COMPUTER VISION: Boat Detector

Enrico Convento ID number:3027572 email: enrico.convento@unipd.it

July 23, 2021

1 Introduction

1.1 Activity Goal

Before introducing the main goal of the activity, a brief dip into the main theme is needed. The project focus on automated analysis of traffic scenes, in particular it focuses on boat traffic in the sea. The goal of this project is to develop a system capable of detecting boats in the image. Different approaches have been implemented through the road to get the final result. This report has not only the purpose of describing final results, but also to provide a full path through the design steps that have been followed. The image presented in Fig. 1 will be used as a sample image to present the various steps that make up the algorithm, this is an image belonging to the test dataset used only as example and not included in the training process.

Figure 1: Input image

1.2 Project structure

The project is been developed in *C++* with the OpenCV library. Also machine learning techniques have been utilized, thus also a small amount of *Python* code has been make use of. The *object detection problem* consists in determining the location and scale of all object instances, if any, that are present in an image. Thus, the objective of an object detector is to find all object instances of one or more given object classes regardless of scale, location, pose, view with respect to the camera, partial occlusions, and illumination conditions. In this case, locating objects by means of a bounding box and the associated class.

1.3 R-CNN: an overview

1.3.1 R-CNN: standard approach

The project draws inspiration from the *R-CNN* object detection algorithm presented into the paper *Rich feature hierarchies for accurate object detection and semantic segmentation written by Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik*. This work is published in 2013, since then several developments have been presented, however many of these are end-to-end learning approaches that cannot be used in this case. The brilliant idea on which this algorithm is build came from

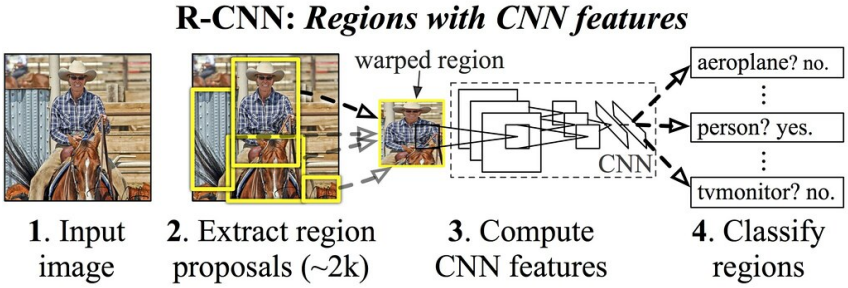


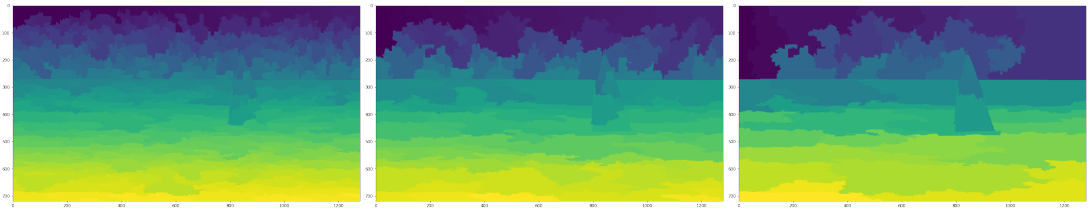
Figure 2: R-CNN: standard scheme

mixing up the result from another interesting paper, named *Selective Search for Object Recognition presented by by J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers and A.W.M.* and the results of applying CNN to the image classification problem. In short summary, the R-CNN generates around 2,000 category-independent region proposals for the input image, extracts a fixed length feature vector from each proposal using a CNN, and then classifies each region with category-specific linear SVMs. The process is summarized and depicted in Fig. 2.

