**Product Recognition on Store Shelves**

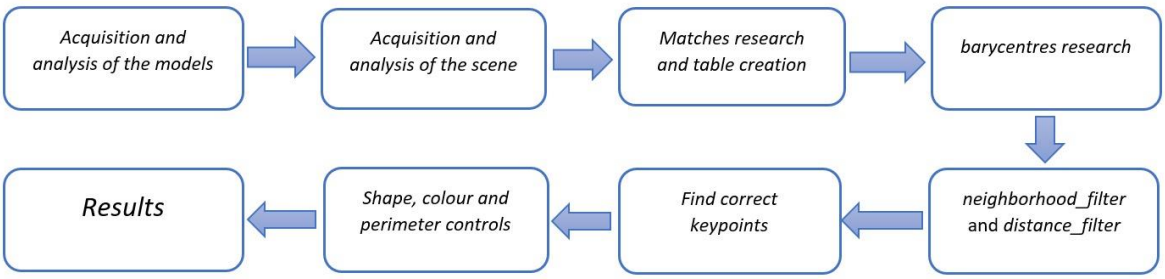
Overall Task:

Develop a computer vision system that, given a reference image for each product, is able to identify boxes of cereals of different brands from one picture of a store shelf. For each type of product displayed in the shelf the system should report:

1. Number of instances.
2. Dimension of each instance (width and height of the bounding box that enclose them in pixel).
3. Position in the image reference system of each instance (centre of the bounding box that enclose them in pixel).

Step A & B (Mandatory) - Multiple Product and Instance Detection:

The proposed problem presents as main issue the research of multiple instances of cereal boxes, having as a reference only one picture for each type of product, condition that prevents the use of algorithms based on multiple training images. For this reason our approach is based on a feature detection algorithm, that is the SIFT algorithm, which has been described as the most robust of its class w.r.t. rotation and viewpoint changes. Its main drawback, in fact, is related to computation time, not so relevant from a practical point of view in the execution of this project. However, this solution has not been applied in the standard way, mainly used for single object detection, but has been integrated with a table for barycentre research, in order to be suitable also for multiple instances. In the following we will explain step by step (as represented in the scheme below) all the stages performed to fulfil our purpose.




First of all, an offline phase is carried out to execute all those actions which do not require the scene analysis. This choice has been made because of the possible applications of this project. Indeed, for example, a computing device provided with a camera could have already in memory all info about adopted models and functions, and it would compute at run time just those data related to the captured scene, which was not available before. This is also why our code presents an input line where the scene of interest can be chosen, instead of using a sequence of *for* loops to gather all scenes at once.

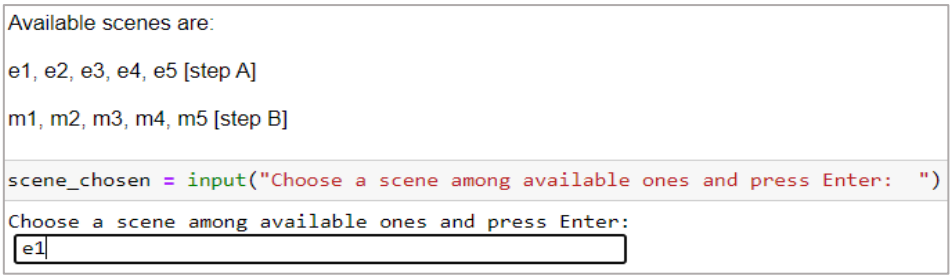
In this initial phase all model images are acquired from the *models* folder, both in the coloured and gray versions. In addition to this, after a conversion from BGR to RGB (due to OpenCV convention in image loading), the 3 axes of each model are extracted as single images.

Each model is then elaborated to get its reference colour, obtained by computing the mean of its channels in a given area. For our purposes we've chosen the lowest part of the model, neglecting the first third of the image. Cereal boxes, in fact, usually have common characteristics at the top (like the logo) and they differ below, especially in color terms.

Finally, also model keypoints are found and described, ready to be used in the next part of the code to get correspondences. This computation is done involving all channels, because a higher number of collected points of interest turns out to be better to perform barycentres research later.

An important note to be done is that, throughout the project, in order to keep track of all collected information (like those just seen), we deploy as main tool the list of lists, that can be accessed using indexes corresponding to images.

The following part is dedicated to the online phase. As already introduced, the input function shown below is exploited to let the user choose which scene to analyze.




