Computer Vision Final Project - Boat Detection

Davide Allegro Gioel Adriano Vencato

July 2021

1 README

On command line, in order to execute the program perform the following steps:

1. Go to Final_project_Vencato_Allegro directory
2. mkdir build
3. cd build
4. cmake ..
5. make
6. Then execute: ./Final_project ../data/Kaggle_dataset ../data/Venice_dataset ../model/model_CNN.pb ../data/Ground_truth_segmentation

2 Introduction

The main aim of this project was to develop an automated system for:

- Boat Detection
- Sea Segmentation

For these purposes we have immediately thought to adopt an approach as generic as possible, in order to obtain good results for a wide and different dataset of images.
For the first assignment we have implemented a Convolutional Neural Network from scratch combined with a sliding window approach. More precisely we have adopted a CNN because it is well known for its performance on image classification. Furthermore it was fed by region proposal generated by the sliding window.
As far as the second assignment is concerned, a general approach was not an easy task, due to the big difference between the two provided datasets:

- For the Kaggle dataset we have adopted Kmeans color segmentation combined with hough lines transform and morphological operators with the exception of the last image where Otsu thresholding method was applied.
- For the venice dataset it is used Otsu thresholding method combined with morphological operators of erosion and dilation.

In the following sections, the introduced approaches are described more accurately. For the design of the whole project, both students of the group gave an important contribution at each step, in terms of ideas and practice implementation. The whole implementation required more or less 20 full days, 8-10 hours/day.

3 Workflow Pyhton

As it is introduced previously, for the boat detection it was planned to use a machine learning method. For the implementation of the Convolutional Neural network Network (CNN) it was adopted the Colab Notebook, in order to exploit Google resources.
The CNN was implemented from scratch, by using the module *Keras* of the library *TensorFlow*. The creation of the network was done throught the *Sequential()* model, necessary to stacking all the layers of the whole CNN.
More precisely the Convolutional Neural Network was structured in order to receive as input, images of size 300x300. It is composed by the following layers:

- First Convolutional Layer with 32 - 3x3 filters;
- Max Pooling Layer with a 2x2 pooling window;
- Second Convolutional Layer with 32 - 3x3 filters;
- Max Pooling Layer with a 2x2 pooling window;
- Third Convolutional Layer with 64 - 3x3 filters;
- Max Pooling Layer with a 2x2 pooling window;
- Flatten layer
- Dense Layer with 64 units
- Dropout Layer to overcome the problem of overfitting
- Dense Layer with 1 units which gives the output

Then the CNN was trained with the created Dataset (4.1), exploiting the binary cross entropy which computes cross-entropy loss between true labels and predicted labels; and the Adam optimizer with a learning rate equal to 0.001.
In Fig. 1 the loss and the accuracy obtained during the training's epochs are reported.

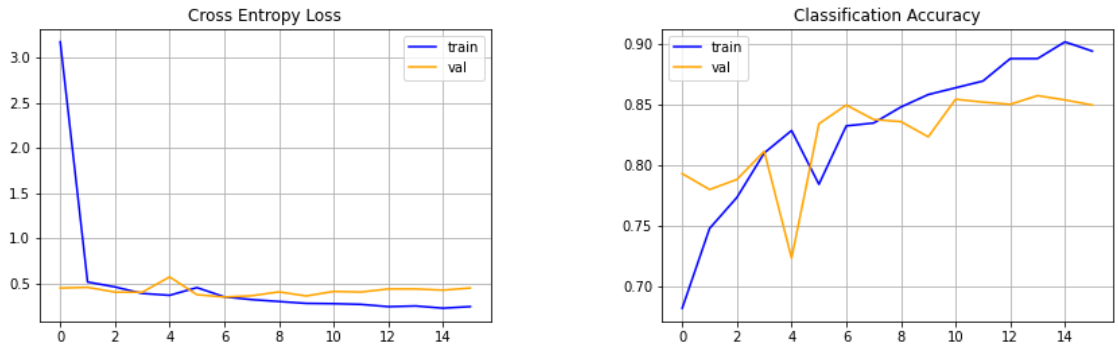


Figure 1: Loss and Accuracy of the model

The model was then freezed and saved on the file model_CNN_pb, which was exported on C++ for the computation of boat detection.

4 Workflow C++

In order to tackle both the detection and segmentation parts, a multi-class program structure was implemented. More precisely the following classes were created:

- Dataset
- Base
- Detection
- Segmentation
- HoughLine

4.1 Dataset

Since a Machine Learning approach was adopted, an ad hoc dataset was necessary to be found. At first, the convolutional neural network was trained with the two provided datasets:

- http://www.dis.uniroma1.it/ labrococo/MAR/classification.htm
- https://www.kaggle.com/clorichel/boat-types-recognition/version/1

The found neural network was tested on the sub-images generated by the sliding window. However the outcoming results were not acceptable, since a lot of images were misclassified. This problem was due to the fact that the CNN was trained on a completely different dataset images, where it was possible to recognize an entire boat. Instead, when the sliding window scanned a boat, the generated sub-images framed only parts of it and hence the CNN was not able to classify them as boat. For this reason, thanks to the sliding window, the aforementioned class was created to generate a whole new dataset composed by cut images from a qualitative representive of the two datasets mentioned previously..
Once the $Boat$ and $NoBoat$ folders were filled by the corresponding cut images, by hand, it was necessary to carefully look at and eventually move them to the correct folder.

In particular, the Dataset class consists of only the *Constructor* ***Dataset****(String images_path, String model_CNN_path)*, where *images_path* was the specific folder path in which the selected images were stored, while *model_CNN_path* was needed to load the trained CNN.

