Product Recognition on Store Shelves

Damiano Bolognini
Department of Computer Science and Engineering,
University of Bologna
damiano.bolognini@studio.unibo.it

Giulio Todde
Department of Computer Science and Engineering,
University of Bologna
giulio.todde@studio.unibo.it

ABSTRACT

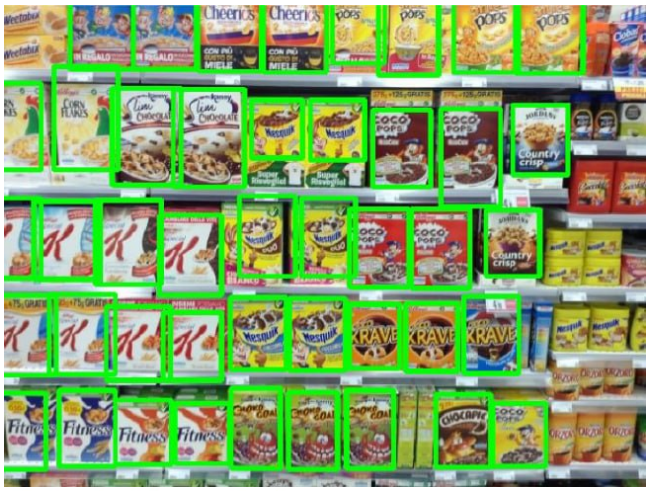
The goal of this project is to use computer vision techniques to perform object detection. We want a system able to recognize cereal boxes on store shelves. The task is complex, due to the high number of different possible variation in which the same object can occur into the scene. Our solution combines the SIFT algorithm and the Mean Shift clustering scheme.

1 PROJECT DESCRIPTION

The goal of the project is to develop a computer vision system that, given a reference image for each product, must identify boxes of cereals of different brands from one picture of a store shelf, as seen in the Figure 1.

We face three different tasks:

- Task A: "Multiple Product Detection". The object detection system must identify single instance of products given: one reference image for each item and a scene image.
- Task B: "Multiple Instance Detection". In addition to what achieved at step A, the system must be able to detect multiple instances of the same product.
- Task C: "Whole shelve challenge (optional)". The system must detect as much products as possible in a challenging scenario: more than 40 different product instances for each picture, distractor elements (e.g., price tags…) and low-resolution image.



```
Product 0 - 2 instance found:
    Instance 1 {position: (256,328), width: 57px, height: 80px}
    Instance 2 {position: (311,328), width: 57px, height: 80px}
Product 1 - 1 instance found:
…
```

Figure 1: Desired output

2 PROPOSED SOLUTION

The strategy followed to achieve this problem is the same of a classical object detection system: extract the salient points from the target and the model image (train and model respectively), compute their descriptors and finally match the descriptors between the two images. The easiest way to implement this pipeline is to use SIFT [1] algorithm to extract the salient points and compute the descriptors, then apply the FLANN [2] algorithm to perform the matching and finally extract the homography using the RANSAC [3] method in order to compute the bounding box of the retrieved object. Before undergoing the extraction of the salient points, we apply a Gaussian local filter [4] to both model and train image, this simple operation led to a high improvement in the object detection. Furthermore, since the SIFT algorithm is used usually to recognize a single instance of the object in the scene, we have exploited a clustering algorithm, particularly MEAN SHIFT [5], to extract multiple instances of the same model object inside the scene. All these instruments are implemented inside a single function, in this way we can use the same system for different tasks changing only the hyper-parameters required, making our system as scalable as possible.

2.1 The Gaussian Filter

We have chosen the Gaussian filter because it has the property (already exploited inside SIFT) to suppress noise and secondary details, as if the image is seen from a bigger distance, see the Figure 2. This behaviour is optimal for our application because in general the given model images have a higher resolution and level of detail than the train ones, especially inside the task C, where the objects in the scene are small and blurred. The idea is to make the model image as similar as possible to the train one, this led to an improvement in the extraction of the key points, thus we will apply a stronger filter to the model images, and a light one to the train images.

To achieve that, we change the sigma parameter of the Gaussian filter: higher is the sigma the more blurred the image will be.

Another filter we have considered is the Bilateral one [4], which remove the noise from the image without blurring it. Even if the results obtained are not bad, it has not the same efficacy of the Gaussian filter, precisely because it does not blur the image, furthermore the Bilateral filter is slower than the Gaussian, and its hyper-parameter optimization is harder because it requires a higher number of hyper-parameters.