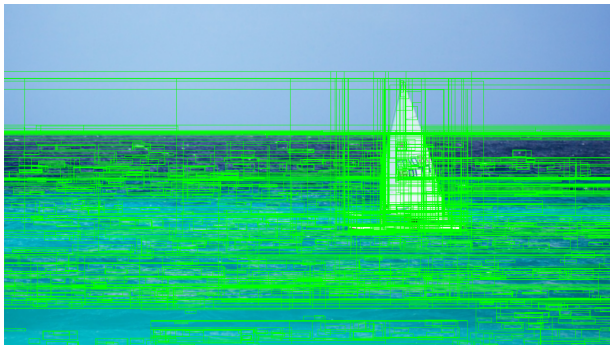
Selective search Among the different steps that make up the process, the second step will be focused in particular, that is the one that concerns the region proposal. The region proposal algorithms are what makes the difference compared to a brute force sliding window approach, as they decrease the computational needs, and it should be emphasized that despite this, in this algorithm is not the lightest in these terms. *Selective search* is one of the most successful category-independent region proposal algorithms, in a nutshell, it exploits a hierarchical grouping algorithm to form the basis of selective search, applying Felzenszwalb and Huttenlocher's efficient graph based segmentation algorithm to create small partition of images called "initial regions". Then selective search later uses a greedy algorithm to iteratively group regions together, it merges together region according to color, size, texture and shape. The *OpenCV* library provides a set of methods to implement this algorithm, in the project the function *utilities::selectiveSearch* encapsulate all of this, it takes as input an image and return a vector of boxes that represent the result of the algorithm. The results of the "region grouping" steps are shown in Fig. 6, notice that the algorithm correctly group segmented regions. As last thing in Fig. 3(d) the final result of the selective search algorithm is shown.

1.3.2 R-CNN: modification

Although the approach used closely follows the RCNN lineup, some changes have been made for various reasons. First of all, the use of the pre-trained model MobileNet. The standard algorithm to carry out steps three and four uses AlexNet to extract features from the proposed regions and



(a) *Segmentation: step 1* (b) *Segmentation: step 2* (c) *Segmentation: step 3*



(d) *Selective search: output bounding box*

Figure 3: Selective search results

then classify them using SVM. This approach was initially followed step by step using VGGNet in place of AlexNet since the last is relatively obsolete. Basically VGGNet model is taken, from it the last layers are modified as it is used to extract features, the network is trained, then the SVM classifier is created, and substantially the model is complete. However, due to the lack of computing power available, MobileNet was chosen as it is a simple but efficient and not very computationally intensive convolutional neural network, incorporating the last two steps into one efficiently, namely it incorporates all the machine learning part in one step. In Tab. 1 it can be seen that MobileNet is actually much lighter than VGGNet and the accuracy level is almost the same.

Model	Accuracy	Parameters (millions)
VGGNet 16	71.5	138
MobileNet v2	71.8	3.47

Table 1: Network comparison

1.4 Training

In this section the training process is described, to learn the model the two dataset provided in the project delivery have been used. Boat detection is a complex task, different scales, intra-class variance and perspective are some of the main problems that need to be faced, then the training must take into account all of these.

1.4.1 Data preparation and ground-truth

One of the main steps that are basically indispensable for this kind of task is the **dataset preparation**. Assuming that the *ground truth* information are available, in the following this problem will also be addressed, the training dataset is composed by two blocks, namely the images and the ground truth

files.

Ground truth Ground truth files have been collecting, cooperating with other colleagues using the functionalities provided by *https://superannotate.com/*, those files have been extracted for each image in Kaggle and MAR datasets. In case of object detection task applied for object detection, the ground truth represents the desired output given an input. The ground truth contains the coordinate of the "true" bounding box that contains the boat represented by a bounding rectangle, which is defined by top-left corner coordinates, width and height associated to the bounding boxes. The structure of the files used in the project related to this is a little different respectively for training and testing, in particular as regards the training, for each image a file with the same name is associated but with a .json extension is used as it is easier to treat by scrolling through the images, while for the test a single .json file was used as in this case it is used as a dictionary in which you go to look for the values of ground truth to measure the performance, in order to read this kind files the *json.hpp* library file is been used. The produced files have been submitted, and they are the *COCO_Training_Kaggle.json*,*COCO_Training_MAR.json* and *mar_annotation_test.json* which make up the contents of the folder *GT*.

Dataset preparation The dataset preparation is composed of two parts, first for a given image, from to the ground truth file associated the true bounding box coordinates are extracted for each box in the image. Now that the ground truth for the image is available, the next step is to apply the selective search algorithm used to extract the region proposals as show in Fig. 3(d) and then a selection is made, to speed up the image was scaled by a factor of two. The metric used in this