4.2 Base

The Base class, then exploited by the Detection and Segmentation classes, is characterized by the *Constructor* ***Base****(String pattern, int fl)*. In particular:

- *pattern*: path in which the test images, those needed for performance evaluation, either from Venice or Kaggle datasets, are stored.
- *fl: it is set to 0 if Venice images are loaded otherwise it is needed to be set to 1.*

4.3 Detection

The Detection class was structured as follows:

Constructor

- ***Detection****(String images_path, String model_CNN_path)*

Methods

- ***void detection****(vector<Mat> image, String model_CNN_pb)*
  The sliding window is implemented by this method, which receives as input the images to be analyzed *image* and the path of CNN that has to be loaded *model_CNN_pb*. Moreover, alternatively to this method, the multi scale pyramid approach was tried to be implemented, in which, once the window scanned the whole image, the scanning of the resized version could be performed, so that the detection could result scale invariant. However, the first approach was preferred, for its improvements on the accuracy of the detection.
  More precisely, this is an iterative method, which works with the already trained CNN. After fixing the stepsize and the sliding window's size, each sub-images is provided as input to the CNN which gives as output *1* if it is classified as a *Boat*, *0* otherwise. Then each rectangle correspondent to a Boat with a probability higher then 0.8 was stored in *vector<Rect> all_rects* that was necessary for the method *rect_return(...)*. Since with this approach, most of the time, a boat was identified by more rectangles, the openCV function *groupRectangles()* was adopted in order to get only one seed rectangle for each boat. It clusters all the input rectangles using the rectangle equivalence criteria that combines rectangles with similar sizes and similar locations.

Figure 2: Result of groupRectangles: on the left all sub-images identified as *Boat* by the CNN; on the right the seed rectangles

- ***vector<Rect> rect_return****(vector<Rect> all_rects, vector<Rect> seed_rects, int scaling)*
  This method is necessary for the final bounding box representation, which consists in merging all the rectangles which belong to the same seed rectangle. It receives as input *all_rects* which is composed by all rects identified as *Boat* by the CNN in the previous

method; *seed_rects* is characterized by the seed rectangle generated by *groupRectangles()*; *scaling* is a parameter which set how the rectangle merging has to be done. More precisely, this method allows to merge all the rectangles in *all_rects* which intersect a specific seed rect, in order to get a final rectangle that ideally represent the bounding box of each boat of the image. In the Fig 3 below the final result is reported:

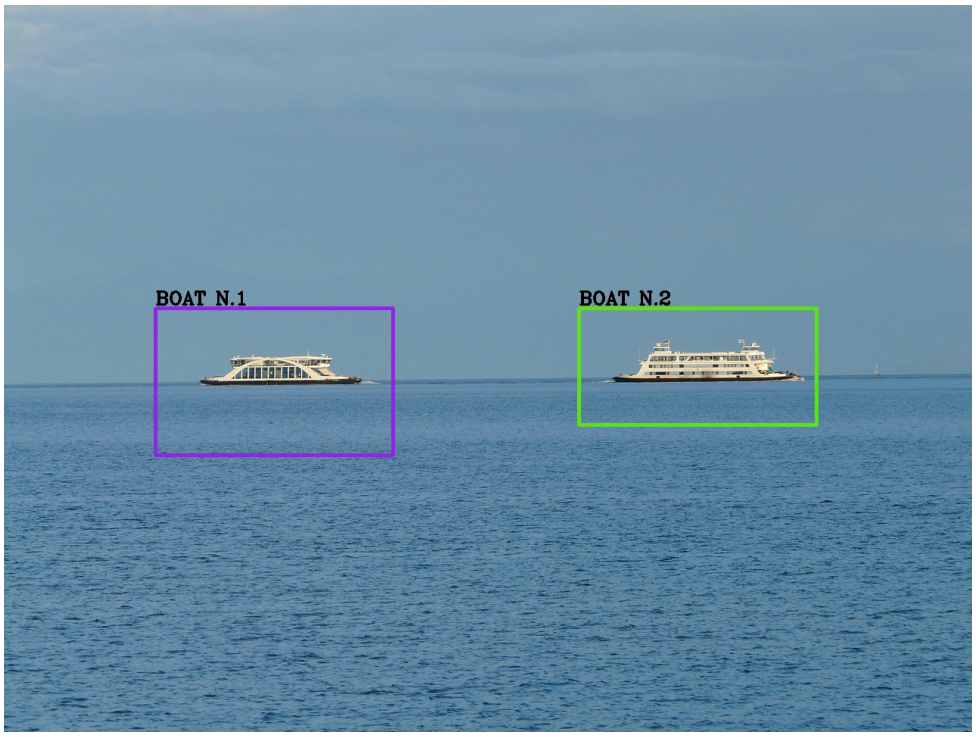


Figure 3: Final predicted bounding boxes

- ***void getMetrics****(vector<Rect> predicted_rects, Mat image_predicted, int index)*
  This method was implemented for the final detection evaluation of the automated system. The test was done, by exploiting the ground truth collected by some of the class groups reported inside the current method.
  The accuracy is measured. by Intersection over Union metric (IoU). It evaluates the performance of the predicted bounding box by adopting the following equation:

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

  where *Area of Overlap* corresponds to the area of the rectangle generated by the intersection of Ground-Truth bounding box and Predicted bounding box, instead *Area of Union* is given by the union of them. Moreover, if the bounding box identifies as *Boat* an object *No Boat*, it is categorized as false positive. In Fig. 4 the boat image with the predicted bounding boxes (random colors) and Ground truth bounding boxes (white color) is reported.

