Computer Vision Final Project Report

Riccardo Cescon, 2026247

July 23, 2021

1 Introduction

The goal of this project is to detect boats in images. There are several approaches seen during the course, and not only, that can be used to tackle this task. I am fully aware that state-of-the-art methods rely on CNN well trained on huge datasets but the method I will adopt does not make use of any NN-based classifier.

2 Method

I based my approach on the paper *Selective Search for Object Recognition* by J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, A.W.M. Smeulders.
In the paper the authors presented an alternative algorithm to the sliding window approach to find possible region proposals to be categorized. They also provided a learning pipeline to train a model able to detect objects out of the proposals given by their Selective Search.
Since the implementation of this algorithm provided by **OpenCV** for some images was not able to give any region proposal I added also other two techniques to improve the performances. One is a cascade classifier and the other is a Canny-based edge detector. Finally, to reduce the number of false positives detected I adopted a SVM classifier which makes use of the HOG features which was another possible implementation, alternative to the SIFT features, provided by the aformentioned paper.
The ground truths used in this project to train the models are taken from the one kindly shared by some course mates. They created a text file for each image representing the boat top left and bottom right corners. An example can be: *boat:263;1106;30;733;*. For some images they also provided the coordinates for some *hidden boats* but I did not take into account them since they are only small pieces of boat in the images' margins.

3 Implementation

3.1 Selective Search + SVM classifier

Starting from the paper, I implemented a classifier based on the Selective Search. The Selective Search is used to propose some regions of interest in the image while to detect if a region is boat or not I used a Bag of Visual Words (BoW) framework to train a classifier. The selective search algorithm is already implemented in **OpenCV**. I used it in the class Searcher in which there is the method **segmenter** which given an image returns the region proposed by the algorithm. Since Selective Search for most of the images produces a lot of regions of interest I decided to remove the ones with area smaller than 10% of the image's one because it was suggested in the paper and its also pretty reasonable since the smaller the object the harder is to detect it and it is also very unlikely to have a very small object in the image. Since the algorithm returns the bounding boxes ordered accordingly to a probability of being interesting I decided to take only the first 1000 rectangles proposed. As far as the features used in the paper, they adopted SIFT descriptors along with Extended OpponentSIFT and RGB-SIFT. Since in **OpenCV** only SIFT is implemented I went for a simpler version without considering the other

two. To generate the codewords I made use of 50 representative images and I extracted the SIFT descriptors from keypoints (the pipeline for this point was the same as the one of Lab5). I tried also an alternative to the use of the library method **detect** to find keypoints that is to extract descriptors in a dense grid of 20x20. This was in my opinion a better way to create the vocabulary but somehow my implementation gave worst results than simply extracting keypoints and the training process was of course slower. Some examples of images used to train a vocabulary are presented in Fig. 1.

(a) image0010.png (b) image0780.png (c) image0905.png

Figure 1: Some examples of images from the venice dataset used to create the vocabulary.

I then made use of the library class **BOWKMeansTrainer** which given a set of descriptors adopt KMeans algorithm to cluster them and find the centers corresponding to the vocabulary's codewords. I empirically found that there is not much difference in the vocabulary size provided that it is sufficiently representative, say more than 250 codewords. In this project I used $k = 1000$. Once the vocabulary is formed I needed to train a model, in my case a SVM, to classify the new descriptors extracted from a testing image. In fact from a new image we extract the descriptors that are quantized in a vector of dimension 1000, using the vocabulary trained before. Every entry of this vector represents the frequency of finding a feature that is present also in the vocabulary. Based on those frequencies the classifier is able to say whether there is a boat or not.
To train the SVM I used the **OpenCV** class of Machine Learning. I set the Kernel to **RBF** and the type to **ONE CLASS** since we need to detect only boats. The dataset used was the union of the positive examples extracted using the ground truths coming from Venice and Kaggle.
The class I developed to do this task is **ObjectRecognition**. It provides the methods:

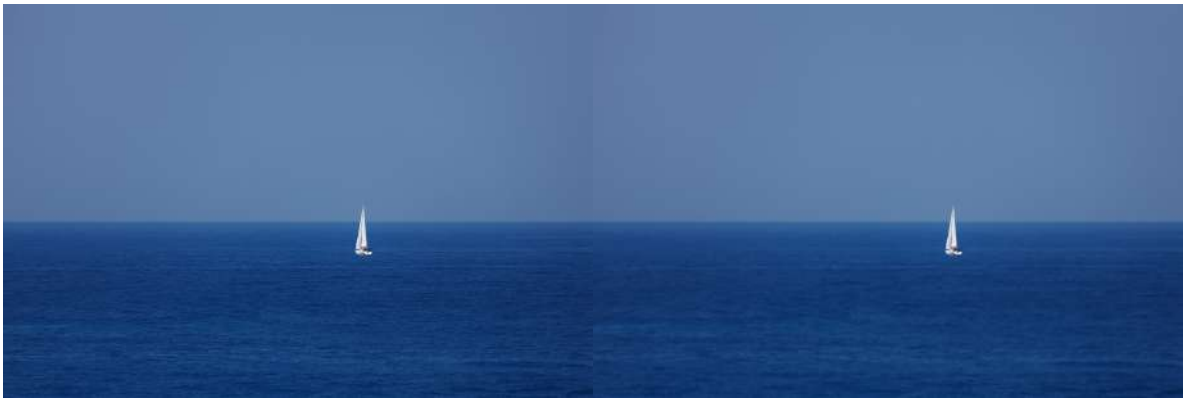
- **createVocabulary**: function which creates the vocabulary implementing the BoW approach described before, if it is not already trained and loaded from file.
- **train**: function that trains the SVM model, if it is not already trained, using the BoW image description coming from the vocabulary. In the train process to optimize the parameters involved I adopted a k-fold validation using $k = 5$.
- **predict**: function which given a new image detects the keypoints and quantize them in an histogram using the BoW and finally predict if it is a boat or not using the SVM classifier.
- **loadImage**: function used to load the images needed for the train process.

3.2 Cascade Classifier

This part was added to have more region proposals since the Selective Search was missing some boats. To train a cascade I used the executable train_cascade available with the **OpenCV 3** version. To run the executable I needed a file containing the negative images' paths and another text file representing the positive images' paths and the corresponding boats' number and location. The cascade made use of Haar features as in the Viola-Jones face detector.

3.3 Canny edge detector

As already pointed out, to add more bounding box proposals I used also Canny Edge detector. The main idea comes from the **OpenCV** tutorial *Creating Bounding boxes and circles for contours* with of course some adjustments. In particular since the testing images were pretty noisy I added first a Bilateral Filter which during the course we have seen to be the best possible choice. Another preprocessing made was an adaptive equalization using the library class **CLAHE** which is shown to



(a) Original image (b) Processed image

Figure 2: In the figure we can see the original image and its processed version. We can see how the filter blurs the image removing noise while at the same time the equalization is able to enlarge the contrast.

be more effective than a global equalization, removing possible loss in details due to over-brightness.
The filtering was not mandatory in the BoW framework but surely it does not compromise results but, along with the **CLAHE** equalization, it is very important in the edge detection. In particular if we do not filter the image lots of small boxes would appear due to the heavy noise present in it. Enhancing the contrast helps also to better find the contours of objects. An example of pre-processing on a test image can be seen in Fig. 2.
The main work done in this part is an edge extraction using Canny. After that we save the contours using the library function **findContours** and we use them to interpolate a polygon that is finally approximated as a bounding box. Not all bounding boxes are good because there are some that come out due to the presence of noise not completely filtered out or because of some small edge of objects we are not interested in. To remove these outliers I adopted a static threshold which removes all the bounding boxes whose area is smaller than the 20% than the maximum area found.
The function that does the entire procedure all in one is **edgeExtractor** from the class **Searcher** I mentioned earlier.

3.4 HOG feature classifier

Finally, since the pipeline described up to now produced a lot of false positives I implemented another final classifier to deal with them and reduce their number. This classifier is again a SVM trained with a lot of images of non-boat (more than 20k) and the same positive images used to train the previous classifier of the BoW. As an input it takes the HOG descriptor of a resized version of the image (48x48) which accounts for the gradients' orientations on a fixed grid.
My implementation of this is the class **HogClassifier** which has the following methods:

- **HOG_Feature**: computes the HOG descriptor of the given image.
- **train**: train a SVM classifier, if it is not already found from file, using a **RBF** kernel and C_SVC as type since we are classifying two different classes. Again I adopted k-fold validation ($k = 3$).
- **constructLabels**: computes the labels given the set of examples.
- **predict**: given a new image computes the relative HOG descriptor and predicts its label.
- **loadImage**: loads images from a file.

The main drawback of using the implementation of SVM provided by **OpenCV** is that it returns a prediction label denoting the class but it is not possible to obtain a confidence value on such prediction.

