Boat detector with HOG features, Selective Search Segmentation and SVM

Forin Paolo
University of Padua, DEI
Master Degree in Computer Engineering
Computer Vision course a.a. 2020-21

1 Introduction

Many paper talk about cars detection or people detection, but in this paper I show a simple boat detector using Histogram of Oriented Gradients features (HOG) [8], Selective Search segmentation (SS) and Support Vector Machine (SVM).
An important aspect is to understand that there are many types of boat and also waves or ribs that make this detection difficult. Therefore, some pre-processing is useful.

1.1 Report structure

The report is structured in many sections where we analyze:

- methodology and dataset;
- code;
- results obtained;
- conclusion.

2 Methodology and dataset

For the project I used two different dataset [1], [3] with some pre-processing to extract positive and negative samples, correspondingly boat and not boat.
Moreover, the test dataset is create using dataset [7] and other image taken from internet.
My approach is to train an SVM using the features extracted on positive and negative image. On the contrary, in test using Selective Search I extract a subimage (roi) and on this I extract features to classify it as boat or not boat using the trained SVM.
I decided to use HOG features because are good features for vehicle detection, plaques and other similar object with similar inter class shape [8].
I also proved another approach using SIFT and ORB as features, but with HOG I have better results [6]. Moreover, the idea to use Selective Search is due to these initial approaches.

2.1 Dataset

Given the two dataset [1] and [3] using the annotation tool provided by OpenCV [4], I extract the ground of truth of the images. This entails that all the ground of truth used are conform at this annotation:

”global path image” ”number boxes” ”top left x” ”top left y” ”width” ”height”
if there are more boxes, only the parameter of boxes are repeated, eg:
”global path image” 2 2 3 12 13 3 2 31 42
that means two truth boxes:

- top left corner coordinate = (2,3), width = 12, height = 13;
- top left corner coordinate = (3,2), width = 31, height = 42.

Now using the function **create_samples** that required the following parameters:

- path_file: path of the file with the ground of truth;
- path_dir_dataset: path of the directory of the dataset;
- boat_save_path: path where save the positive samples;
- no_boat_save_path: path where save negative samples;
- neg_width: min width of negative samples;
- neg_height: min height of negative samples;
- n_not_boat: min number of negative samples for image.

In this way I created the set of positive and negative images used to train the SVM. The positive image are exactly the boat on the image, instead the negative image are extracted using the metrics IoU (Section: 2.2) and selected as not boat if IoU is equal to 0.
For the project I use 600 positive and 942 negative samples, given by the datasets [1] and [3] where I also removed some image that are similar to another. Instead, for the test set I used [7] and other images founded on internet.

2.2 IoU

The metric chosen to evaluate a model is the Intersection over Union [9]:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{1}$$

where $Area\ of\ Overlap$ is the area of the intersection of the two rectangles and $Area\ of\ Union$ is the area of the union of the two rectangles. Moreover, to evaluate a model we need the ground of truth of the test set, so one of the two rectangles is a true box and the other is a predicted.
In my code is possible:

- to evaluate a model given the ground of truth of the test set (default command);
- to see only the predicted boxes (setting –only_detect=true).

2.3 Support Vector Machine

Support Vector Machine (SVM) is a supervised learning algorithm for classification or regression problem. Its goal is to find the best hyperplane that maximise the margin, which is the distance between the support vectors. Moreover, a sample is a support vector if its distance from the hyperplane is the smallest of its class.
In this case we use a linear soft SVM with default parameters except for "type" beacuse it must do a regression task. Therefore:

- Kernel: linear;
- Degree Q = 3;
- Type: EPS_SVR, because regression task;
- P factor: P = 0.1, because EPS_SVR;
- C factor: C = 0.01, because soft classifier;

2.4 Selective Search Segmentation

Selective Search segmentation is an algorithm that works on steps and return proposal regions where the object can be [2]. The steps are:

- Do image segmentation (often with Felzenszwalb method [5]);
- Recursively combine similar regions into larger ones. Greedy algorithm:
  - From set of regions, choose two that are most similar.
  - Combine them into a single, larger region.
  - Repeat until only one region remains.

  This yields a hierarchy of successively larger regions.
- Use the generated regions to produce candidate object locations.

There are many different way to calculate the similarity, and Selective Search uses 4 different measures based on color, texture, size and shape compatibility [10].

3 Code

In this section I report my code and comment every function what it does.

3.1 Library

The library that I implemented are utils and algorithm.

3.1.1 UTILS It is composed by 5 methods:

- **load_image**:
  - input: path of directory that contains negative and positive images, extension of negative and positive images, size of the image.
  - output: vector with all image and vector with all labels (1 if boat and -1 if not boat).