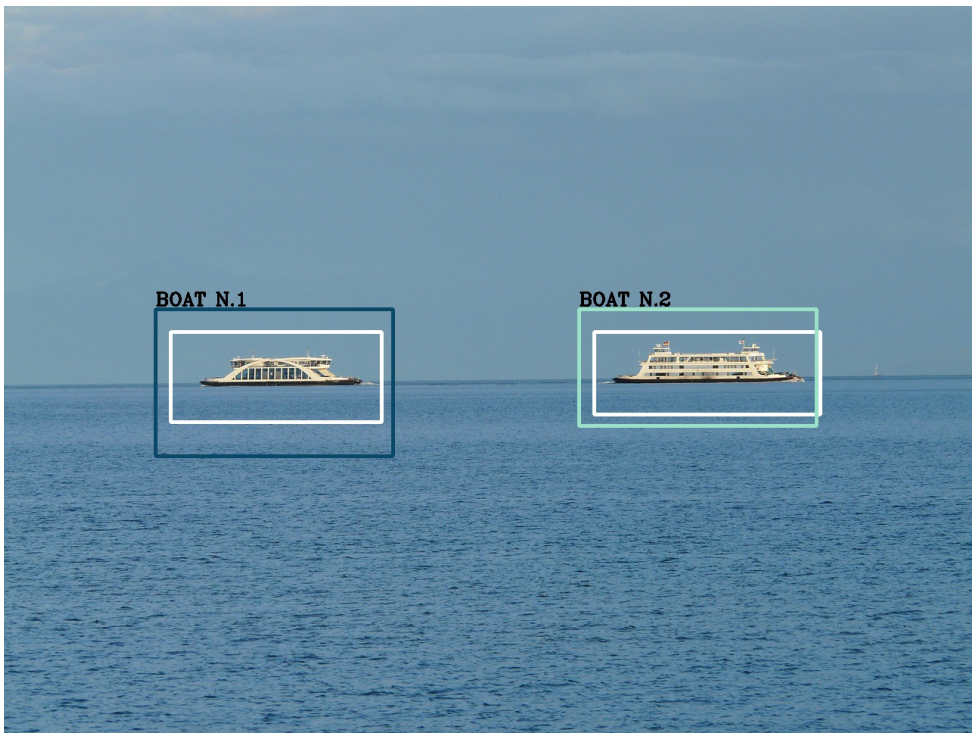


Figure 4: Ground truth bounding boxes & predicted bounding boxes

4.4 Segmentation

The Segmentation class is composed by:
Constructor

- ***Segmentation****(String images_path, String model_CNN_path)*

Methods

- ***void click****(Mat image)*
  This method was necessary to obtain the ground-truth segmentation images for each image of the 2 provided test datasets. It was decided to assign the labels "sea", "not sea" as follows: a rectangle shape was drawn by clicking 2 times in the interested image and that portion was colored in white, representing sea pixels. All the other pixels were set to black, representing not sea pixels.

- ***void onMouse****(int event, int x, int y, int f, void* userdata)*
  Callback function used by click method.

- ***void segmentation****(String ground_truth_segmentation_path)*
  Once the ground-truth segmentation was collected in the apposite folders, the corresponding path *ground_truth_segmentation_path* could be given to the method.
  Depending on which dataset is considered, different approaches have been tried out. More specifically, as far the Kaggle dataset is concerned, an approach as much general as possible was attempted to be obtained. A combination of the following techniques was implemented in all the Kaggle images with the exception of the last images in which an other techinque, i.e Otsu method, showed to be working better.
  For instance, the left part of Fig. 5 is considered. After the Gaussian filter is applied, color Kmeans clustering techique (K=2) was performed in order to assign to all the pixels a specific label, based only on the color information. The result is shown in the right part of the same figure. By applying only the clustering, perfect sea segmentation

was not achieved and hence it was needed to furthermore process the image. More precisely, Canny edge and Hough line detectors, where in this case a specific class was created, were exploited to find the line that best separates the sea from the remaining part. The found line can be observed in the left part of Fig. 6. It was then possible to consider separately 2 portions of the same image, separated by the found line. By noticing that the sea was mapped always to a darker color with respect to the other cluster, the openCV function *dilate* was used in the top sub-image in order to map all the misclassified pixels to the correct cluster. Similarly, the function *erode* was used in the bottom part. Since this image processing could wrongly misclassify pixels belonging to the sea, the morphological operator *dilate* was used again. The number of iterations of each mentioned operator was different while the choice of the structuring element was the same.

Figure 5: Caption

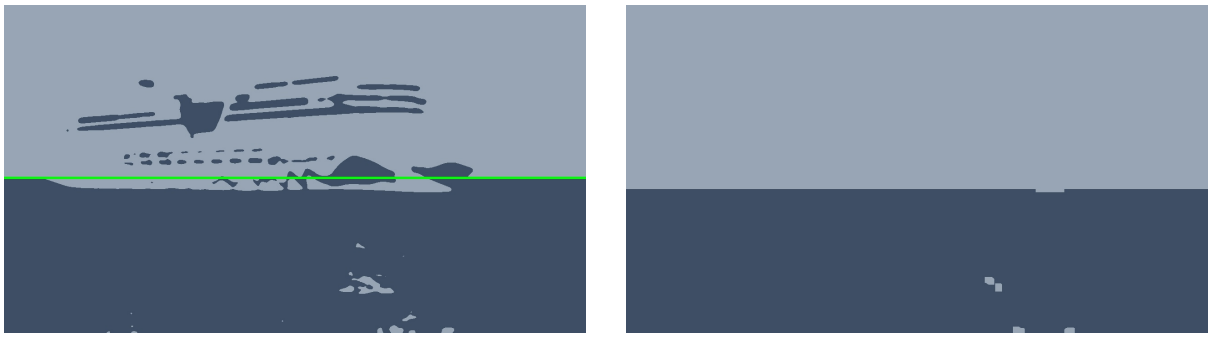


Figure 6: Caption

- ***double getMetrics****(String ground_truth_images, Mat segmentated_images, int index)*
  This method returns the pixel accuracy that quantifies the quality of the segmentation. It is called inside the *segmentation* method for any image. For this reason it only requires, beside the path *ground_truth_images*, the segmented image and parameter *index* so that it is able to compare the segmented image *segmented_image* with the index-th image stored in the path. The comparison consists in evaluating the *intersection* over *union* metric IoU, where, considering the ground-truth and segmented images, *intersection* represents the number of all pixels labelled as Sea in both the 2 images, while *union* represents the number of those labelled as Sea in at least one of the 2 images. The resulting metrics are reported in the Results section.

- ***void swap_colors****(Mat& image)*
  Since the segmentation ground-truth collection was done considering as *sea* the white

pixels and as *not sea* the black ones, while the 2 colors provided by the combination of techniques described in the segmentation method were swapped, this function simply swaps the order.

4.5 HoughLine

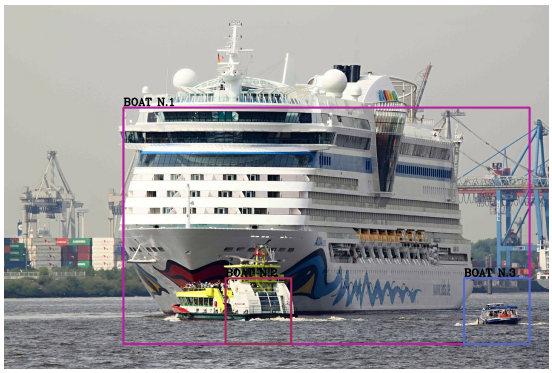
This class is exploited by the segmentation method. The constructor receives the image to be analyzed as input, i.e. the output of the Canny edge detector, and the necessary parameters. The method *doAlgorithm* performs the Hough line algorithm.

