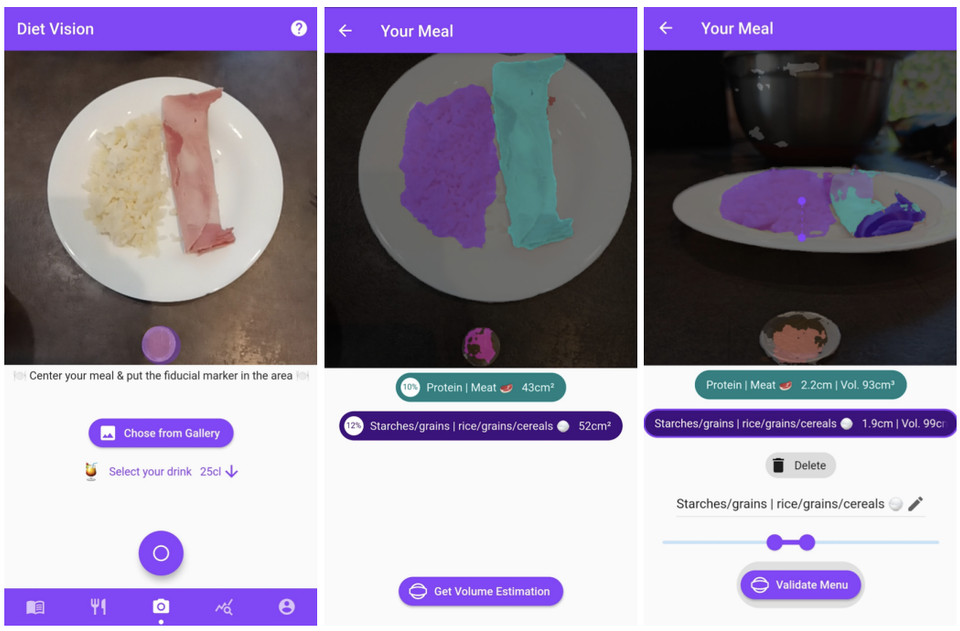
(b) SIDE View: Segmentation of an Orange juice in a glass

Figure 14: How to use the drink/beverage segmentation.

5 Results

DietVision has been tested on real meals in order to evaluate its accuracy. We present here a few of these results, but please note that the app should be applied on a wider variety of food in order to get a complete sense of its performance and limitations (results could be significantly more accurate en some types of food than others).

First meal : rice and a slice of ham



(a) Camera preview of the meal.

(b) Top-view surface results.

(c) 45° view volume results.

Figure 15: App screenshots for rice and ham.

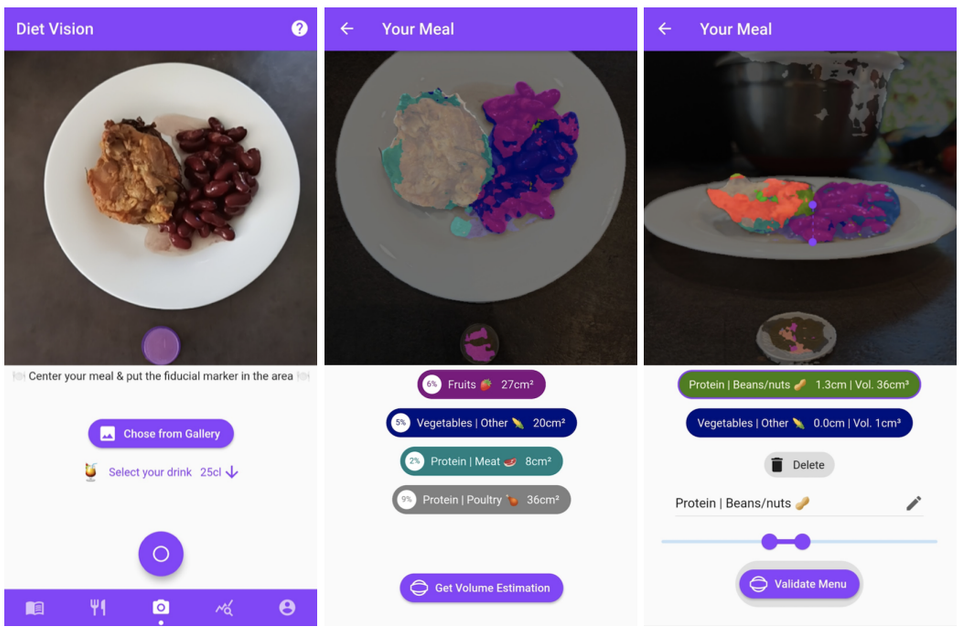
We can see that the segmentation result of the top-view picture is visually nearly perfect (Figure 15(b)). Thus, the surface estimation should be great as long as the fiducial coin has been precisely

matched with the purple area (Figure 15(a)). However, we can see that the segmentation of the 45° view isn't as clean, especially for the ham. Fortunately, the detected thickness can be adjusted. The nutritional values of this meal after it's been validated are given in the following table :