Once the *scene_chosen* variable has been set, the series of steps also performed before (including image loading, channel extraction and keypoint research and description) is used to get all information to compare the scene with the templates.
Keypoints extracted from different channels are then considered together and merged in a unique structure, which is exploited to better organize possible couples of correspondent points got by the matcher.
In practice, these correspondences are established studying the descriptors coming from both the scene and the models, and then computing their Euclidian distance.

Also, considering that the set of points to analyse could be really large, the idea of computing all distances (according to the standard brute force matcher technique) was rejected, because it would have been too slow in terms of computation time. In its place it has been used a FLANN (Fast Library for Approximate Nearest Neighbors) based matcher, known to be more efficient in case of large datasets and for high dimensional features, together with an approximate KD-tree indexing technique, which allows to organize the descriptors in a way such that it is easier to match them.
This type of matcher is directly included in OpenCV and enables also to choose the number of closest elements we want to find for each model descriptor. This degree of freedom turns out to be fundamental in our multiple instance problem: indeed, it is possible that the same keypoint has multiple correct correspondences in the same scene, and, as a consequence, all of them should be detected.
In general, inside supermarkets no more than 2 boxes of the same type are placed on the same shelf, as it can be seen also by the sample images. According to this, the number of closest elements has been set to 3 to have a fair capability and avoid possible errors.

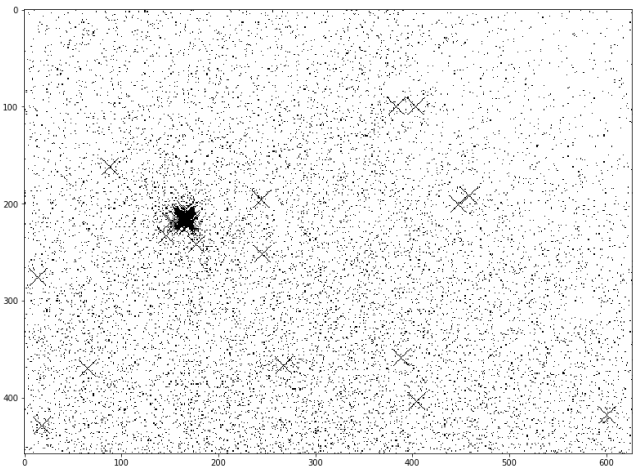
Once that all correspondences for each model descriptor have been found, they are placed in a structure of lists indexed with their reference sample cereal box number. In the same way also the coordinates and sizes of the matched keypoints are stored to be used later.

Completed all of these steps, it is evident that not all the matches found are good and, more important, not all of them refer to the same physical box. In order to solve this problem a barycentre-research approach has been implemented: an algorithm that relies on a modified version of the GHT (Generalized Hough Transform). More precisely, the standard r table has been substituted entirely with a list of *keypoint-vector* couples for every model image. The *vector* is nothing more than the distance between our keypoint and the known barycentre of the model, computed in coordinates as half of the box width and height. For this purpose, a suitable function has been created: *build_r_table(barycentre, src_vector)*.

After the construction of the described “table”, its lines, associated to single model features, can then be accessed by their matched counterparts in the scene (3 as mentioned before), through the function *accumulate_kp(r_table, Shelf_Image, dst_vector)*. In this way it is possible to cast votes for the barycentres positions: indeed, each joining vector found in the constructed list, if appropriately scaled according to the

