Project Report: Boat Detector

Giulia Pezzutti

Student ID:1234012

23rd July, 2021

Contents

1 Introduction

The implementation of systems for the automatic boat detection is an important aspect for control systems that nowadays are applied for different purposes, mostly in real time conditions. In this sense, an example can be the traffic monitoring. Many approaches can be taken into consideration to solve this problem and one that can be adopted regards the combination of computer vision techniques with deep learning through neural network training and its subsequent use for prediction. In this project, a possible approach for the cited task is presented.

The pipeline built in this project for boat detection exploits the following steps: an initial pre-processing, in this case regarding colour space conversion, resize, noise removal and characteristics extraction, is performed, followed by the run of a ad-hoc trained neural network. For this purpose, for each image the neural network will be able to predict a mask, by definition an image containing white regions where the boat is predicted to be present and black in the other parts.

The work presentation is structured as follows: in section 2, the main choices for the dataset ground truth creation are reported in order to make the whole project replicable; in section 3, the exploited neural network and its training steps are introduced. In section 4 the implemented Boat Detector class is presented in its attributes and methods and section 5 describes the application of this class, with all its possible usages. At the end, in section 6, the main performances on some test dataset are presented in order to evaluate the prediction capabilities of the process.

The whole code is available also in the cited GitHub repository[1].

2 Ground Truth

In order to perform the training of the neural network, the first step of the project is the creation of the ground truth on two datasets: Mar[2] and Kaggle[3]. Two steps are needed for the complete building of the ground truth.

Initially, a tool[4] for the fast bounding-boxes creation around the object of interest has been exploited: for each image, after the indication of the boats location by the user, it generates a XML file (with the same name of the input image) containing the coordinates of top-left and bottom-right corners for each box found. With my colleagues, we decided to use two types of labels: boat and hidden boat. The first one allows to indicate fully-visible boats, while the second partially-visible ones; the boats of which only a very small portion is visible are instead not labelled. In this work, however, both labels types were used for the building of the correspondent masks, subsequently used as known labels in the machine learning techniques because in a neural network the presence of a whole or partial boat does not influence.

A short Python code for the conversion from XML to txt has been implemented, in order to obtain, for each image, a file (with the same name) containing in each row the label associated to a detected boat and its corners coordinates. In particular, in the (x, y) reference system inside the image, the reported coordinates are respectively x_{min}, x_{max}, y_{min} and y_{max}. In addition, if in an image no boats were detected, the txt file will be empty.

The cited implementation is reported in *src-python/xml_to_txt.py* file.

3 Convolutional Neural Network

The used neural network is a deep convolutional one composed by the repetition of convolutional, activation and batch normalization layers. The choice of convolutional layers derive from their ability to analyze images thanks to the repeated application of square kernels while the choice of batch normalization layers comes from the great possibility of generalization of the generated model that they can introduce. The whole network has been implemented in Keras and the definition of each layer with its main characteristics is reported in table 1.

The used input size is $(224, 224)$ since it is a standard shape for neural networks input as image; it allows, at the same time, to have a big enough image to maintain the main characteristics and not too big one, which could lead to a hough computation, not supported by the computer. The input image, in particular, after being reshaped to be acceptable for the network, is divided by 255 in each pixel value: in this way, the input nodes will assume only values between 0 and 1, a usefult aspect to have a better forwarding and subsequent gradient descent procedure.

The output is a set of nodes with shape equal to the input one: specifically, since its aims is to predict the whole mask for the boat detection (i.e. value 1 in the rectangles in which the boat is present, 0 otherwise), a number of nodes equal to the input image size and a 'sigmoid' activation function are necessary for the last layer. To be able to have the cited number of output nodes, in each convolutional layer, the padding parameter is set to 'same': in this way, it is possible to maintain in each block the same shape.

For the network training, the dataset has been created using the 'training' task of this project's main, as presented in section 5, starting from the images dataset and their ground truth created as presented in section 2. All the images inside this dataset have been used and, in particular, they have been separated into training (80%) and validation (20%) sets to have a validation of the model during training. The training has been performed with a mini-batches approach with batch size of 32 and it has been repeated for 10 epochs. For the weights learning, the training exploited the 'adam' optimizer and 'mean squared error' as loss function.

The code relative to the network training can be found in *src-python/training.py*.

Layer	**Properties**
Conv2D	k = (3, 3), f = 128
Batch Normalization	
Activation	'relu'
Conv2D	k = (3, 3), f = 256
Batch Normalization	
Activation	'relu'
Conv2D	k = (3, 3), f = 512
Batch Normalization	
Activation	'relu'
Conv2D	k = (3, 3), f = 1
Activation	'sigmoid'

Table 1
Convolutional Neural Network for mask prediction.