  At the end all images have same size and the order of the output vectors is important, because 1^{st} image has 1^{st} label, 2^{nd} image has 2^{nd} label and so on.
- **convert2ml**:
  - input: vector of matrix with one row or one column. All matrix have the same number of features (they are 1xn or nx1).
  - output: single matrix composed by all matrix passed. The number of rows are the number of matrix in input, the number of column are the number of features.

  With this method I can merge the features extracted in the training set and then I obtain a matrix in the correct form to train an SVM with the flag ROW_SAMPLES.
- **compute_iou**:
  - input: two rectangles.
  - output: IoU of the rectangles passed.

  This method is important to create negative images and to evaluate models.
- **file_to_vectorOfLines**:
  - input: path of the ground of truth.
  - output: vector with all name.extension of the images and vector with all true boxes.

  The order of the output vectors is important, because 1^{st} name.extension has 1^{st} vector of true boxes, 2^{nd} name.extension has 2^{nd} vector of true boxes and so on.
- **create_samples**: already seen on section 2.1.

3.1.2 ALGORITHM It is composed by one class with 7 methods. I choose to use one class because may be in the future I will implement another algorithm for object detection. The methods are:

- **boat_detector**: It is the constructor.
- **nms**:
  - input: vector of rectangles, vector of scores of the rectangles, a threshold, the number of neighbors and the minimum score sum.
  - output: vector of rectangles that satisfied the parameters and their predicted scores.

  With this methods I remove some predicted rectangles and I keep only the strongest.
- **extract_hog**:
  - input: windows size for the hog detector and the vector of images from which extract HOG features.
  - output: vector of gradients.

  Also here is important the order, indeed the features of the 1^{st} image are the 1^{st} on the vector gradients, and so on. Moreover, the size passed for the hog detector is the same size that the image must has, in this way I have all features vector with same size.
- **pre_processing**:
  - input: image to pre-process.

– output: image pre-processed.

The pre processed image is a edges image.

- **check_rect**:
  - input: image.
  - output: true or false.

  It return true if in the image passed there are at least 25 points not black. I used it with the image returned by pre_processing to remove rectangles given by Selective Search in order to have less false positive.
- **train_SVM**:
  - input: vector containing all features, vector with all labels and the path where to save the model.
  - output: save the model trained.

  In this methods I train an SVM, to do so I call the function convert2ml above defined.
- **test**:
  - input: path to the model, path to the test directory, extension of the test images, size for hog features, path to the ground of truth of the test images, boolean value to show the IoU or not.
  - output: show 4 (true) or 2 (false) images, depends on the boolean value.

  If the boolean value is true it show:

  1. image with all rectangles predicted with score grater than $0,5$ and their score.
  2. image with all rectangles that remain after performing non maximum suppression (nms defined on section 3.1) and their score.
  3. image with all true boxes extracted by the ground of truth file.
  4. image with all rectangles that remain after nms and their IoU calculate with respect to the boxes extracted by the ground of truth file.

  If the boolean value is false it show only the images numbers 1 and 2.

3.2 Command line input

The project work only if there are inputs elements, they are:

- *help*: show all the commands.
- *–pos_dir=*: string. Path of the directory with positive samples.
- *–neg_dir=*: string. Path of the directory with negative samples.
- *–test_dir=*: string. Path of the directory with test samples.
- *–gt_test=*: string. Path of the ground of truth of the test images.
- *–gt_train=*: string. Path of the ground of truth of the train images (we use it for create positive and negative samples).
- *–type_neg=*: string. Extension of negative images (default: .jpg).
- *–type_pos=*: string. Extension of positive images (default: .jpg).
- *–type_test=*: string. Extension of test images (default: .jpg).
- *–only_test=*: boolean. Test a trained detector with ground of truth of test images (default: true).
- *–only_detect=*: boolean. Test a trained detector without ground of truth of test images (default: false).
- *–create_sample=*: boolean. Create samples given other parameters that are asked after (default: false). The other parameter asked are:
  - path to the directory with the dataset;
  - minimum width, height and maximum number of negative images extracted from one image.
- *–train_test=*: boolean. Train a new model and test it, for the testing are allowed the value on only_detect and only_test (default: false).
- *–model_path=*: string. Path to the model (default: ../svm.yml).

example of command line input:

./boat_detector –train_test=true –pos_dir=/data/boat
–neg_dir=/data/not_boat –gt_test=/test.txt
–test_dir=/data/test

Last useful consideration to run the code in the correct way:

- one file for the ground of truth for the dataset or test images (you can use annotation tool [4] to create the ground of truth file);
- on MacBook it works properly, but in Linux the function to calculate time does not work, a simple solution is to comment them in functions load_image, train_SVM and test.