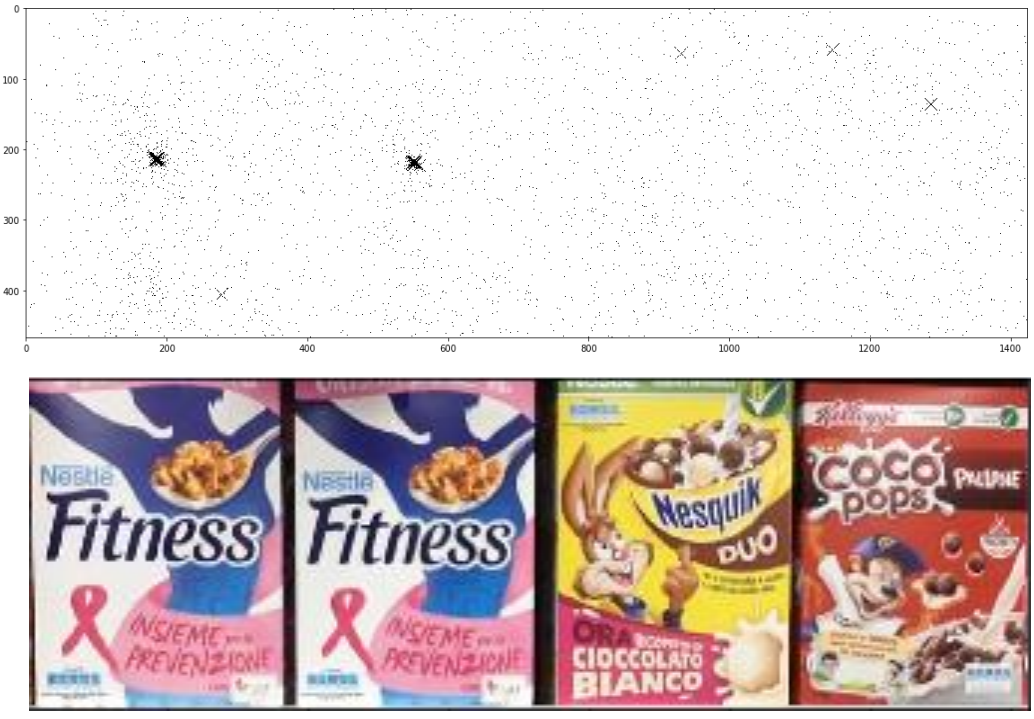
ratio between the old keypoint size and the new one, will give the coordinates of its associated centre. Of course, this won't be always a proper estimation, but just in the case of a correct matching between features. The return value of the function is a matrix called *accumulator*, whose shape is exactly the same of the currently analysed scene and whose elements contain the number of times that a specific point has been computed thanks to the table vectors. Its role is to give the possibility to keep track of all the votes, but also to enable the user to see them displayed on screen. The latter task is realized with a dedicated function, *plot_accumulator(accumulator, shelf_image)*, in which also a list of the most voted barycentres for each model is computed, in order to be used later.

As an example for the step A, below it is shown the accumulator associated with model 0 and scene e1, in which the box appears to be present (as it can be seen in the scene itself). All points which had more than 2 votes have been represented with a cross: as a consequence, just giving a first look to the image, it is evident that the Nesquik box will be placed in the left side of the image, centred with respect to its height.



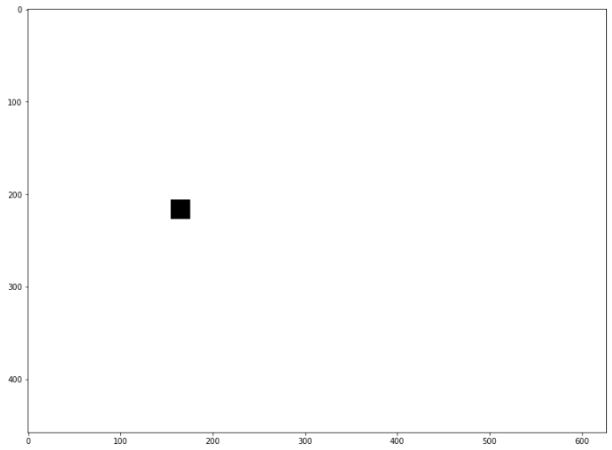


For step B we can consider model 24 (Fitness cereals) in scene m1: this shows that also multiple instances can be found with the same method.