To overcome this problem I tried to get also a raw value setting to **RAW_OUTPUT** the parameter in the **predict** method. In such case the output represents the distance of the point from the margin found by the support vectors. I then looked at the maximum distance and divided all the positive labels by the maximum trying in some sense to normalize and obtain a probability. In this way I was able to use the function **NMSBoxes** of the dnn module, which performs a non-maxima suppression of the bounding boxes classified as boat.

3.5 Performance evaluation

To evaluate the performances made by my boat detector I developed the function **computeIoUMax** which computes the Intersection over Union (IoU) between every bounding box of the ground truth and each predicted rectangle and save the maximum value obtained (the IoU is computed using the function **computeIoU**). Then I average all these values (one for each predicted bounding box) obtaining an overall result for the entire image.

3.6 Results

3.6.1 Kaggle dataset

In the Kaggle dataset my approach performed good but not excellent since I am not using a state-of-the-art model and the first part of the selective search in most of the images did not produce interesting region of interest to classify with the BoW. This can be a drawback and lack of the **OpenCV** implementation of the algorithm.

3.6.2 Image 01

Figure 3: IoU: 0.43. The algorithm is able to find the cruise even if not with a unique bounding box. The other two small boats are instead missed.

3.6.3 Image 02

Figure 4: IoU: 0.27. The algorithm is able to find the small boat. The bounding box is however not covering it entirely.

3.6.4 Image 03

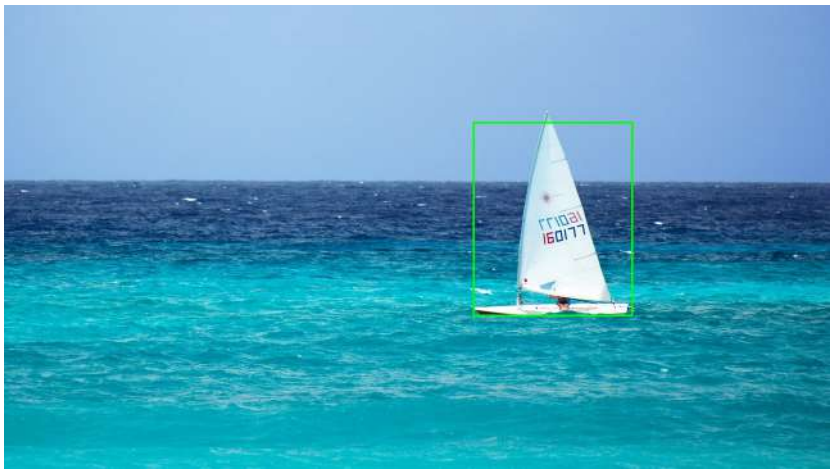


Figure 5: IoU: 0.89. In this case we have a very good boat detection.

3.6.5 Image 04

Figure 6: IoU: 0.95. In this case we have a very good boat detection.

3.6.6 Image 05

Figure 7: IoU: 0.0. In this case we miss completely the boat.

3.6.7 Image 06

Figure 8: IoU: 0.19. Also in this case the detection is not effective. We can only detect a portion of the boat.

3.6.8 Image 07

Figure 9: IoU: 0.5. In this case the detection is pretty good since the boat are both found, but the bounding boxes are not perfectly surrounding them.

3.6.9 Image 08

Figure 10: IoU: 0.39. In this case there are only two of the three boats detected.

3.6.10 Image 09

Figure 11: IoU: 0.48. The boat is detected, unfortunately the selective search algorithm proposed also a region of interest with water.

3.6.11 Image 10

Figure 12: IoU: 0.48. The rubber boat is detected quite well.

3.7 Venice dataset

The detection of this dataset was quite hard compared to the previous one in which the algorithm performed not so bad. The main reasons in my opinion are firstly the number of boat per figure which are more in this dataset and secondly the quality of images. Indeed there are a lot of small boats in the background in some of the following images that in my opinion are almost impossible to detect with a non-DeepLearning approach in which the number and the quality of features are sensibly increased by a well-trained NN.