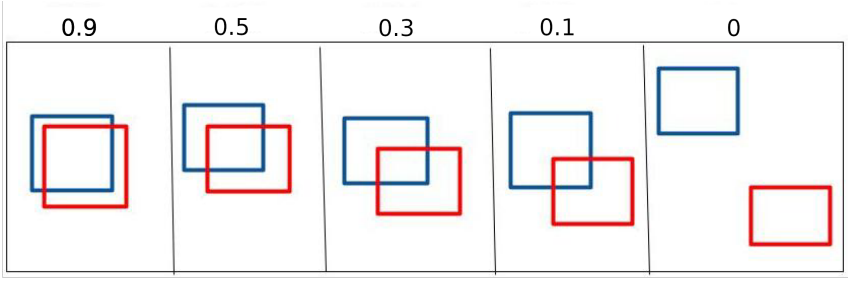


Figure 4: IoU: visual intuition

case is the **IoU** (Intersection over Union), it describes the overlap between two boxes, the associated value ranges between 0 and 1, if the intersection area is small then the IoU value is closer to zero and then this mean "bad" overlapping, otherwise if two boxes that completely overlap, the area of the intersection would be equal to the area of their union, and therefore the IoU would be equal to 1. A better visualization of these facts is shown in Fig. 4.

The selection phase return, following the paper, what are defined as *positive and negative samples*. As *positive sample* the ground truth box is taken, in Fig. 5 that box is highlighted in blue color. Then for each image a scan of all the proposal boxes obtained in the step before is performed, and a set of five boxes that have IoU less than 0.3 with the corresponding ground truth rectangle is grab, defining the so called *negative samples* point up in red color in Fig. 5. This process is been implemented in the file named *dataset_preparation.cpp*, it essentially fills two directories with those positive and negative samples respectively named *boat* and *non_boat*. Of course, in the real dataset preparation the overall process is repeated for each training image, while in the files submitted this

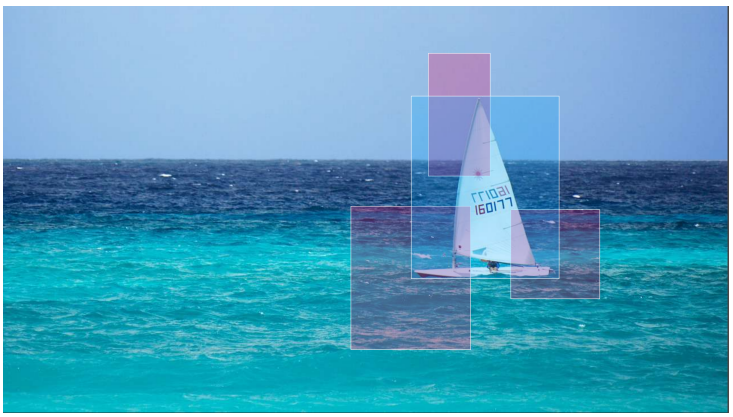


Figure 5: Positive and negative samples

part can be tested with only two images.

1.4.2 Transfer-learning

At this point half of the work has been done, all these samples are saved, and we switch to *Python* in order to train the network by providing it with the data just collected. What is done is precisely called tansfer-learning, through this process instead of starting the learning process from scratch, you start from patterns that have been learned when solving a different problem. In this way you leverage previous learnings and avoid starting from scratch. To implement it, the pre-trained on ImageNet model MobileNet v2 is used. The last part of the training process is implemented in the notebook *training.ipynb* in which the previous folders must be uploaded. This part cannot be tested since it need to upload files on Google Drive. After learn the model, it is downloaded and ready to be tested, in the project the model is saved as *model.pb*. It is necessary to underline that the dataset was built by joining together the two supplied datasets, respectively Kaggle and MAR, as we wanted to obtain a model as general as possible. This is done at the cost of losing a bit of performance, as the range of possible cases is much more varied in this case.

1.5 Test

In this section is shown how the test is performed, this is not the only purpose of this section, in order to provide a better explanation of the results a comparison of strengths and weaknesses will be made. The testing process can be divided into three areas, the pre-processing part, the actual test and in the end the post-processing part. All the code needed to test the system can be found in the *main.cpp* file that takes support form the *utilities.cpp* which include several functions that are used in the main program.

Pre-processing The preprocessing part is implemented by the function *utilities::imgPreprocessing*, this subroutine permits applying to the input image a Gaussian smoothing or perform histogram equalization, the aim of the blur is to remove useless details while the histogram equalization enhancement has the purpose of give more strong to the boat patterns. This image processing attempt was intended to help the algorithm to highlight parts of the image or to remove details, however from the results it is observed that it is not a fundamental part but a light Gaussian smoothing can help.