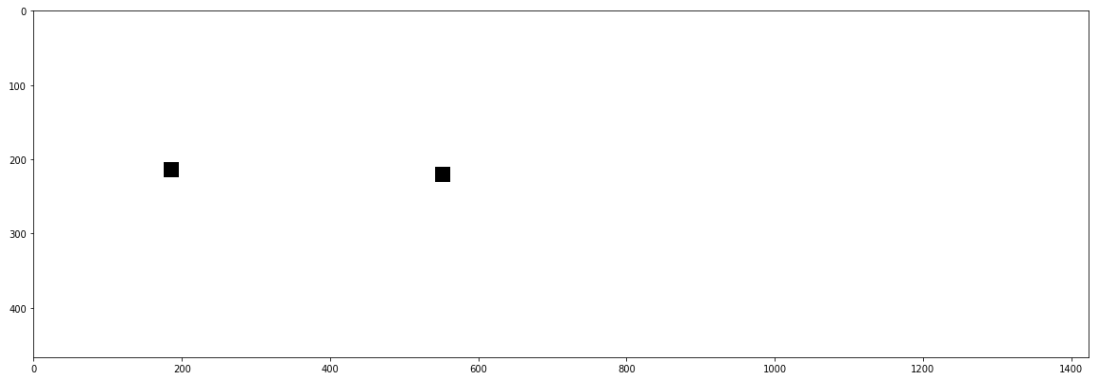


At this point, for each model, the list of coordinates which have received enough votes to be considered potential barycentres is not final: in fact, it should be refined and filtered before being considered the definitive one. For this purpose, two types of filters are adopted: the *neighborhood_filter* and the *distance_filter*. The former enables to skim the list eliminating outliers: each point is analysed in its neighbourhood and just those with at least a given number of near votes are kept. This operation is done twice: at first with a range of 15 pixels, and then, after having applied the other filter, with a range of 35 pixels. On the other end, the latter eliminates redundant barycentres scrolling the list and averaging between neighbouring ones, namely those for which Euclidian distance is lower than 100 pixels.
The result of these operations can be viewed on the screen via the function *plot_filtered_bar(accumulator, shelf_image, filtered_bar)*, which represents each found barycentre with a filled black square. For example, for the previous shown scenes and models, the result is this:

```
[[216, 165]]
model 0: Nesquik Cioccomilk
There are 1 Barycenter(s)
```




```
[[214, 186], [220, 553]]
model 24: Fitness
There are 2 Barycenter(s)
```




Finding the barycentre is not sufficient to be able to identify correctly the cereal box in its dimensions. This step, in fact, needs once again the matched features to work: indeed, given the correspondences, we have to compute a suitable transformation that brings points from the model reference system to the scene one. Being the considered objects planar, under the pinhole camera model this transformation can be addressed with an homography, a 3x3 matrix that maps points expressed in homogeneous coordinates from a 3D to a 2D space.
However, the estimation of the matrix can't be done considering all correspondences (which up to now have never been filtered), but it should refer just to those points that for sure belong to the analysed box in the scene. To achieve this goal, a list of "good" matches is computed for every found barycentre of every model:

via the function *find_correct_keypoints*, only those matched keypoints for which the joining vector in the r_table leads near the found barycentre are kept, because this proves they actually belong to the box instance.

From this point onwards, a series of precautions are adopted to be sure to correctly identify the instances: first of all, even before homography estimation, all barycentres which didn't receive enough good matches are eliminated. Then, exploiting matches coordinates, the transformation is computed using Random Sample Consensus (RANSAC), an algorithm that can discard outliers and that is able to make a robust correct estimation also with noisy data. Thank to this, we can realise a bounding box just transforming with the found matrix the known corners from the model image to the scene image and connecting them. Obviously, at this point, also height and width can be computed, as requested in the project.
It can be noticed that for ChocoKrave boxes a slight modification has been introduced for visualization purposes: the bounding box, in fact, is moved downwards due to an always present translation between reference and corresponding instances.

Three other precautions are taken: a shape control, a colour control and a perimeter control.
The shape control is based on the hypothesis that, in this application, boxes cannot be rotated or deformed by perspective projection to the point of becoming trapezoidal or, anyway, very different from a parallelogram. Such a result would be of course an evaluation mistake and should not be considered.
The colour control recalls the initial calculation of the average reference colour made on the models. Computing, in this case with respect to the lowest part of the bounding box, the average colour across the 3 channels, it is possible to compare the found instance with the original reference and discard those models which are very similar in terms of features but completely different in term of colours (e.g. ChocoKrave boxes).
Finally, the perimeter control checks whether all found boxes are approximately of the same size: in this way disproportionate bounding boxes will be avoided.

The last part is dedicated to visualization: with OpenCV defined functions the correct bounding boxes can be drawn on the image, also reporting the name of the model in their lower side. It can be noticed that across the whole project, and also in this case, in order to help visualization in all steps, the function *findmodel(I, check)* has been repeatedly called. This function just returns automatically name and number of the chosen reference.
A final print is then realised to display on screen the number of instances of every model, but also the centre (computed with *centroid(vertexes)* function) and dimension of each bounding box.
All project results are reported below.