	Ground truth	DietVision estimation
Calories	97.2 kcal	153.46 kcal
Protein	7.79g	10.45g
Carbohydrates	14.20g	12.93g
Sugar	0.2g	0.02g
Fat	0.83g	6.2g

We can see that most of the estimated nutritional values are on the same range as the target ones. This is the case for proteins, carbohydrates, and the sugar value that is close to zero. However, the fat value is greatly overestimated by the app. This can be explained by the fact that DietVision has a single *Meat* class that is assigned to any kind of meat (beef, pork, veal, etc.) and thus has a single fat/100g value. However, some of the most commonly consumed meats have a high fat value (beef steak, pork chops), while our ham seems to have a particularly low fat value according to the ground truth. This trend also greatly affects the calories value. This could be adjusted by setting a fat value that fit better all the meat types or by adding the possibility to the user to refine the prediction by precising the type of meat in a future version of the app.

Second meal : chicken and red beans



(a) Camera preview of the meal. (b) Top-view surface results. (c) 45° view volume results.

Figure 16: App screenshots for chicken and red beans.

We can see that the segmentation was more challenging for this meal. Most of the piece of chicken has been correctly identified as *poultry* on the top picture, however, a large portion of the beans is labeled as *fruits* instead of *other vegetables* so the resulting surface is underestimated (Figure 16(b)). The second picture is worse, but again the thickness for each class can be adjusted to match what is actually on the plate (Figure 16(c)). The resulting nutritional values are given in the following table :

	Ground truth	DietVision estimation
Calories	203.13 kcal	241.34 kcal
Protein	26.11g	17.75g
Carbohydrates	9.88g	3.3g
Sugar	0.76g	0.78g
Fat	5.41g	17.72g

Once again, there is a certain consistency in the results : sugar value is low while protein one is high. Proteins and carbohydrates seem however significantly underestimated, while the fat value is again overestimated (chicken is a lean meat).

These results suggest that accuracy could be improved with a more robust segmentation model, adjusted nutritional values for each class, or the ability for the user to specify foods, while keeping the same pipeline.

6 Conclusion

DietVision is a cross-platform app that estimates nutritional values of a meal from images. Its approach relies on machine learning, stereo-vision and intuitive user feedback. It has been designed to be accessible to the largest number of users and to easily accommodate new features in the future.

7 Installation

We are targeting a release on the Apple App Store and the Google Play Store in order to ease the installation process and reach a larger audience. However, for now the app can be compiled and installed from the source code by following these few steps :

1. Follow the official Flutter Documentation to properly set up the Flutter SDK on your system.
2. Clone the DietVision's GitHub Repository.

   ```
   $ git clone https://github.com/leopoldmaillard/diet-vision-flutter.git
   ```

3. Navigate to the project's root directory and run

   ```
   $ flutter run --no-sound-null-safety --release
   ```

Note that Apple's Xcode is required to run the app on an iOS device as well as trusting the app in the device's settings.

Alternatively, Android users can directly download and install the app via the *Release* section of the GitHub repository.

Acknowledgement

We thank Éric Reynaud (math teacher, CPGE Alphonse Daudet) for his insights on solid geometry and parallax. We also thank Ikram Ziani (MSc student, Université de Montpellier) for helping on the thickness estimation algorithm. Finally, our warmest thanks to Shaun Maher (Columbia University) for his weekly guidance throughout the project.

References

[1] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[2] Google. Seefood mobile segmenter. `https://tfhub.dev/google/seefood/segmenter/mobile_food_segmenter_V1/1`.

[3] Frank Po Wen Lo, Yingnan Sun, Jianing Qiu, and Benny Lo. Image-based food classification and volume estimation for dietary assessment: A review. *IEEE Journal of Biomedical and Health Informatics*, 24(7):1926–1939, 2020.

[4] MyFoodData.com. MyFoodData Nutrition Facts SpreadSheet Release 1.4.

[5] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[6] David Haytowitz U. Ruth Charrondiere and Barbara Stadlmayr. FAO/INFOODS Density Database Version 2.0, 2012.