5 Boat Detection results

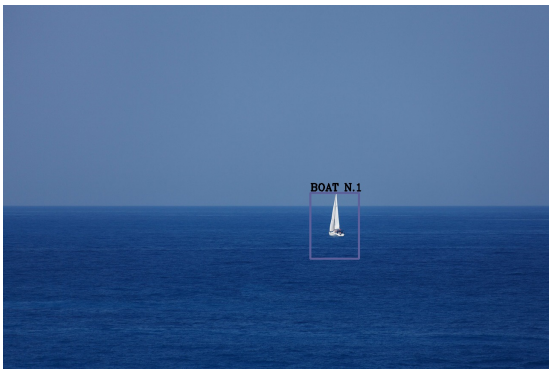
In the next subsection are reported the results obtained by the implemented boat detector, distinguishing the Kaggle dataset and Venice dataset.

5.1 Kaggle dataset

For the images of this dataset, any image pre-processing was used. The CNN was able to detect correctly the boat on the images, and to create the bounding box accurately.
Moreover it is interesting to observe that for the smaller boats (Fig. 7(b),7(e),7(g),7(h)), the predicted bounding boxes overestimated a little bit the real area of the boat. This is due to dimension of the sliding window which is chosen as a trade off, in order to get acceptable results with boats of different sizes. If the sliding window had been chosen smaller, on little boat better results would have been obtained but the performance on bigger ships would have been decreased, since the CNN would not been able to recognize a small part of a big cruise. Furthermore, in Tab.5.1 are reported the metrics IoU obtained for each image.

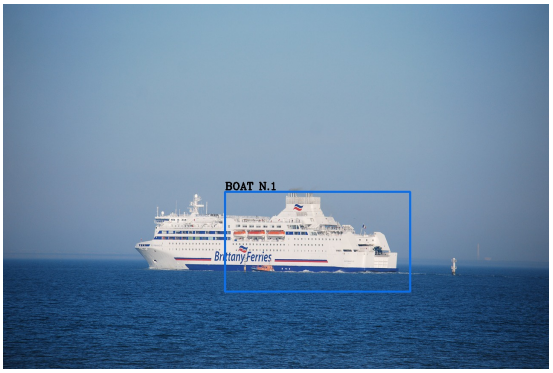


((a))

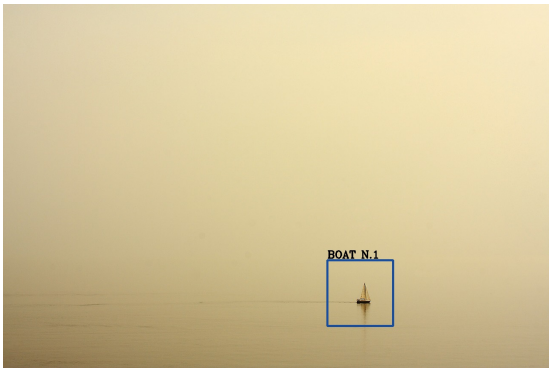


((b))

((c))



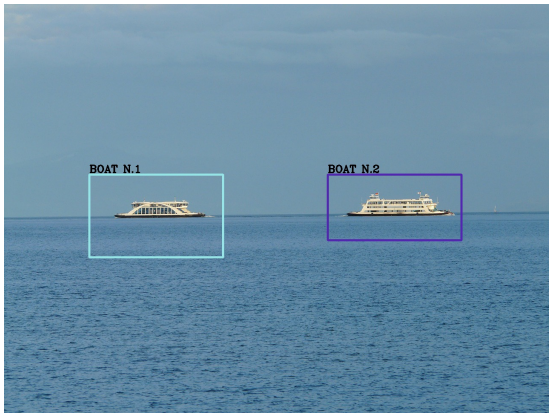
((d))


BOAT N.1

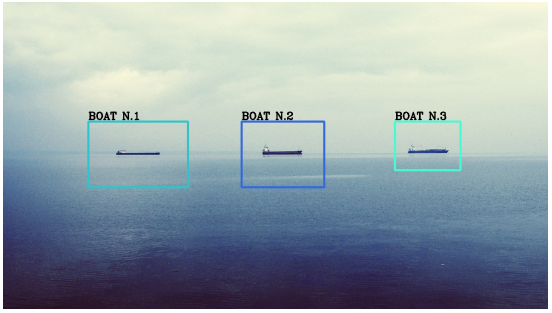
((e))


BOAT N.1

((f))


BOAT N.1
BOAT N.2

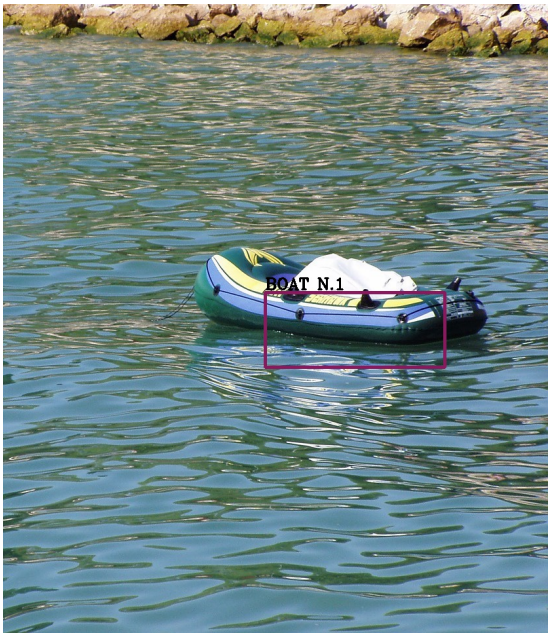
((g))


BOAT N.1
BOAT N.2
BOAT N.3

((h))


BOAT N.1

((i))


BOAT N.1

((j))

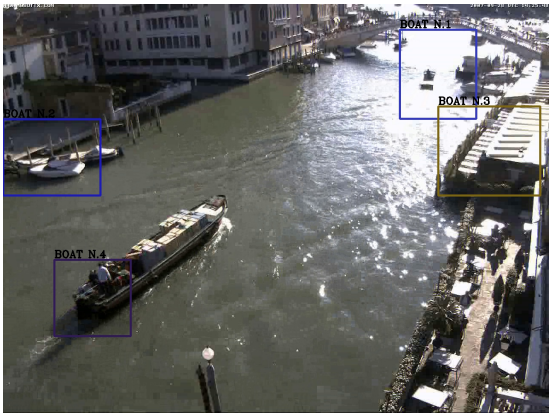
KAGGLE (IoU)	Boat n.1	Boat n.2	Boat n.3	#False positive
Image 0 (Fig.7(a))	0.674	0.480	0.649	0
Image 1 (Fig.7(b))	0.716	-	-	0
Image 2 (Fig.7(c))	0.289	-	-	0
Image 3 (Fig.7(d))	0.626	-	-	0
Image 4 (Fig.7(e))	0.503	-	-	0
Image 5 (Fig.7(f))	0.786	-	-	0
Image 6 (Fig.7(g))	0.547	0.657	-	0
Image 7 (Fig.7(h))	0.494	0.595	0.601	0
Image 8 (Fig.7(i))	0.582	-	-	0
Image 9 (Fig.7(j))	0.401	-	-	0