4 Results

In this section I show the results that I got. First of all I show the original image and the pre processed one. The following figures have this structure:

left image the input image.

right image the output image.

As mentioned above, the pre processing return an edges image, in this way I made a trade off because I want to see the boat but not the waves (see Figure 1 and Figure 2). Moreover, many images have buildings or clouds or other object that I was unable to eliminate (see Figure 3). Therefore I did a trade off on edges detection, in this way I eliminated some boat but also some building edges (see Figure 2).

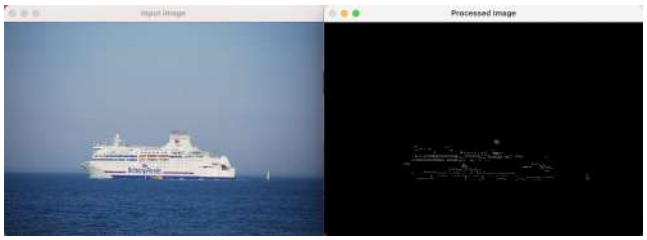


Figure 1: pre processing image with one boat

Figure 2: pre processing image with boats and buildings

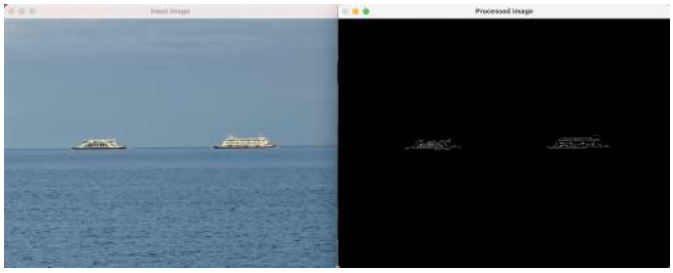


Figure 3: pre processing image with two boats

At least I show the results for some the test images. The following figures have the same structure:

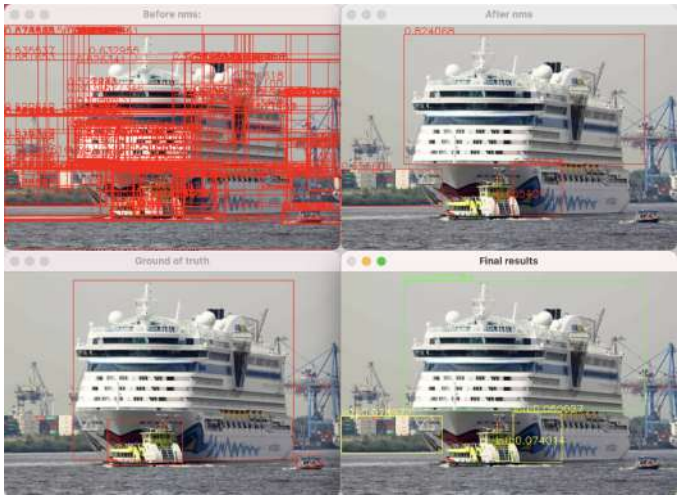
Top left image all boxes predicted with probability to be a boat greater than 0.5.

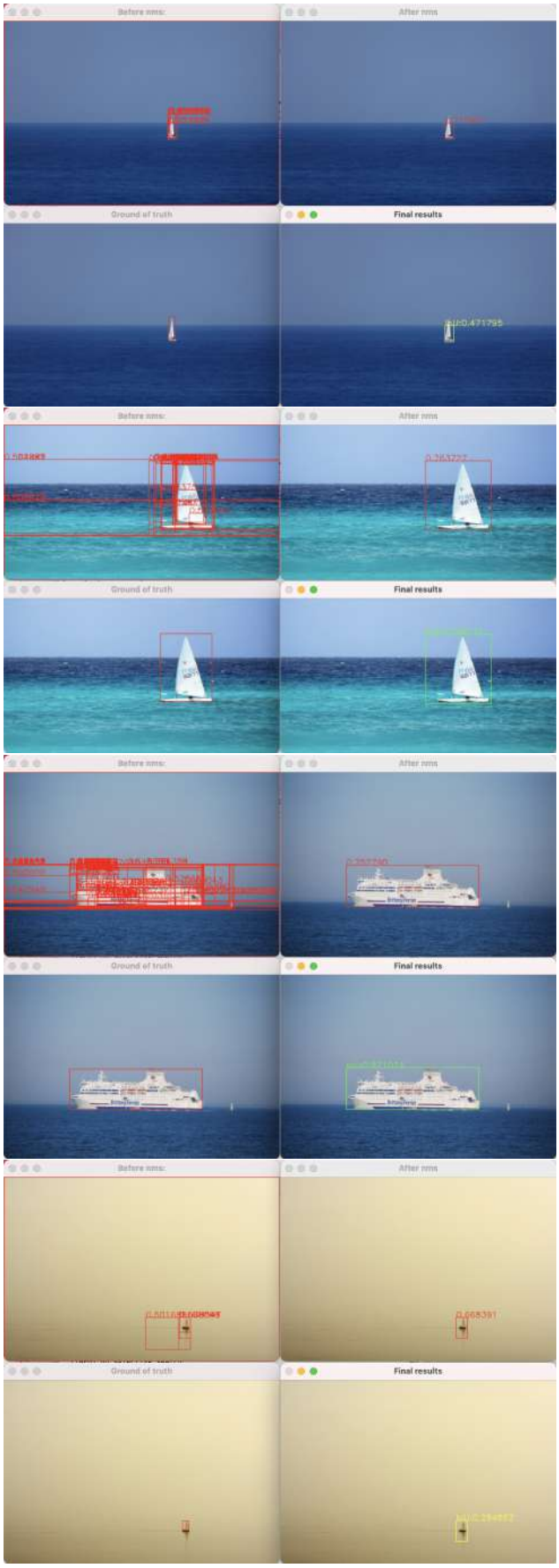
Top right image boxes that remain after applied non maximum suppression.