Main part At this point, the main algorithm can be executed. As from the lineup, the first step is to perform selective search, this step takes a lot of time especially if an image belonging to the MAR dataset, this is in fact one of the weaknesses of R-CNN that will be partially solved by the developments of this algorithm. However, the number of proposed regions feeding the neural network can be limited, as the algorithm often provides more than ten thousand sample images. Once the regions have been extracted, the next play consists on using the previously trained model for classifying those regions in order selecting those that appear to contain boats. Before applying the the model the regions that are warped into images of size $224 * 224$. The module *cv:dnn* provides functions to perform this kind of operation, everything done is encapsulated in the function *runDNN* that is composed by the functions *blobFromImage(),setInput()* and *forward()* which in sequence create a blob from each region that becomes an input for the model, and output for each region a value that ranges between 0 and 1 which represents the score on the prediction.

Post-processing Now it is the time for post-processing, this part is the more complex from an image processing point of view. The ladder followed is explained in the following:

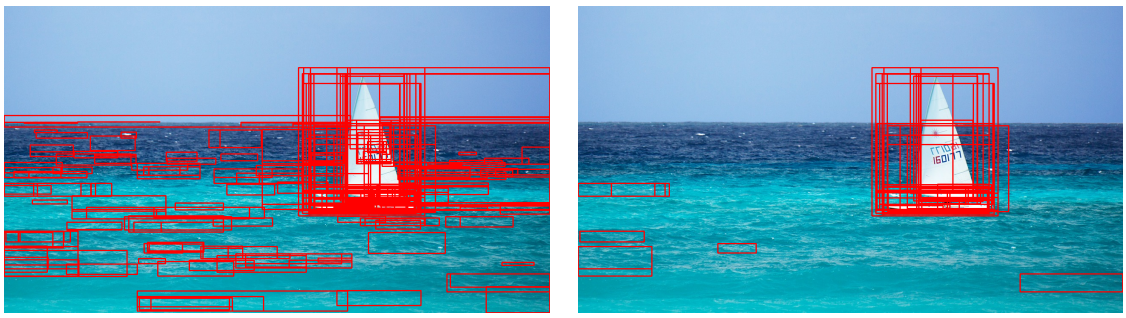
1. The network outputs a confidence vector, which precisely for each rectangle gives us an accuracy value on the result. This vector is immediately useful as it allows you to sort the array of boxes in order of confidence using the function *utilities::argSort*, which in addition to ordering returns a predetermined number of boxes in order to thin out the vector of region proposals. The number of these boxes is setted to 300, a trade-off is needed to be made in order to fix this threshold since a lot of boxes helps in images in which there are a lot of details or multiple objects while a small amount of boxes helps in the opposite case, a threshold must be set in any case in order to make the algorithm robust.

2. The sorted vector returned from the previous step is used to perform Non-maxima suppression thanks to the function *utilities::runNMS*. This function is based on two thresholds, *confThreshold* and *nmsThreshold*. The algorithm does perform the following step:

   (a) It starts removing all the boxes with score under *confThreshold*

   (b) After this thinning, from the survived boxes it makes a boxes comparison based on IoU using the other threshold, it is actual the box with the highest confidence with the those have survived the first drop out and delete all the ones that have an high overlap but low confidence.

   The thresholds are set to respectively 0.2 and 0.95, the second threshold that regards *nmsThreshold* is quite high, this come form the fact that NMS only select the number of regions that are within certain limits but it does not change the geometry of the latter, for this reason it was preferred to have a soft effect and solve the problem in the following steps. The results obtained applying NMS to the previous set of boxes is presented in Fig. 6 in particular it can be observed the result before (a) and after applying this suppression algorithm (b), important to notice how the algorithm has a strong impact on the number of boxes. NMS is at the same time one of the strong point of the algorithm and one of the weakest, this algorithm is strongly correlated with the goodness of the model, if the model is not well trained the confidence very likely returns unreliable values that cannot be used. To achieve the actual model several trainings were needed, and the results is not at all satisfactory.