It can be observed that all the boats are detected correctly, and even accurately because most of the metrics IoU are higher than 0.5. It means that the CNN works very well with this kind of images, and if it cooperates with the sliding window approach, they give an accurate boat detector. The parameters of the sliding window adopted for this dataset are:

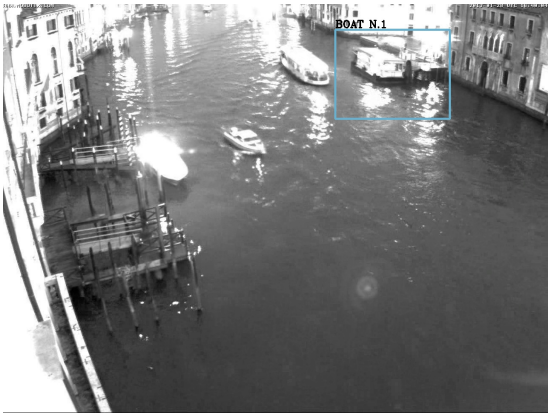
- stepSize $= 40$;
- windowsRows $= 115$;
- windowsCols $= 115$;

5.2 Venice dataset

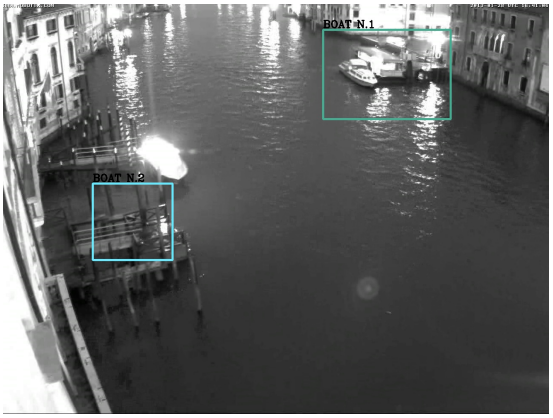
As far as the boat detection on Venice dataset is concerned, boat detection was not an easy task, due to the more complex environment in which the boats are located. In order to get better results and acceptable performances, even if lower than in the previous dataset, an image pre-processing was performed. First of all, each RGB image of the Venice dataset was converted to HSV image, and the channel V has been equalized. It represents the values of the lightness of the pixel. Then, each equalized image was converted back to RGB image and it was smoothed by adopting a Gaussian smoothing filter of size 3x3. Hence the obtained images are given as input to the whole system sliding window/CNN. In the next images are shown the results obtained.



((k))



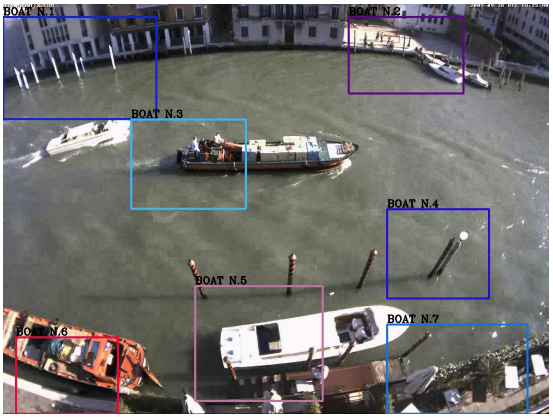
((l))


BOAT N.1
BOAT N.2

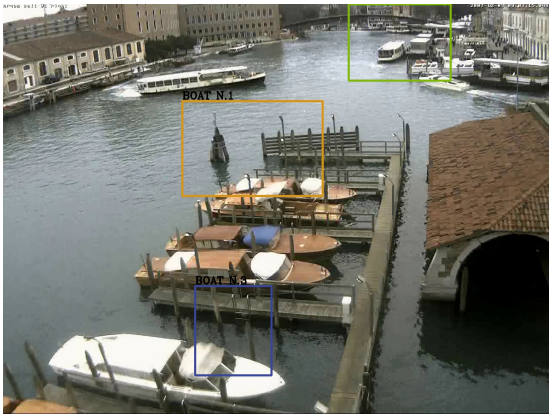
((m))


BOAT N.1
BOAT N.2

((n))


BOAT N.1
BOAT N.2
BOAT N.3
BOAT N.4
BOAT N.5
BOAT N.7

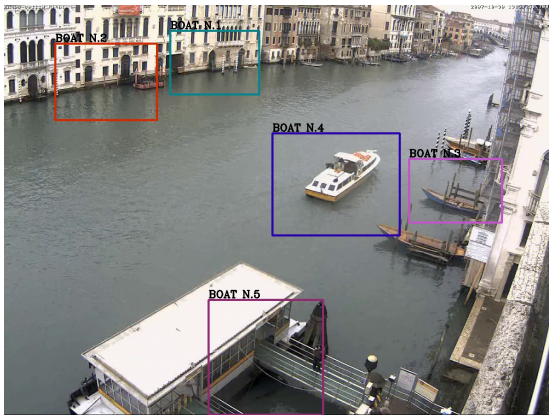
((o))


BOAT N.1
BOAT N.3

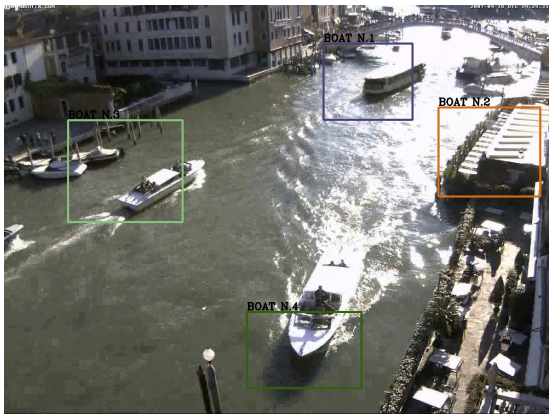
((p))