<u>scene e1</u>


```
model 0: Nesquik Cioccomilk
Position: (162, 215) Width: 310 px Height: 441 px
number of instances: 1
model 11: ChocoKrave Nocciole
Position: (443, 195) Width: 299 px Height: 391 px
number of instances: 1
```

## scene e2


```
model 24: Fitness
Position: (167, 219) Width: 340 px Height: 492 px
number of instances: 1
model 25: Coco Pops
Position: (878, 240) Width: 302 px Height: 459 px
number of instances: 1
model 26: Nesquik Duo
Position: (538, 219) Width: 330 px Height: 494 px
number of instances: 1
```

## scene e3


```
model 0: Nesquik Cioccomilk
Position: (170, 237) Width: 328 px Height: 448 px
number of instances: 1
model 1: ChocoKrave al latte
Position: (817, 223) Width: 306 px Height: 411 px
number of instances: 1
model 11: ChocoKrave Nocciole
Position: (476, 220) Width: 308 px Height: 392 px
number of instances: 1
```

## scene e4


```
model 0: Nesquik Cioccomilk
Position: (158, 740) Width: 328 px Height: 448 px
number of instances: 1
model 11: ChocoKrave Nocciole
Position: (463, 718) Width: 305 px Height: 395 px
number of instances: 1
model 25: Coco Pops
Position: (553, 204) Width: 307 px Height: 446 px
number of instances: 1
model 26: Nesquik Duo
Position: (206, 195) Width: 341 px Height: 497 px
number of instances: 1
```

## scene e5


```
model 19: Country crisp
Position: (503, 190) Width: 292 px Height: 389 px
number of instances: 1
model 25: Coco Pops
Position: (161, 219) Width: 319 px Height: 433 px
number of instances: 1
```

## scene m1


```
model 24: Fitness
Position: (186, 217) Width: 363 px
Height: 504 px
Position: (550, 220) Width: 339 px
Height: 492 px
number of instances: 2
model 25: Coco Pops
Position: (1261, 235) Width: 303 px
Height: 452 px
number of instances: 1
model 26: Nesquik Duo
Position: (922, 219) Width: 330 px
Height: 491 px
```

scene m2


```
model 0: Nesquik Cioccomilk
Position: (179, 300) Width: 348 px Height: 452 px
number of instances: 1
model 1: ChocoKrave al latte
Position: (1180, 284) Width: 299 px Height: 418 px
Position: (848, 288) Width: 314 px Height: 415 px
number of instances: 2
model 11: ChocoKrave Nocciole
Position: (510, 280) Width: 311 px Height: 400 px
number of instances: 1
```

scene m3


```
model 19: Country crisp
Position: (1233, 190) Width: 289 px Height: 387 px
number of instances: 1
model 25: Coco Pops
Position: (557, 214) Width: 329 px Height: 449 px
Position: (890, 228) Width: 313 px Height: 448 px
number of instances: 2
model 26: Nesquik Duo
Position: (195, 206) Width: 365 px Height: 513 px
number of instances: 1
```

scene m4


```
model 24: Fitness
Position: (163, 193) Width: 377 px
Height: 503 px
Position: (541, 196) Width: 358 px
Height: 500 px
number of instances: 2
model 25: Coco Pops
Position: (1273, 198) Width: 313 px
Height: 426 px
Position: (1589, 215) Width: 300 px
Height: 441 px
number of instances: 2
model 26: Nesquik Duo
Position: (921, 195) Width: 343 px
Height: 499 px
number of instances: 1
```

scene m5


```
model 1: ChocoKrave al latte
Position: (833, 746) Width: 299 px Height: 414 px
Position: (500, 752) Width: 315 px Height: 413 px
number of instances: 2
model 11: ChocoKrave Nocciole
Position: (163, 746) Width: 312 px Height: 402 px
number of instances: 1
model 19: Country crisp
Position: (910, 192) Width: 293 px Height: 385 px
number of instances: 1
model 25: Coco Pops
Position: (235, 214) Width: 333 px Height: 440 px
Position: (567, 228) Width: 314 px Height: 444 px
number of instances: 2
```

Step C (optional) - Whole shelves challenge:

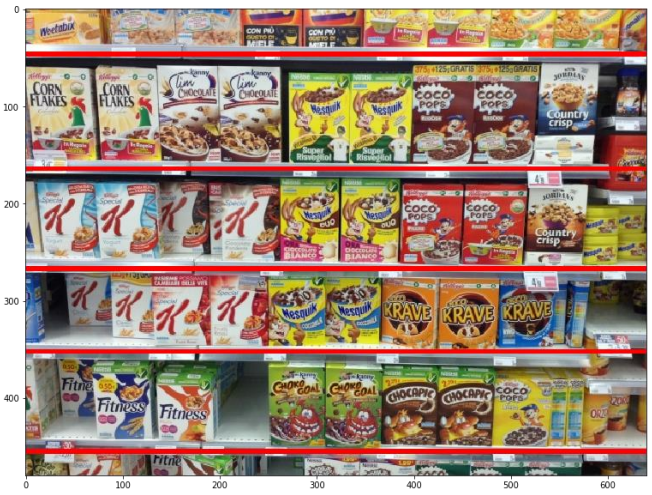
This second part of the project introduces a more challenging scenario: many different product instances for each picture, distractor elements and low-resolution images. Consequently, despite having set up the problem resolution in a similar way to the previous case, some substantial changes have been introduced.
It should be noticed that, being not able to create an algorithm totally independent from given images as before, we chose to make assumptions, whenever needed, on the given models and not on the scenes.

In order not to repeat concepts already discussed, just the main differences between the two cases are presented in this report:

1. Since the resolution of the instances contained in the scenes is quite low compared to the models one, the latter are blurred by a Gaussian function with a Kernel whose dimension increases together with the reference model width.

2. Models are treated differently according to their characteristics: indeed, some of them can be differentiated only because of their colour, being different flavours of the same type of cereal. As a consequence, for those models, a region of interest is decided for the colour analysis and all necessary information for the next steps are computed (both filtered histograms for each channel and colour pixel percentage calculation).

3. Dealing with the scenes, before analysing all the aspects related to the keypoints and the matches, we decided to crop them into different sub-images in order to simplify as much as possible the research. The cropping can be made exploiting the horizontal structure of the shelves: scene is at first blurred and segmented into foreground and background, using respectively *cv2.GaussianBlur* and *cv2.adaptiveThreshold*. After that, an opening operation and a successive dilation are used to highlight those lines of interest (relying on a horizontal structuring element sufficiently wide in length). Studying the obtained binary image along the vertical direction, we are then able to detect the shelves. The result of this step is reported below for scene h1 with red lines:



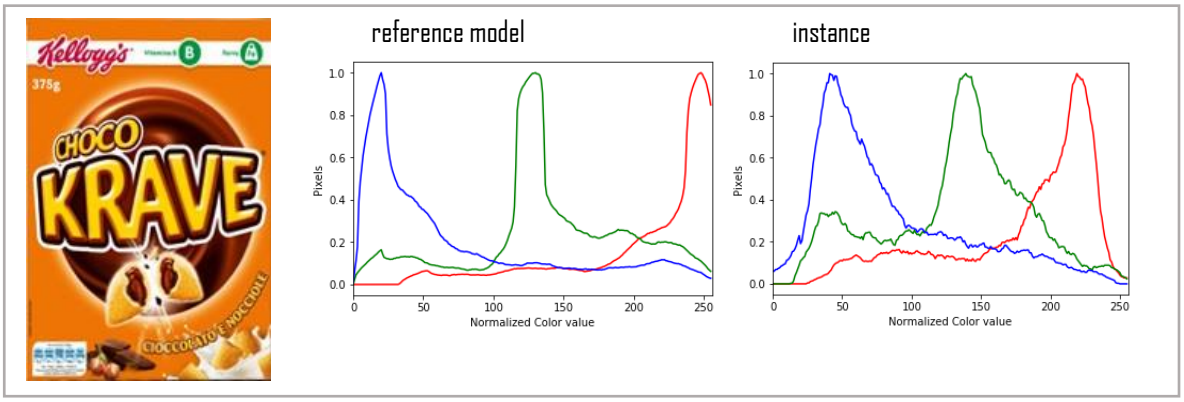

Each cropped image is analysed separately: too small sub-images are kept for later, in order to reconstruct the bigger image in the end; all the others are subjected, in order to find the barycentres, to the application of a similar algorithm to the one explained before.