(a) *Argsort result before NMS* (b) *NMS result*

Figure 6: NMS comparison

3. The third step is used to remove all the "uninteresting" areas like sea, constructions or other things that do not identify a boat, in particular an edge map is used so that only boxes containing strong edges are kept. Basically the process move to an edge map using Canny algorithm with the aim of extract the stronger edges. For each box the number of non-zero pixels that it delimits in the edge map is counted, through the function *cv::countNonZero* and then a filtering operation is performed through an empirically established threshold. This passage is not crucial in images where the level of detail is poor. The image presented in Fig. 7 is a very representative image of this case, to show the effect of the filtering operation the thresholds have been modified for the optimal ones, but this has been done to show that this type of operation can help by removing areas that the detector should not detect as, in this specific case, the sea. The changes can be observed by looking first at Fig. 7(a) and then in Fig. 7(b). It is important to remark that there exist cases in which just the NMS returns

(a) *Edge threshold disabled* (b) *Connected components*

Figure 7: Edge thresholding effect

regions that correctly match the boat, in those cases, this method does not modify the regions which are output directly, these cases will be dealt with later.

4. The last passage of the test process is the one that connect and return the final bounding boxes. The function used is *utilities::densityThresh* that solves the problem of merging different rectangle that surround the same boat. It uses the concept of accumulation cells, the input of this step is the set of boxes returned in step 3, and what is done in basically create a gray scale image in which the pixels value ranges from 0 to 255, initially all the pixel have value equal to zero, and each box is used as a mask on this gray scale image. Given a box, each pixel value is summed "$+1$" in the area surrounded by the box, this for each bounding box. This accumulation process builds a density map that show the distribution of the boxes as shown in Fig. 8, quite simply if more rectangles overlap, the more the value on the map grows,

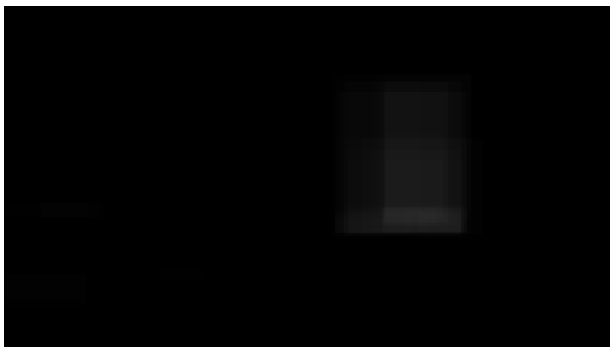


Figure 8: Density map

high density zones have a whitish color. At this point, a thresholding is done with threshold value *th* equal to 4, or in different words it searches for at least four overlapping rectangles. At this point it is very likely that more connected components are present on the map for which the function is used *connectedComponents* to label them and then new rectangles are built in a geometric way in order to incorporate all the connected component. In Fig. 9 the labeled component extracted from the sample image is shown, of course, due to the fact that in the original image only one boat, only one component is visible. Looking at Fig. 11(b) more labeled components can be visualized.

Figure 9: Connected component

1.5.1 Performance measurement

In conclusion, it is important to sum up and evaluate the results by comparing the strengths and weaknesses that the algorithm and its implementation possess and if possible introduce methods to solve these problems. In this section, several images will present in order to summarize in a general way all the problematic cases that can be observed running the program. In each image in the following a comparison with the ground truth can be observed in a visual way and also the numerical results of the IoU is depicted in the image. In all the following images, the blue bounding box represent the ground truth, while the red one is the predicted bounding box. In Fig. 10(a) is depicted the result obtained at the end of this long path that follows all the various steps that the program takes. The result is one of the best obtained and this is due to the fact that this is a very simple image in which only one boat is present, the background is not so complex and also the boat size facilitates detection. On the other hand, Fig. 10(b) show an interesting case of multiple boat detection, in this case the result could be considered satisfactory since the accuracy is around the 70% but multiple detection is a not trivial problem. Taking up what was said above and looking at