BOAT N.1

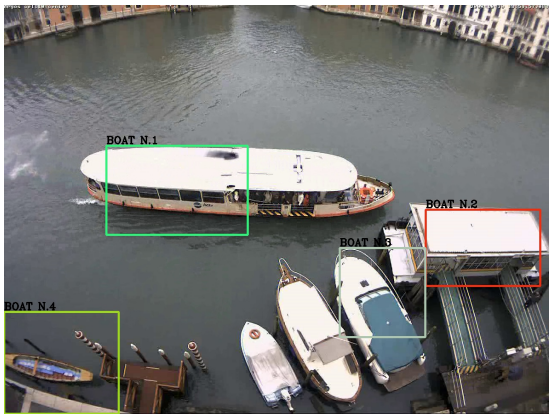
((q))


BOAT N.2
BOAT N.1
BOAT N.4
BOAT N.3
BOAT N.5

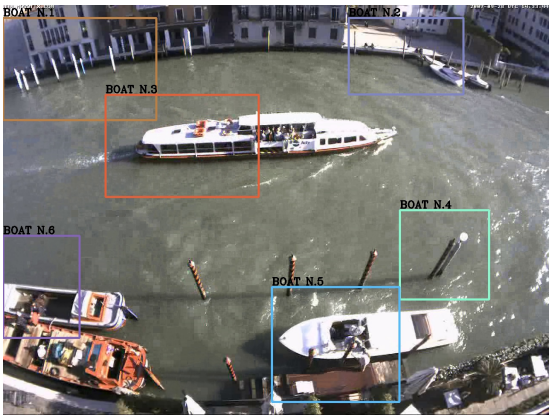
((r))


BOAT N.1
BOAT N.2
BOAT N.3
BOAT N.4

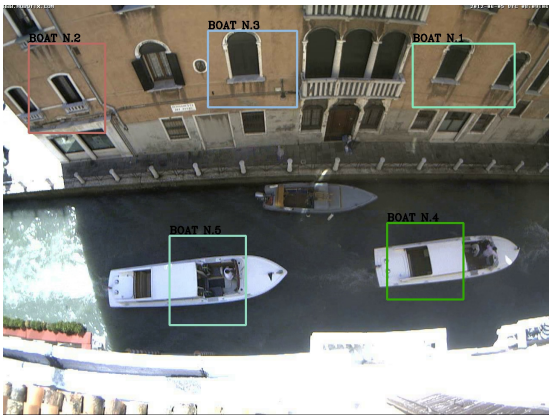
((s))


BOAT N.1
BOAT N.2
BOAT N.3
BOAT N.4

((t))



((u))



((v))

As it can be observed the obtained results are not so accurate, due to the complexity of the environment. These issues probably could be overcame by adopting a more complex CNN, trained on a wider dataset, composed even by gray-scale images. Indeed as it can be seen in the gray-scale images (Fig. 7(l), 7(m)) the model does not give precise results. Moreover, in some other images there are some boats which are located on a vertical position and CNN was not able to detect them, since the neural network was trained mostly with images of horizontal boats. On the other hand, in other images (for example Fig.7(k), 7(o), 7(s), 7(o)) the boat detector got better performances, even if not so precise as in the Kaggle dataset. Clearly, due to the environment complexity even the False positives are incremented. In Tab. 5.2 below the precise metrics obtained with this dataset are reported.

VENICE (IoU)	Boat n.1	Boat n.2	Boat n.3	Boat n.4	Boat n.5	#False p.
Image 0 (Fig.7(k))	0.397	0.733	0.196	-	-	1
Image 1 (Fig.7(l))	Undetected	Undetected	Undetected	-	-	1
Image 2 (Fig.7(m))	0.222	Undetected	-	-	-	1
Image 3 (Fig.7(n))	0.374	Undetected	-	-	-	1
Image 4 (Fig.7(o))	0.243	0.413	0.345	0.217	Undetected	3
Image 5 (Fig.7(p))	0.231	0.158	0.159	Undetected	Undetected	0
Image 6 (Fig.7(q))	0.384	Undetected	Undetected	Undetected	Undetected	0
Image 7 (Fig.7(r))	0.417	0.543	Undetected	Undetected	Undetected	3
Image 8 (Fig.7(s))	0.294	0.352	0.150	0.384	Undetected	1
Image 9 (Fig.7(t))	0.385	0.404	0.485	Undetected	Undetected	1
Image 10 (Fig.7(u))	0.165	0.212	0.371	0.408	0.434	2
Image 11 (Fig.7(v))	0.345	0.467	-	-	-	3

Obviously, it confirms the observations made above. The accuracy on this dataset is decreased for the reasons mentioned previously. Even if it was adopted an image pre-processing, the results for some images were not able to improve, due to performance limits of the Convolutional Neural Network. The parameters of the sliding windows are: stepSize = 30, windowRows = 150, windowCols = 150.

6 Segmentation results

In the following 2 sub-sections the segmentation results and corresponding metrics on both the Kaggle and Venice datasets are reported.

6.1 Kaggle dataset

As it is possible to observe in the following figures, the images related to the Kaggle dataset show good results as it is also confirmed by the metrics reported in Tab. 6.1.