4 Boat Detector class

The BoatDetector class has been implemented in order to build the dataset for the neural network training or, the network is already available, for the prediction or evaluation of the performances. For this purpose, the class contain different members here reported:

- *img*: input image;
- *processed_img*: square image which has been padded, processed and resized;
- *name*: file name of the current image;
- *init_dim*: size of the input image;
- *init_dim_max*: maximum dimension of the input image, needed for the padding;
- *new_dim*: dimensions with which perform resize, corresponding to the neural network input shape;
- *mask*: ground truth mask associated to the current image (if available from input);
- *predicted_mask*: final mask predicted by the neural network, with some processing steps applied;
- *contours*: set of vector Points corresponding to the contours extracted from the thresholded network prediction.

Different class methods have been created in order to represent the task steps: they are reported in the following subsections and with them it is possible to further investigate the proposed members. As stated before, the overall application of the class is reported in section 5.

An example of the application of the methods to an image is reported in figure 1 and the class code can be found in *src/BoatDetector.cpp* and *include/BoatDetector.h*.

4.1 Image Preprocessing

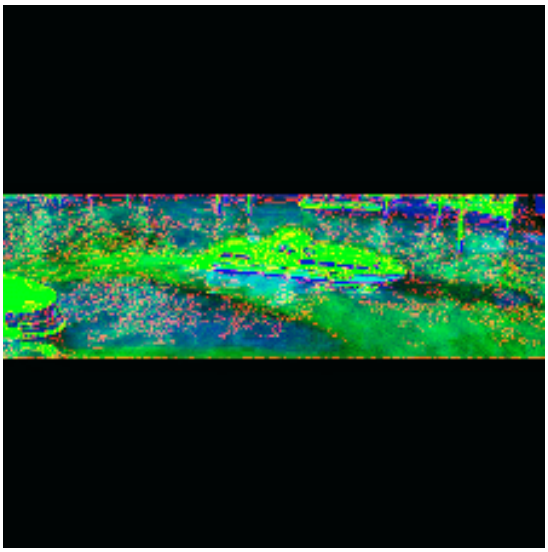
Different pre-processing steps are performed to the input image in order to obtain an image to be used as input for the neural network. These steps are necessary due to the high variability of the boats present inside the image: for this reason, it is not sufficient to only provide the image as input to the neural network, but is necessary also to extract some common characteristics from each image. Another aspect that must be taken into consideration is the fact that the input provided to the network must always have the same dimensionality and for this reason, techniques as features extraction cannot be taken into consideration since they provide a variable number of output vectors according to the image.

The input file is initially read as a *BGR* image. Thanks to the computation of the necessary border width and height and *copyMakeBorder* function, the image is padded in order to obtain a square image of size equal to the greatest input dimension. Note that for a more reliable subsequent computation during the neural network analysis, the image is placed on the center of the square, performing so a equally-spaced padding on the two needed sides.

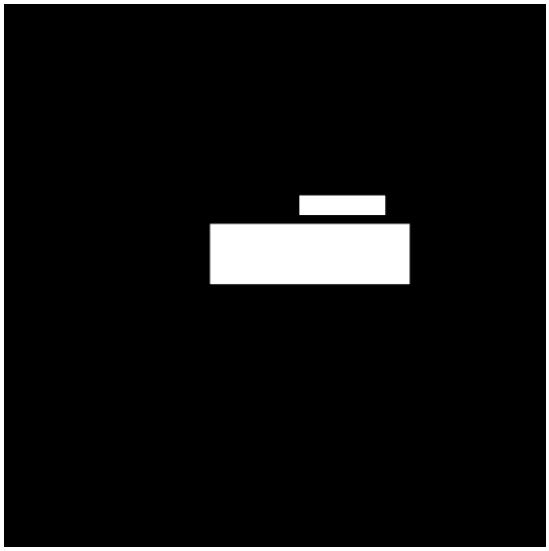
The boats in the datasets are very different one with respect to the other in terms of shapes, color and scale; in addition, the color can be altered by the glare of light and water. By consequence, the color information is not a fundamental characteristic for the prediction and, instead of focus the attention on that, it could be more reliable to focus on other characteristics. For this reason, the image is converted into HSV color space.

The *CLAHE* algorithm for histogram equalization is applied to each channel of the converted image independently: many images present an high amount of glare that could interefere with the prediction and this particular technique is well suitable for this type of contrast thanks to

(a) Initial input image.

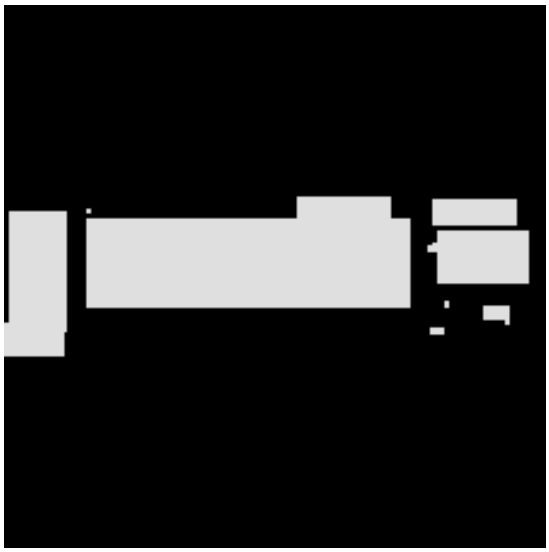


(b) Input image processed for the network.