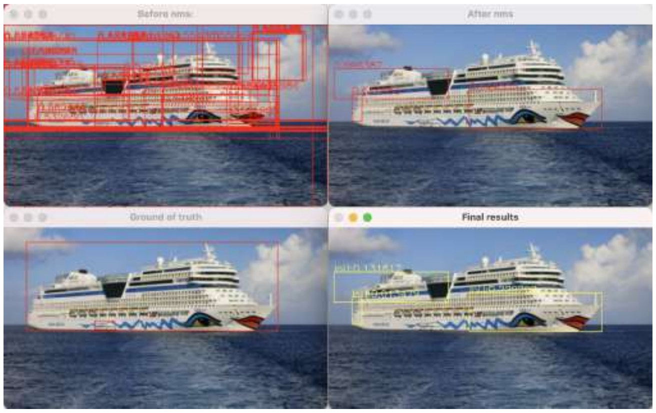
bottom left image all true boxes given by the ground of truth of the image.

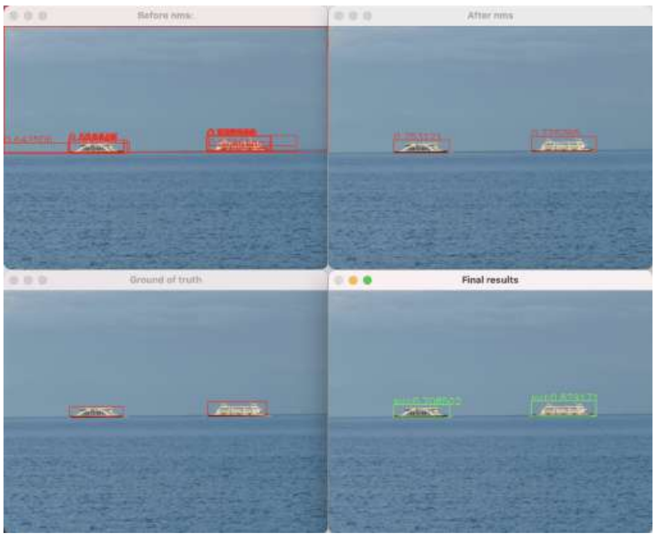
bottom right image IoU between top right image and bottom left image.

If the parameter *–only_detect=* is true, the images shown are only the top one, instead in default case are shown all 4 images.