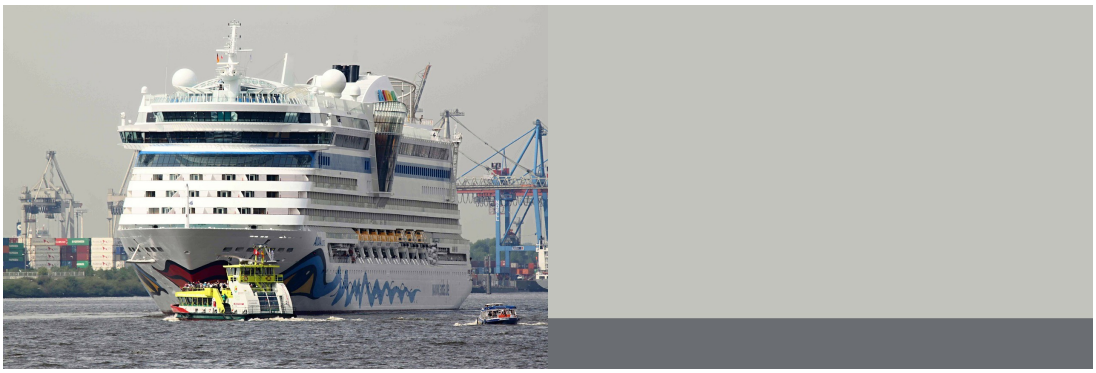


Figure 7

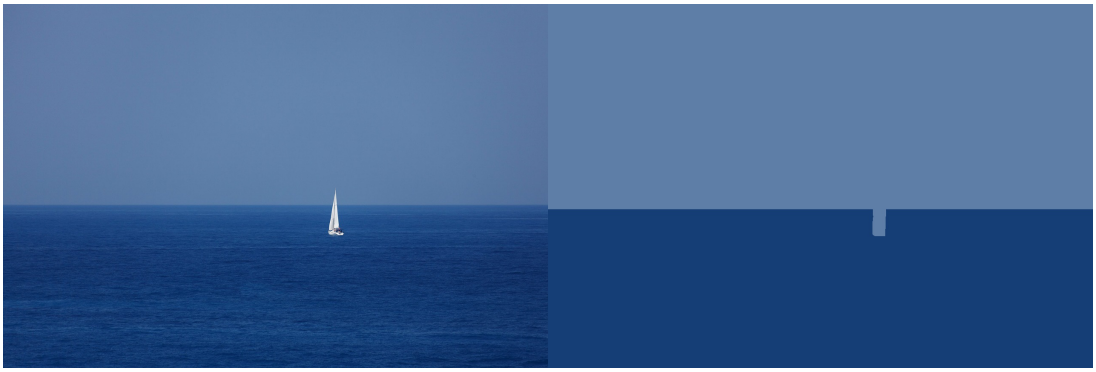


Figure 8

Figure 9

Figure 10

Figure 11

Figure 12

Figure 13

Figure 14

Figure 15

Figure 16

As it is possible to observe, the sea segmentation results are precise, and their accuracy is confirmed by the metrics reported on the following Tab.

KAGGLE DATASET	Pixel accuracy
Image 0 (Fig.7)	0.828
Image 1 (Fig.8)	0.982
Image 2 (Fig.9)	0.983
Image 3 (Fig.10)	0.867
Image 4 (Fig.11)	0.690
Image 5 (Fig.12)	0.972
Image 6 (Fig.13)	0.981
Image 7 (Fig.14)	0.801
Image 8 (Fig.15)	0.928
Image 9 (Fig.16)	0.892

6.2 Venice Dataset

For the sea segmentation on the Venice dataset, the Otsu method combined with HoughLine could not be adopted because it was not possible to find a line which divided the sea from the remaining parts of the image. Hence several methods were tried out, for example K-means segmentation based both on color and position or graph segmentation, but at the end the Otsu thresholding approach was adopted, by adding a combination of erosion and dilation. Their parameters, such as the number of iterations, were set ad hoc for each image. However it could not be possible to obtain the same accuracy obtained in segmentation of the Kaggle dataset, due to the color complexity of the images of Venice. In the following images are reported the results of the segmentation.