4. In the new version of the research step, just those correspondences showing a distance lower than 200 are considered, to avoid an excessive number of outliers. A simplified version of the filters is then used

to detect correct barycentres. Indeed, the number of detected keypoints is greatly smaller than before. Going into detail, the function *distance_filter* is exactly the same, while *count_bar* appends to the barycentres list those points initially neglected because of their individual vote (<2), but which present enough voted pixels in their neighbourhood.

5. Bounding boxes, assuming rotation of instances in the shelves is almost zero, are computed exploiting the ratio between the average sizes of destination and source keypoints. In this case, in fact, the number of correct matches is not big enough to exploit a RANSAC algorithm to perform a homography estimation.

6. The final acceptance test is carried out in 3 different ways according to the type of model:
   - For unique models which don't have colour similarities with others, the bounding box is directly drawn on the image (we avoided colour control because it could be seen as an additional unnecessary computation time, having many different models to analyse).
   - For models of the same brand, but with very similar graphics and colours, we evaluated two possibilities: a histogram evaluation of the max values position, which is model independent but quite difficult because of tags, distractors and other interferences (this turned out useful just for simpler models, i.e. those with higher difference in colour); and a direct colour research, more related to the chosen model but, at the same time, more robust (applied in the case of *difficult_models*).
     The former relies on the fact that histograms of the same boxes should look really similar in terms of shape, also under translation given by light, as it can be seen from the following example (model 11 (ChokoKrave Nocciole) in scene h1):




The latter, in general, counts the percentage incidence of red, blue, green and brown pixels in the previously defined areas and then compares them to the reference model's ones, admitting a certain possible error that could vary according to the model and the number of found pixels. Below, an example of acceptance and refusion are proposed (model 10 in scene h1):

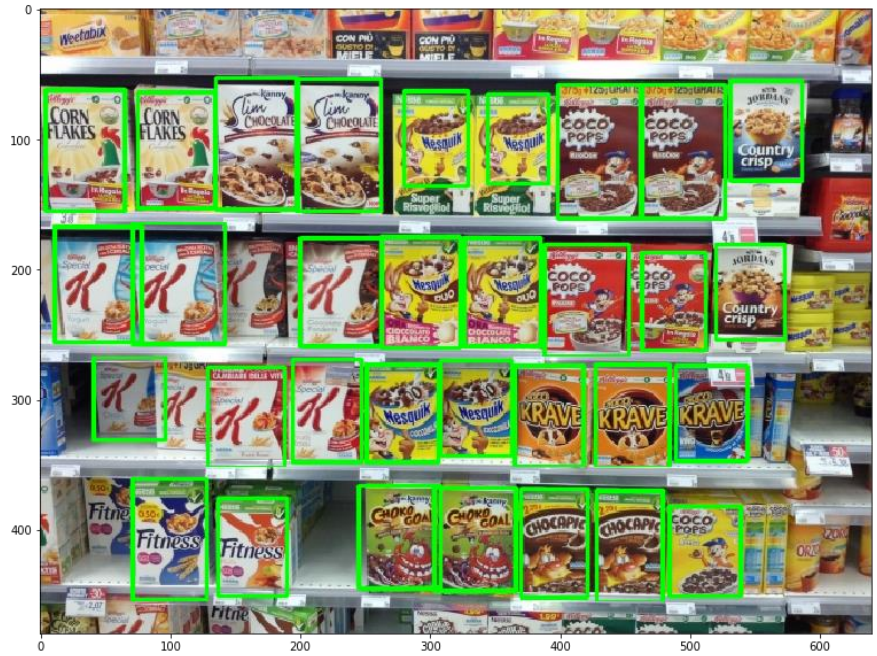
```
colore: 16.71 0.0 0.68 Brown: 0.25
colore modello: 17.91 0.26 0.0 Brown: 10.86
accepted: False
```

```
colore: 26.28 0.0 0.0 Brown: 10.77
colore modello: 17.91 0.26 0.0 Brown: 10.86
accepted: True
```

After the instances detection all cropped images are then composed together to reconstruct the chosen scene. The final result is quite satisfactory: a great percentage of instances is detected correctly. Moreover, those which couldn't be found presented either the barycentre outside the image (with the impossibility to identify it within the accumulator and the scene) or too many changes and occluded areas with respect to the models proposed, making the SIFT not able to find enough correspondences. All the obtained images are reported below. The printed barycentres and dimensions, on the other hand, can be seen executing the code, as they would take up too much space to be legible.