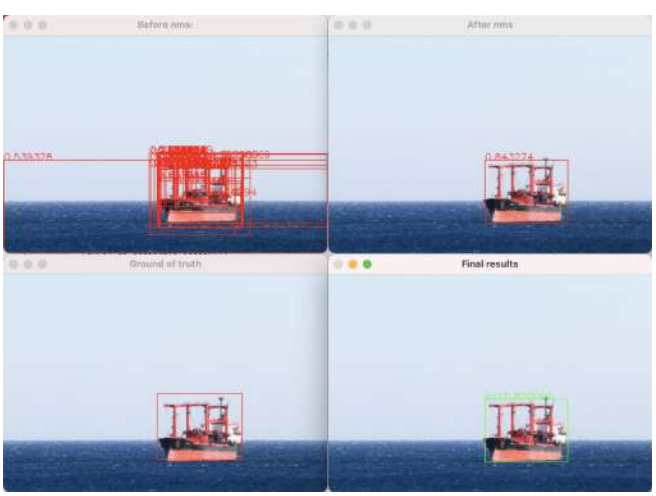


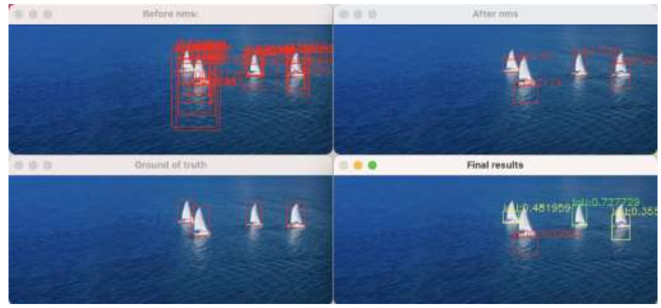


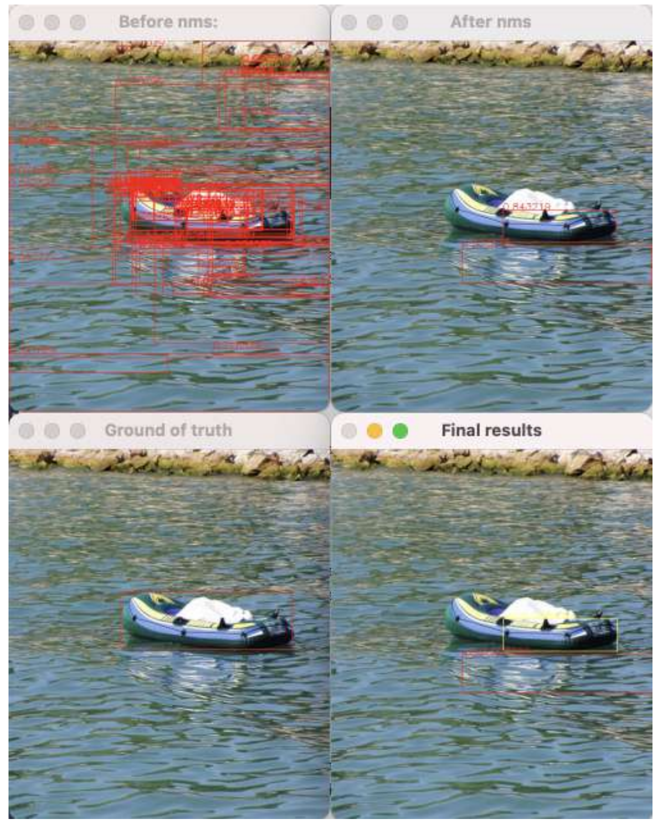

Before nms:
After nms
Ground of truth
Final results

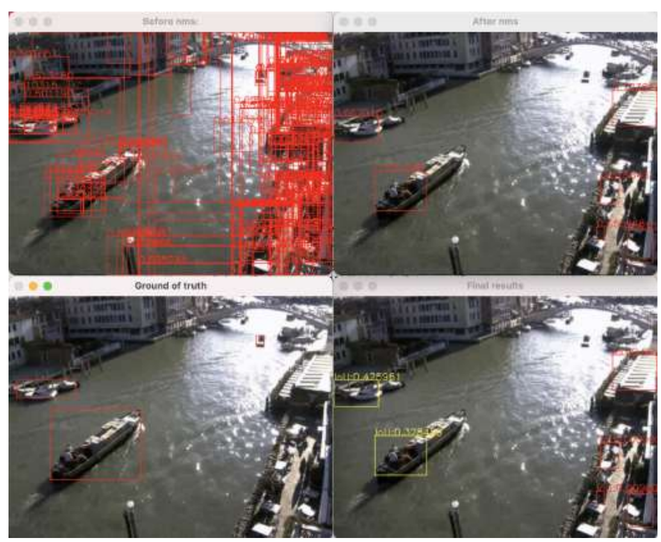

Before nms:
After nms
Ground of truth
Final results


Before nms:
After nms
Ground of truth
Final results

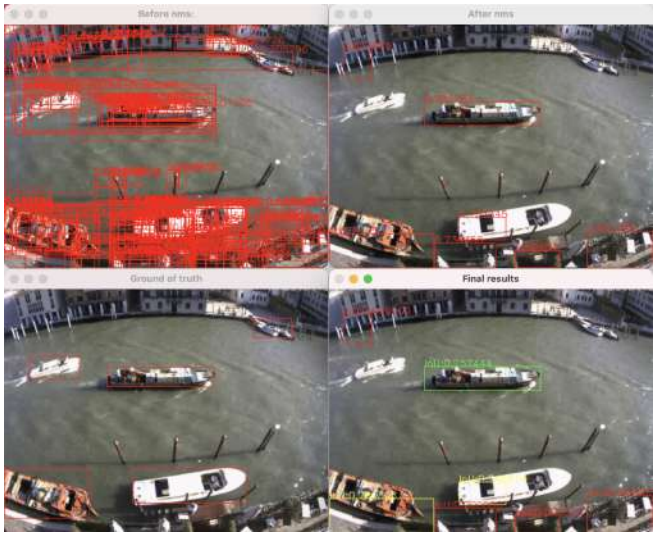

Before nms:
After nms
Ground of truth
Final results

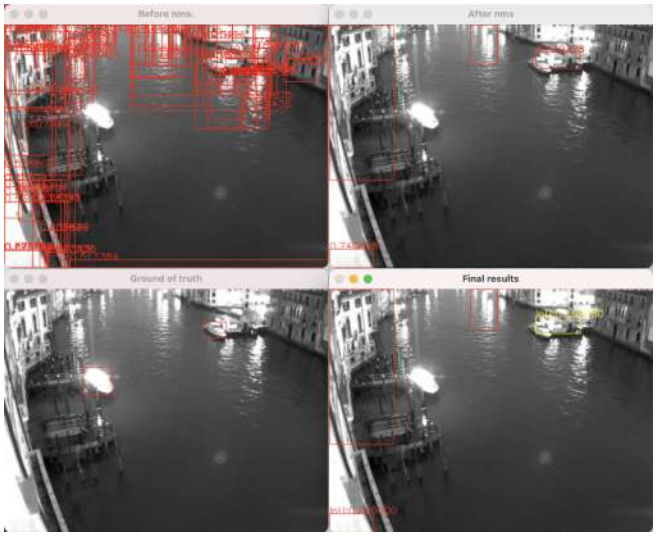

Before nms:
After nms
Ground of truth
Final results


Before nms:
After nms
Ground of truth
Final results

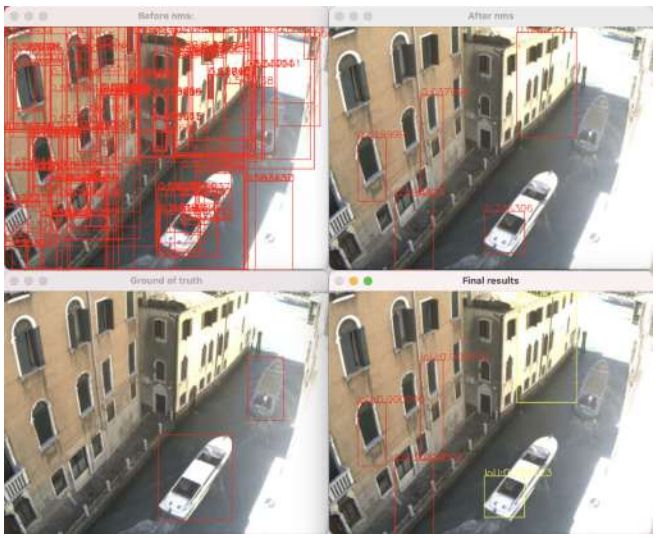

Before nms:
After nms
Ground of truth
Final results

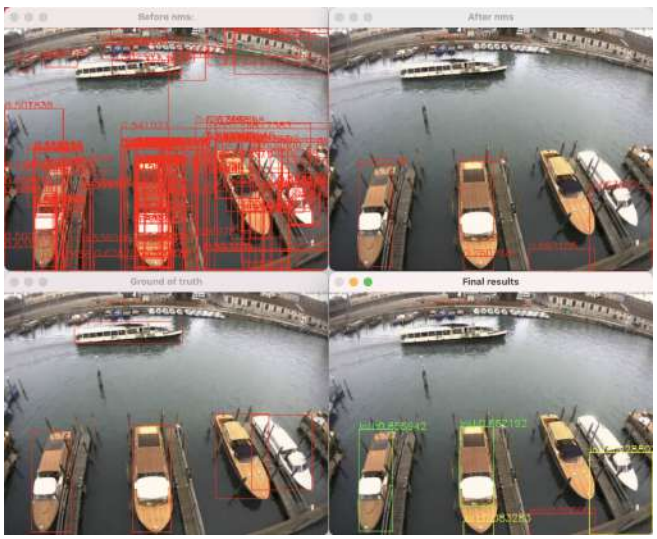

Before nms:
After nms
Ground of truth
Final results

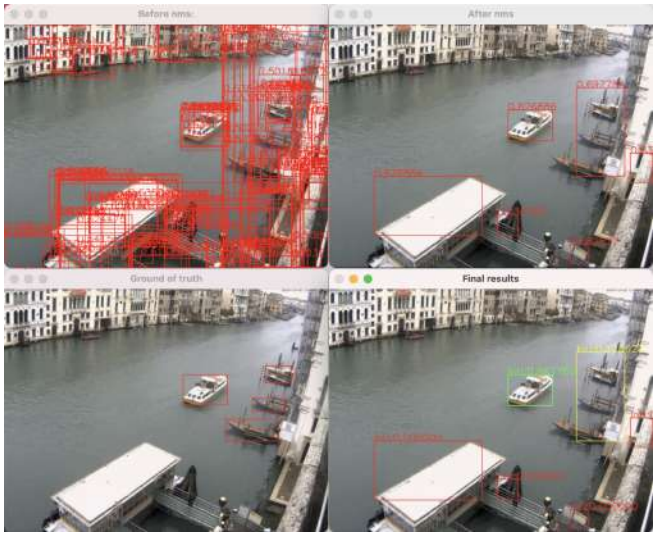

Before nms:
After nms
Ground of truth
Final results

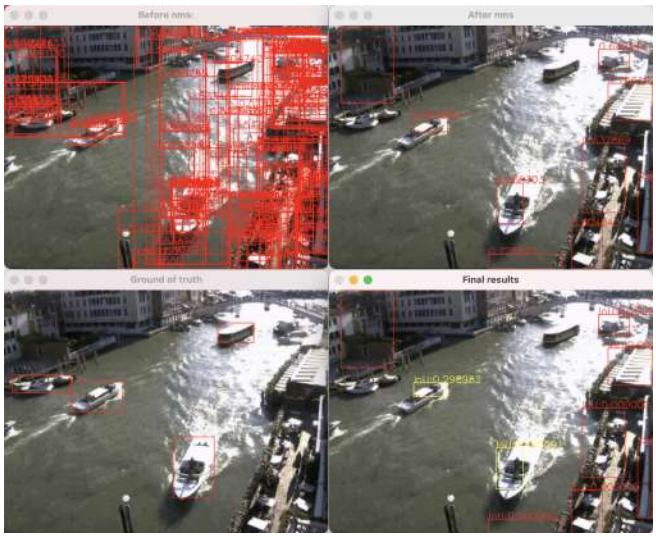

Before nms:
After nms
Ground of truth
Final results

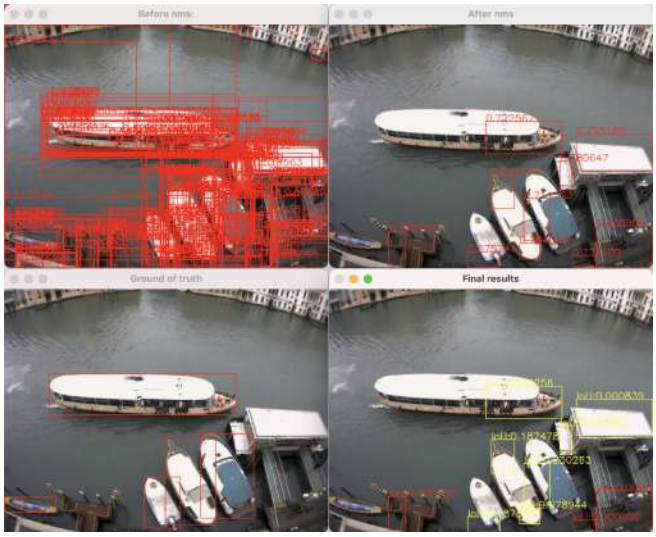

Before nms:
After nms
Ground of truth
Final results

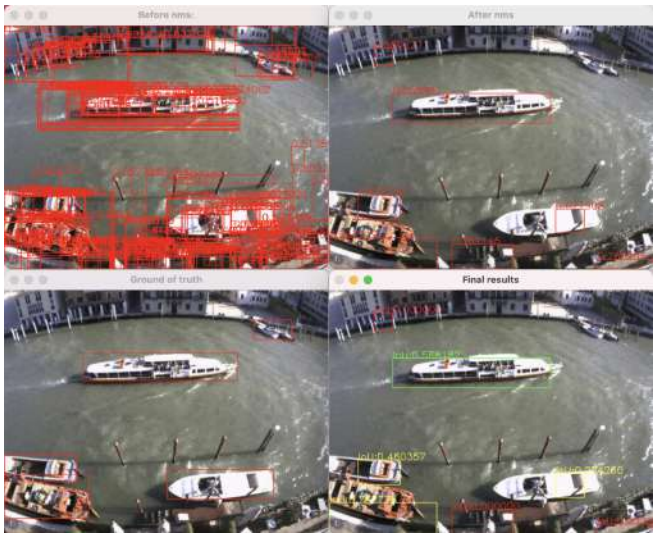

Before nms:
After nms
Ground of truth
Final results

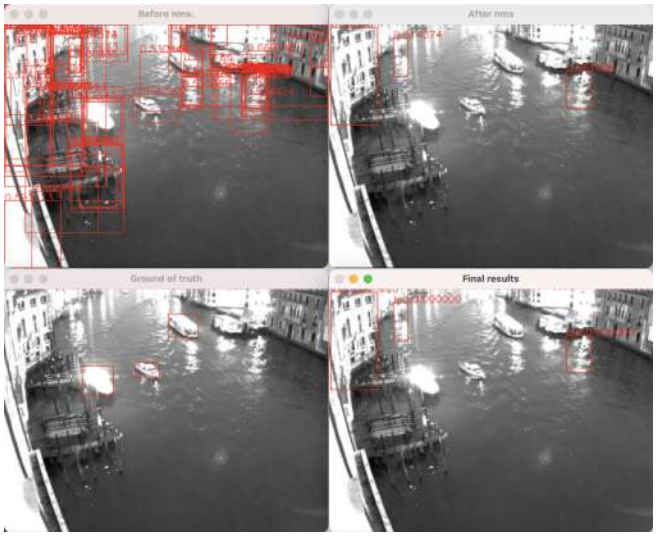

Before nms:
After nms
Ground of truth