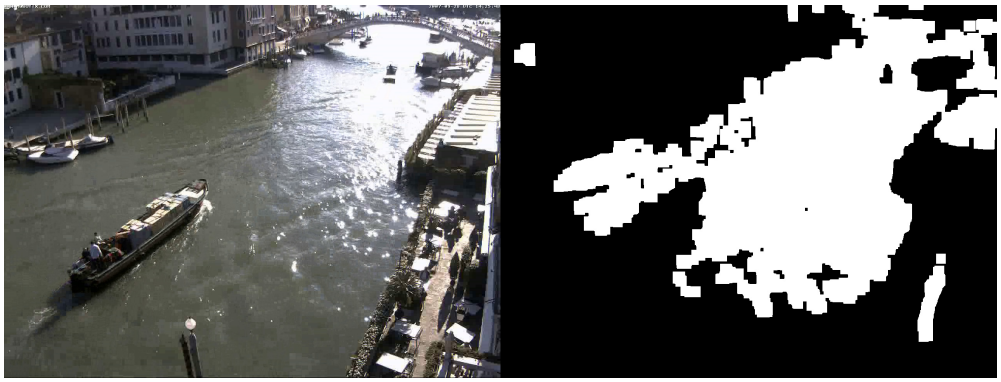


Figure 17

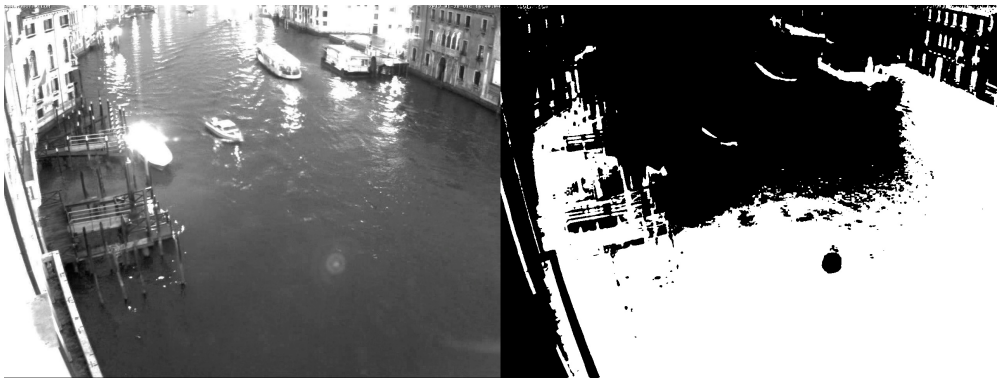


Figure 18

Figure 19

Figure 20

Figure 21

Figure 22

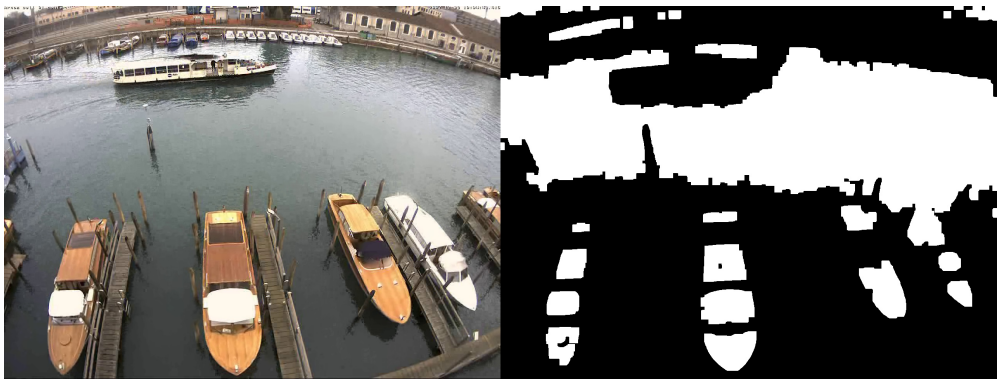


Figure 23

Figure 24

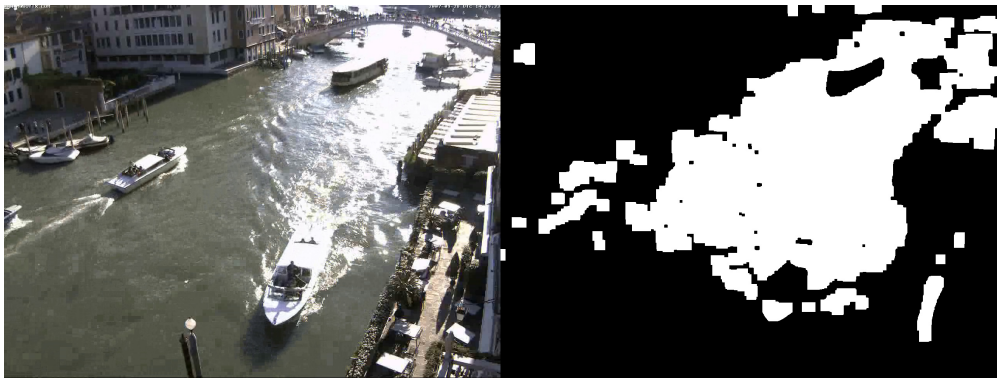


Figure 25

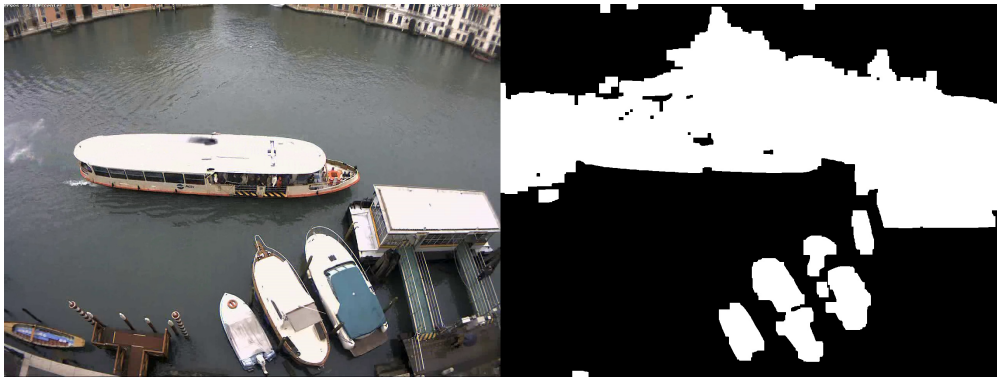


Figure 26

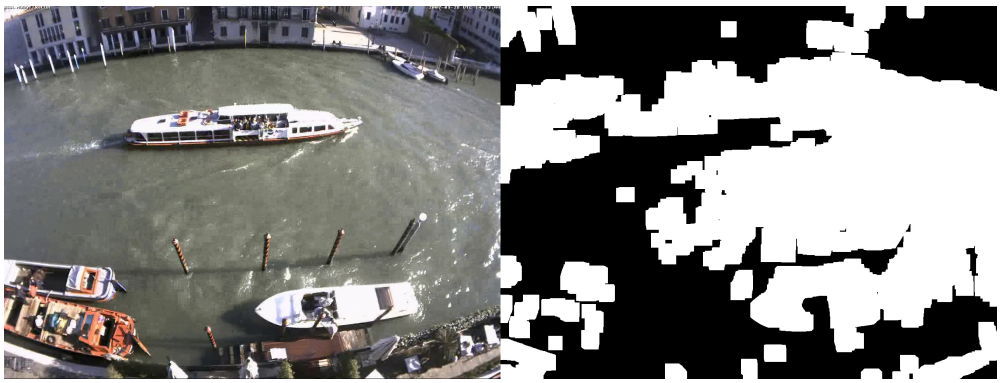


Figure 27

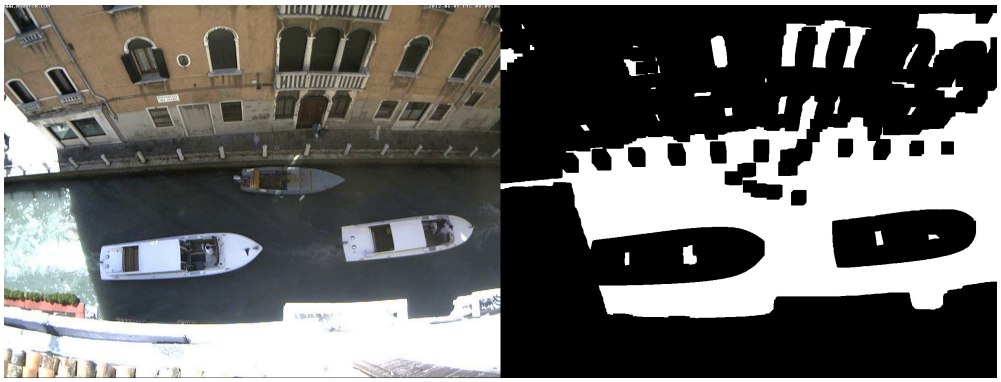


Figure 28

As it was described above, the resulting segmentation was not precise as in the Kaggle dataset; of course, adopting a machine learning method for the sea segmentation on this dataset, could improve significantly the results. In the following Tab.6.2 is reported the accuracy evaluated as explained in the correspondent section.

VENICE DATASET	Pixel accuracy
Image 0 (Fig.7)	0.456
Image 1 (Fig.8)	0.542
Image 2 (Fig.9)	0.535
Image 3 (Fig.10)	0.364
Image 4 (Fig.11)	0.528
Image 5 (Fig.12)	0.459
Image 6 (Fig.13)	0.435
Image 7 (Fig.14)	0.274
Image 8 (Fig.15)	0.420
Image 9 (Fig.16)	0.339
Image 10 (Fig.16)	0.510
Image 11 (Fig.16)	0.495

The values reported on the Tab confirms the observations made previously, indeed the pixel accuracy is lower than the pixel accuracy obtained on the Kaggle dataset, but however for some images is higher than 0.5.