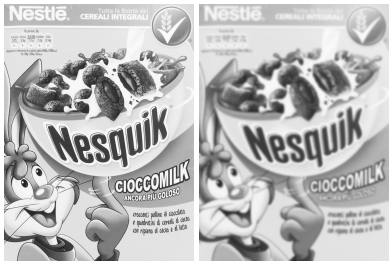


Figure 2: Example of Gaussian blur

2.2 Image recognition

Once applied the Gaussian filter, we proceed at the extraction of the salient points and the computation of their descriptor using the SIFT algorithm. After this operation, we start the clustering to divide the instances of the queried object present in the same scene, the main idea is that the salient points close to each other will belong to the same instance. To cluster the salient points, we have chosen to use the MEAN SHIFT algorithm because it allows to choose the cluster dimensions by simply changing a single hyper-parameter: the quantile. In general, bigger are the objects inside the scene, bigger the quantile value must be (if it is equal to 1 it will create only one cluster), and analogously with small objects. Created the clusters, we proceed to match the descriptors of the model image with the ones of the clusters, we focus on a cluster at the time. We have chosen to use the KD-TREES [6], in this way we have a more efficient search, in particularly we use FLANN which consists of a collection of optimized algorithms for the nearest neighbour problems. Then we eliminate the weak matches using a threshold; and if the number of good matches obtained for the instance is high enough, we apply the RANSAC algorithm to estimate the homography. Finally, we proceed to draw the bounding box of the various instances over the scene.

3 HYPER-PARAMETER OPTIMIZATION

In the task A and B, we have achieved good results by choosing the hyper-parameters empirically, but in the task C this was not possible because the image recognition is harder due to the bad quality of the images, so we decided to perform a hyper-parameter optimization. We have chosen the starting hyper-parameters and then we performed an automatic grid-search around these values to extract the best combinations of them. The hyper-parameters to be tuned are the following:

- Quantile: the clusters dimension;
- Sigma model: the "amount" of blur of the model image;
- Sigma train: the "amount" of blur of the train image;
- Minmatch: the minimum number of good matches the cluster has to have.

We have chosen a first range of values to test, and we stored the results inside a data frame to analyse them. To determine if the recognition made by our system is right, we used the YOLOv3 [7] labels: if the centre of the bounding box computed is inside the region indicated by the corresponding label, then it is a true positive, a false positive otherwise. After the computation of the first results, we classified them based on the number of true and false positives and their ratio. We have performed this grid-search three times, and inside each one we have chosen different ranges based on the results obtained in the previous search, to obtain at the end the best combination, as we can see in the Figure 3.

Quantile	Sigma Model	Sigma Train	MinMatchCount	Sum TP	Sum FP	TP - FP
0.035	2.5	0.4	14.0	91.0	27.0	64.0
0.035	3.0	0.4	13.0	89.0	26.0	63.0
0.035	3.0	0.3	13.0	87.0	25.0	62.0
0.035	2.5	0.3	13.0	95.0	34.0	31.0

Figure 3: Best sets of hyper-parameters for Task C

Figure 4: Bounding boxes and coordinates of the recognized boxes

4 FINAL RESULTS

We can see the final results in the Figure 4.

Task	TruePositive	FalsePositive	TrueNegative	FalseNegative	Recall	Precision	Precision (%)
A	13	0	0	1	0.929	1	100
B	22	0	0	1	0.957	1	100
C	101	10	43	74	0.577	0.909	91

Figure 5: Recall and precision of the different tasks

Figure 6: Low (left) and high (right) quality box images

In addition, we computed the recall and precision of the different tasks, as presented in the Figure 5, (we have considered the Choco Krave® recognition right even if the colour of the box is not the same, i.e., if the system recognises the orange box as the blue one and vice versa).

Inside the task C, the cereal boxes in the model images are often different from the ones inside the train images: even if the product is the same the packaging is different; additionally, some model images have a better quality of others, so they are usually recognized more effectively.

5 FINAL CONSIDERATIONS

The system created can recognize the big majority of the instances present inside the scene, but not all of them. These errors can be attributed to different factors: the system works on grey images, so it does not consider the colours, so similar objects with different colours are often recognised as the same object, the main example of this behaviour is the Choco Krave® box. A validation mechanism based on the colours intensities could overcome this problem, for example it could run the system three times considering at each time only one colour, but this would make the system far slower. Furthermore, some given model images are not optimal to achieve the task, indeed some of them have a dimension or quality too low, making them difficult to recognize. The most obvious example of this phenomenon is the model image 11: we tried to replace it with an image of the same box but with a bigger dimension and better quality, and this led to a better classification, we can see both images in the Figure 6.

In addition to this problem, some cereal boxes inside the model images are not the same of the ones present inside the scene, and since the system is based on the extraction of the salient points, this led to a worsening in the performances. The last problem, this time not fixable, is the high resemblance of different products of the same cereal line, where the main details such as the logo and the mascot are the same. Finally, we have tried to develop a CNN (YOLOv3), but the lack of examples provided makes impossible this approach, leading to a weak network, able to recognize only few boxes.

We conclude by saying that, with the provided images, both in quantity and quality, the system created is strong and valid, even if it is necessary to perform a hyper-parameters optimisation which could be further optimized.

REFERENCES

[1] SIFT: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
[2] FLANN: https://docs.opencv.org/master/db/d18/classcv_1_1flann_1_1GenericIndex.html
[3] RANSAC: https://docs.opencv.org/master/d1/de0/tutorial_py_feature_homography.html
[4] Gaussian and Bilateral filter: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html
[5] MEAN SHIFT: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html
[6] KD-TREE: https://docs.huihoo.com/opencv/2.4/df/d23/classcv_1_1KDTree.html
[7] YOLOv3: https://pjreddie.com/darknet/yolo/