Additional note: *(light blue) yogurt special K* did present, in the directory, a wrong reference model (the same of model 9, *Special K Classic (blue)*). As a consequence, we tried, when possible, to detect them as model 9, being them quite similar. Also, we noticed that the instances of *Nesquik Duo* in the scenes were referring to model 26 and not 6, so we exchanged them in the analysis.

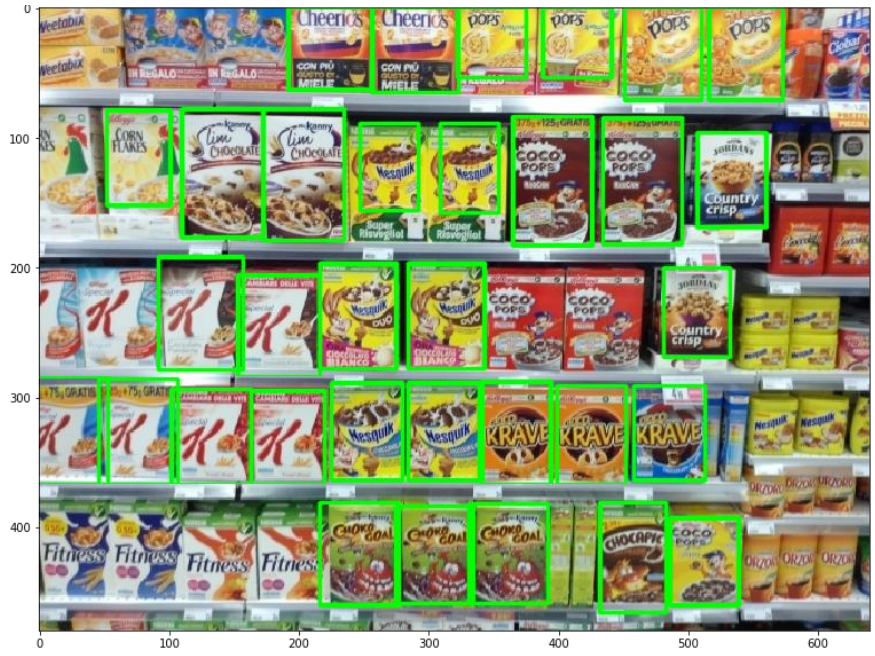
scene h1



Instances number:
32 detected models/35 known models with barycentre inside the image
Total models detection: 91,4%

Not partially occluded models detection: 100% !!

scene h2



Instances number:
33 detected models/43 known models with barycentre inside the image
Total models detection: 76,7%

scene h3

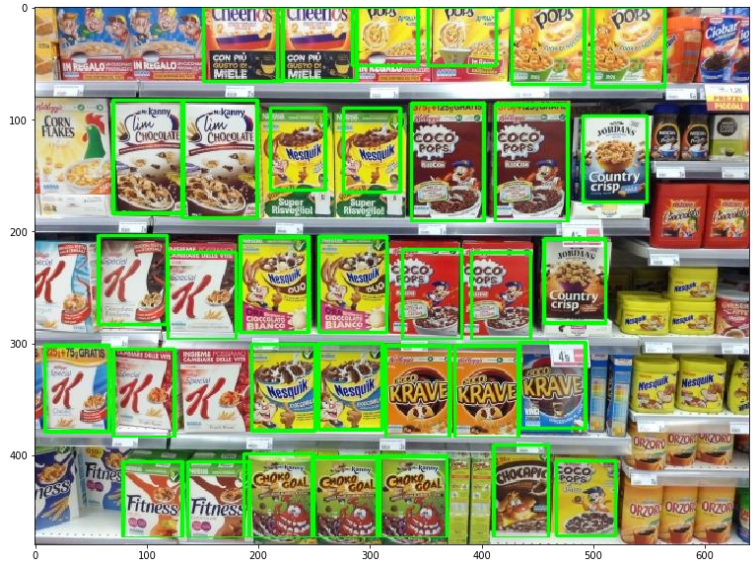
Instances number:
35 detected models/39 known models with barycentre inside the image
Total models detection: 89,7%

scene h4

Instances number:
38 detected models/41 known models with barycentre inside the image
Total models detection: 92,6%

Not partially occluded models detection: 100% and many cut patterns identified!

scene h5



Instances number:
34 detected models/39 known models with barycentre inside the image
Total models detection: 87,2%

Ludovica Ghinelli and Salvatore Bamundo