3.7.1 Image 01

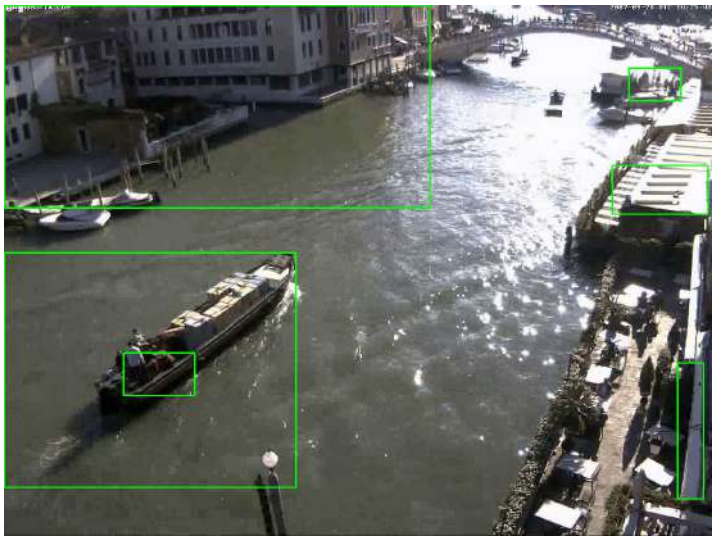


Figure 13: IoU: 0.12. Lots of false positives.

3.7.2 Image 02

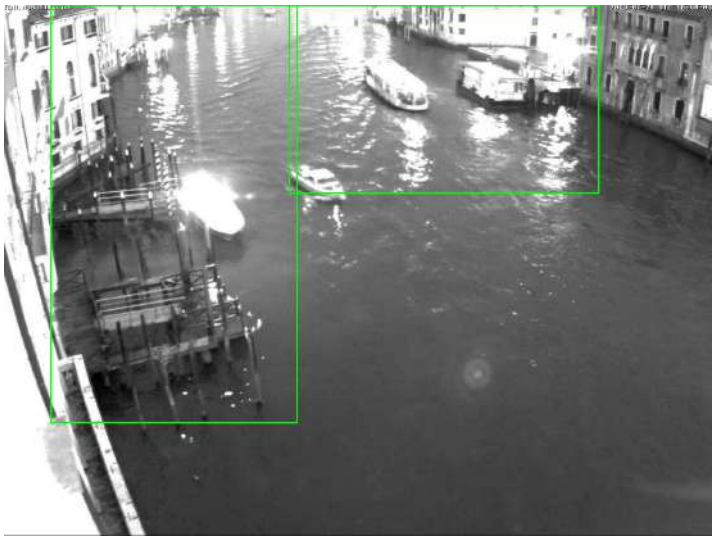


Figure 14: IoU: 0.03. In this case it can detect the boats but the bounding boxes are very large including also non-boat elements.

3.7.3 Image 03

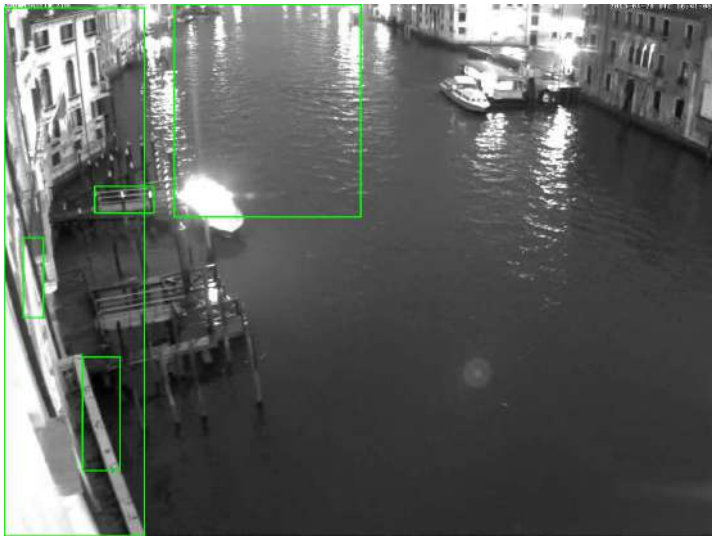


Figure 15: IoU: 0.04. In this case there are false positives.

3.7.4 Image 04

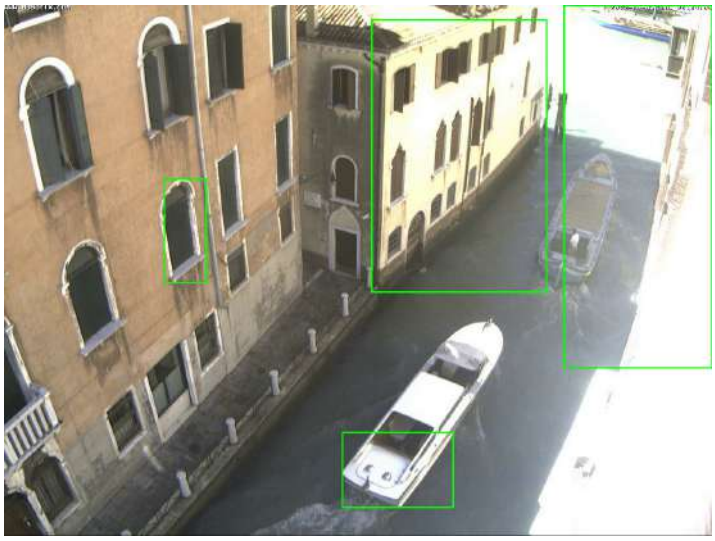


Figure 16: IoU: 0.2. In this case there are false positives.