Final results


Before nms:
After nms
Ground of truth
Final results


Before nms:
After nms
Ground of truth
Final results


Before nms:
After nms
Ground of truth
Final results


Before nms:
After nms
Ground of truth
Final results


Before nms:
After nms
Ground of truth
Final results

Moreover, the code printed out some useful information. In Figure 4 I show what it printed with parameter defined in the example on section 3.2.

```
[INFO] start load images...
[INFO] number of positive images: 600
[INFO] number of negative images: 942
[INFO] total number of images: 1542
[TIME] time to load all images: 0.974619s

[INFO] train svm......[done]
[TIME] time to train svm: 15.9524s

[INFO] saving svm......[done]

[INFO] start testing
[INFO] load model......[done]
[INFO] extracting ground of truth of test images......[done]
[INFO] processing image 1 of 42
  [INFO] do selective search...
  [INFO] rects skipped: 55 of 464
  [TIME] time to process image: 0.620192s

[INFO] processing image 2 of 42
  [INFO] do selective search...
  [INFO] rects skipped: 185 of 1597
  [TIME] time to process image: 1.55498s

[INFO] processing image 3 of 42
  [INFO] do selective search...
  [INFO] rects skipped: 100 of 1358
  [TIME] time to process image: 1.38105s
```

Figure 4: information printed

(Note: to see the edges images, you need to put a cv::imshow() and cv::waitKey(0) on the method pre_processing).

5 Conclusion

In conclusion, this approach work well in some case and not very well on other case. As we can see in section 4, this approach is good with image in BGR color space that are not overexposed, without (or with few) buildings and with the boat not in foreground. There limitations are due to how I extract the roi, because using selective search and HOG I deformed every rectangle to a square and after I calculate HOG. Moreover, I know that HOG can works well with sliding windows approach, but I have better results with Segmentation Search.
With this project I have found difficulties on what pipeline follow, because I also tried with CNN, but importing the model on C++ was very difficult and sometimes did not work. Furthermore, using Selective Search with ORB or SIFT works bad, because I had to do a resize on the gradients.

References

[1] Bloisi, D. D.; Iocchi, L.; Pennisi, A.; and Tombolini, L. 2015. ARGOS-Venice Boat Classification. URL http://www.diag.uniroma1.it//~labrococo/MAR/classification.htm.