(a) *Edge threshold disabled* (b) *Connected components*

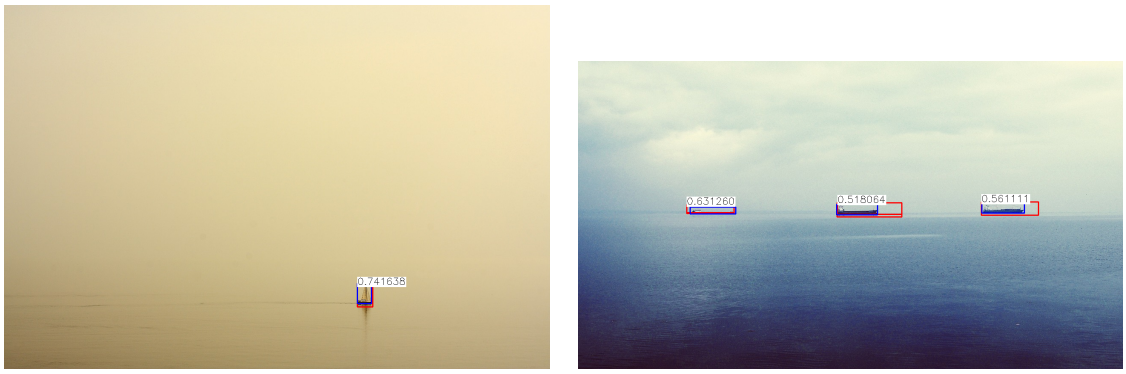
Figure 10: Simple detection cases

Fig. 11(a) it can be observed that the results is not sufficiently good, as multiple boat are enclosed in the same rectangle. From Fig. 11(b) the cause of the problem can be deduced, the gray color connected component is transformed into one unique box since it includes more boats, however if you look carefully you notice that it distinguishes the two different components, however, the thresholding work makes sure that the result is not the desired one. This problem arises each time that more boats are very close each other and cannot be solved with the strategies used. On the other hand, it has been noted that by re-training the classifier and improving it, this problem tends to decrease. The MAR dataset is more difficult to handle due to the fact that often it presents a confused situation, rich of details that sometimes present patterns similar to boat. In order to solve this problem a possible solution is to modify the threshold *th* in *utilities::densityThresh*, in particular set an higher threshold, this can be led to a better result in this singular case but since the final aim is to get a system that is able to obtain acceptable results in a more variety of cases this can be deleterious. Among the pathological cases that create problems in the system is the case in which the size of the boats to be detected is small. In this case, all the post-processing done is a problematic step to perform. This is due to the fact that NMS algorithm very thin the boxes vector as the probability values are very low in this case and furthermore the threshold work performed after NMS plays with overlap values of at least 4 rectangles, consequently if the rectangles are few or even less than the threshold the algorithm stops. To avoid this, a control is performed on the number of boxes returned

(a) *Connected components problems* (b) *Connected components*

Figure 11: Thresholding strategy problem

by the NMS, if the number of boxes is very low then the result of the NMS is returned as the final result. This certainly has advantages, i.e. it allows detecting very small boats as in Fig. 12 however it can cause the detection of boats even if they are not present. An other interesting problem is



(a) *Small size detection* (b) *Small size multiple detection*

Figure 12: Sizes problems

related to the illumination, the ideal for this type of systems is that they work at any moment of the day with lighting sources that can be different. In Fig. 13 we observe a case in which the algorithm fails to detect a boat due to a headlight pointed towards the camera in a night context, with the current implementation there is no way to solve this problem as the boat it is completely obscured by the light beam.

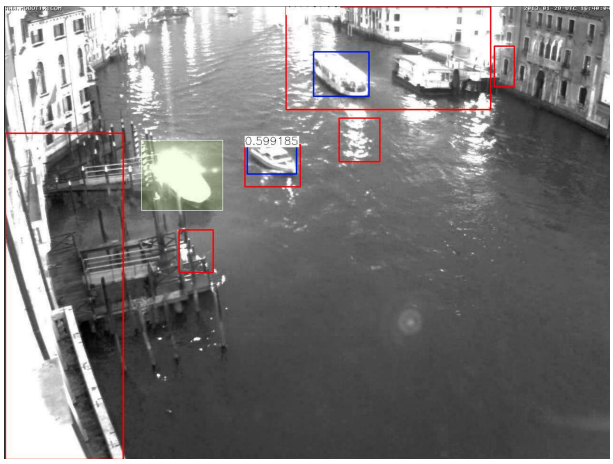


Figure 13: Illumination problems

To summarize what has been done, the performance of the algorithm has been exposed, in particular the problems that afflict it, obviously for convenience not all the results obtained on the test images have been reported as the cases exposed almost summarize everything. A single not indifferent factor was not exposed, namely the execution time. This algorithm cannot be used in real-time applications and does not even guarantee very high precision, as can be seen from the results. As already said, there are developments of this algorithm such as Faster R-CNN that aim eliminate the problems of the presented algorithm and improve it. The realization of the entire project took approximately 130 hours