3.7.5 Image 05

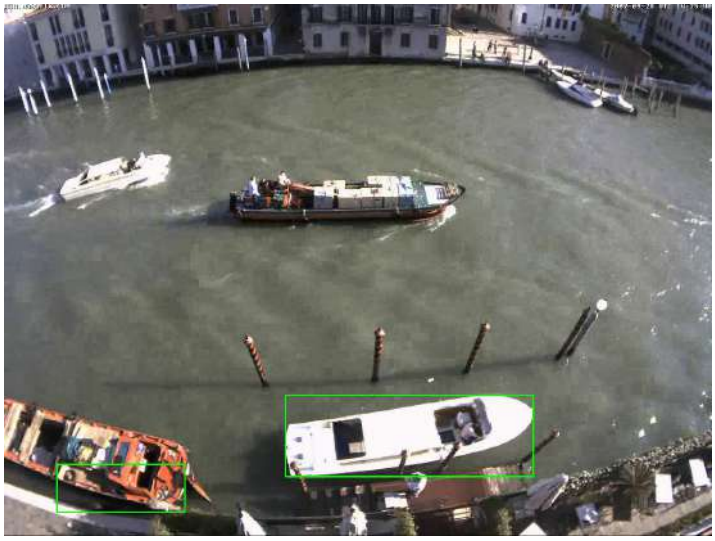


Figure 17: IoU: 0.19. In this case there are some false negatives.

3.7.6 Image 06

Figure 18: IoU: 0.09. Poor results because there are some false positives as well as false negatives.

3.7.7 Image 07

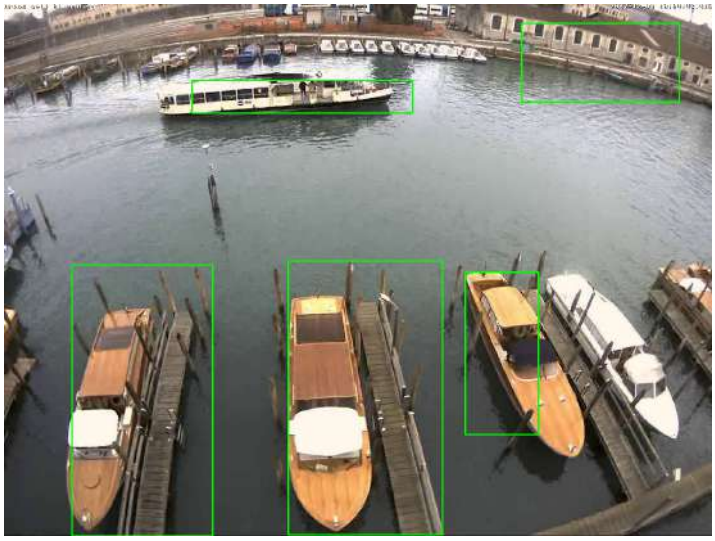


Figure 19: IoU: 0.08. The detection in this case is not bad, though it finds a false positive and the small surrounding boats are not considered.

3.7.8 Image 08

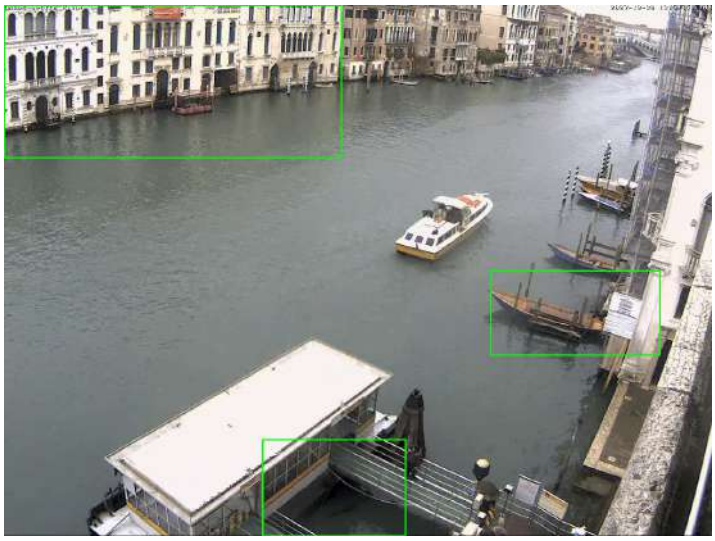


Figure 20: IoU: 0.09. In this case it cannot detect the principal boat and it finds two false positives.