[2] Chandel, V. S. 2017. Selective Search for Object Detection (C++ / Python) URL https://learnopencv.com/selective-search-for-object-detection-cpp-python/. Last Updated on september 18, 2017.

[3] Clorichel. 2018. Boat types recognition. URL https://www.kaggle.com/clorichel/boat-types-recognition/version/1.

[4] Docs, O. -. URL https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html.

[5] Felzenszwalb, P. F.; and Huttenlocher, D. P. 2004. Efficient Graph-Based Image Segmentation .

[6] Mallick, S. 2016. Histogram of Oriented Gradients explained using OpenCV URL https://learnopencv.com/histogram-of-oriented-gradients/. Last Updated on december 6, 2016.

[7] MAR. -. URL https://drive.google.com/file/d/1kCgOFIP7meuUDh49BYxGyTYTg-kJQwys/view.

[8] Mittal, K. 2020. A Gentle Introduction Into The Histogram Of Oriented Gradients URL https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa. Last Updated on december 21, 2020.

[9] Rosebrock, A. 2021. Intersection over Union (IoU) for object detection URL https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. Last Updated on July 1, 2021.

[10] Uijlings, J. R. R.; and Gevers, T. 2013. Selective Search for Object Recognition URL https://www.researchgate.net/publication/262270555.