3.7.9 Image 09

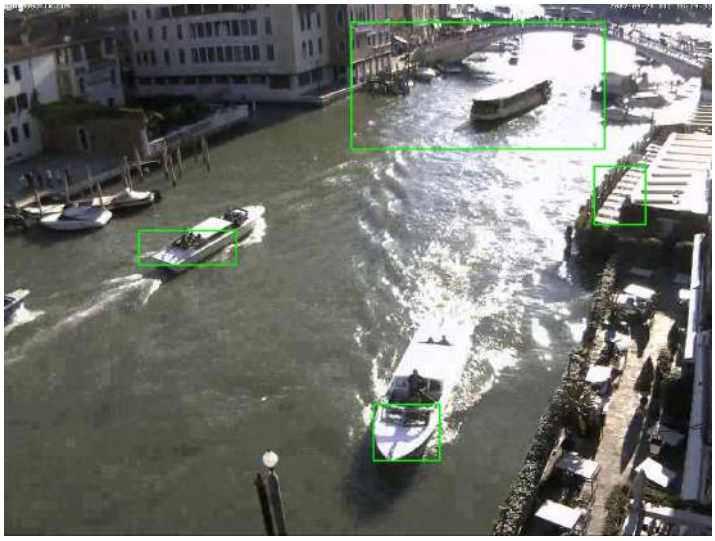


Figure 21: IoU: 0.14. The boats are more or less detected while at the same time there is a false positive.

3.7.10 Image 10

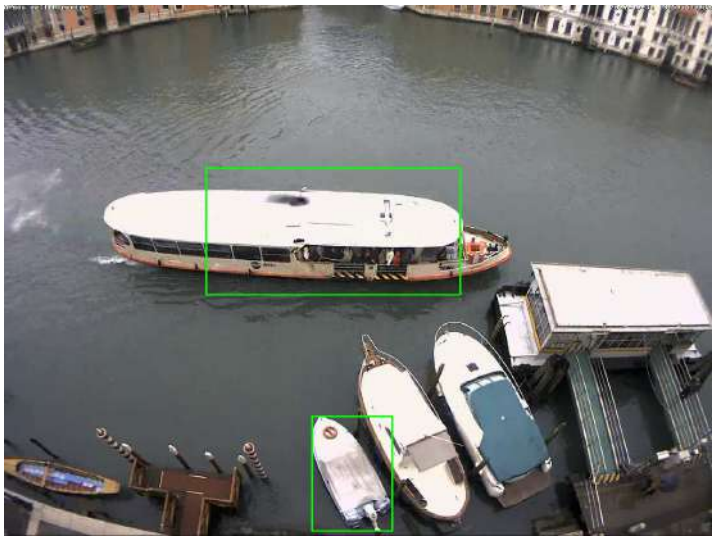


Figure 22: IoU: 0.27. It can detect two out of four boats.

3.7.11 Image 11

Figure 23: IoU: 0.11. There are some false negatives.

3.7.12 Image 12

Figure 24: IoU: 0.05. In this case the algorithm cannot distinguish between the two boats.

3.8 Conclusions

The project made me understand better and implement the framework of the BoW. I also learned other techniques not seen during the course such as the Selective Search and the HOG features which are an alternative way to represent the content of an image, useful when we want to detect objects. The main disadvantages of this implementation were the not high performances reached especially in the Venice dataset where the images were very difficult to detect. Secondly the **OpenCV** framework is not effective when dealing with Machine Learning. For example it does not provide the time required to train a model and often times this process has been very time consuming. Moreover the documentation provided is quite always not explanatory at all which translates in time lost to understand how to make things work. Finally it does not have all the useful methods that for example are available with *scikit-learn*.
As future work I would definitely try to implement a more powerful classifier, since I realized that the final HOG-based classifier is able to remove a lot of false positives but eliminates also a lot of bounding boxes that are correctly classified by the BoW. To do so, it is needed a lot of time to train a classifier on more and representative examples.