(c) Ground truth mask extracted.

(d) Prediction of the network.



(e) Prediction of the network after the processing.

(f) Prediction applied to a input image.

Fig. 1. Processing applied an input image.

its adaptive equalization. The application of a Gaussian smoothing (with a kernel of $(5, 5)$ and sigma equal to 1) is performed to reduce the noise in the equalized image.

In order to extract useful characteristics for the prediction, Canny edge detector algorithm (with thresholds 50 and 200) is applied to the filtered image. The edge image is subsequently merged with saturation and value channels of the filtered image in order to obtain a new one composed by three personalized channels. In this way, the final image that will be given to the neural network will contain the information useful for the prediction.

At the end, the processed image is resized according to the desired dimension provided to the class constructor in order to make it suitable to be used as input for the network computation.

An example of the processed image is reported in figure 1b: it is necessary to take into consideration that due to the strange nature of the image, its visualization can be affected.

4.2 Mask Preprocessing

According to the needs of the program, it is possible (but not necessary) to extract the mask associated to the input image. In particular, the current class method receives in input the path to the txt file containing the mask in a format equal to the one presented in section 2. If the file does not exist or is not possible to open it correctly, an error is generated and the program stops.

The provided txt file is read independently for each row according to its definition (i.e., one row for each boat instance inside the image), and the corner coordinates are extracted. The vector of four extreme Points is generated starting from the coordinates, in order then to build the correspondent white rectangle in the previously generated black mask. The case in which no boats are present in the image is characterize by the fact the input file not contains any row: the method will so only generate the initial black mask.

By definition, the mask must have the same dimension of the input image: the same process of zero-padding (with the same procedure for the border width and height) to obtain a square image and the resize to *new_dim* size are performed. An example of the result is reported in figure 1c.

4.3 Make Prediction

This class method allows to perform a prediction according to the neural network model passed as input as *Net* object of the *OpenCV* class *dnn*.

The network has been trained with the condition to have as input only values between 0 and 1 and so, even during evaluation or prediction tasks, the input must be divided by 255, the maximum value that a pixel can assume. To perform a correct network usage, it is necessary to use the method *blobFromImage*, which allows to create, from the input images, the blobs that can be read as input by the network. While this operation is performed, a scaling a performed in order to obtain the input values between 0 and 1.

The blobs are then given in input to the neural network thanks to the *dnn::Net::setInput* method and the result is obtained with *dnn::Net::forward* function. To obtain the real network prediction, it is necessary to convert the output from blob to image (analogously with *imageFromBlob*): since it returns a vector in which every element is the corresponding output image for a input image, in this computation it has unit size and its element is extracted as output of the function.

In this way, as it is possible to notice in figure 1d, the raw mask prediction is extracted and saved but to be meaningful, to perform a correct visualization and a prediction evaluation, the following methods are necessary.

4.4 Prediction Processing

Since the prediction made by the network is 'raw', it must be processed to obtain a mask as the ground truth one for the input image, i.e. with white rectangles representing the boats detected. Firstly, a process to handle with the fact that the network is predicting float values between 0 and 1 and not 0 or 255 as needed for the mask must be performed. In addition, the neural network predicts random shapes, according to the weights it has learned: it is also necessary to find the smaller rectangle within the shape to be then compared with the provided ground truth.

To deal with the presented tasks, a thresholding operation is applied to obtain the 0-1 mask: in particular, if the pixel value is greater than 0.95, it is substituted with 255, otherwise with 0. The predicted mask is initially processed with a Canny edge detector with thresholds 50 and 200 in order to improve the subsequent contours extractions. With *findContours* function, the contours found inside the edges image are extracted. Each contour is subsequently analyzed thanks to *approxPolyDP*, to approximate them with a close polygon with precision 3 and, for each of them, the correspondent up-right bounding rectangle is built with *boundingRect*. These rectangle contours are then applied to a black mask thanks to *rectangle*: in particular, the correspondent rectangles in the new mask are completely filled with white color. An example of the process is reported in figure 1e: note that the obtained Mat is still with the output shape of the network (so equal to *net_dim*) to allow the eventual evaluation.

4.5 Apply Prediction to Input

In order to make the mask understable for the user, it is applied to the input image (without processing). To do that, the inverse operations of padding and resize are applied, obtaining in this way the mask for the image of *init_dim* size. To simply apply it to the image, the external contours (so the external shape of each predicted rectangle) are extracted and applied to the input image. This final image is the one that is at the end shown to the user since it shows the real prediction and eventually can be saved, as reported in figure 1f.

4.6 Prediction Evaluation

The prediction can be evaluated in terms of Intersection over Union metric, if the ground truth mask is available. In this case, the union and the intersection of the true and predicted masks are calculated respectively with *bitwise_or* and *bitwise_and* functions and for each of them, the count of not-black pixels are performed: as a matter of fact, being masks, the images will have only 0 and 255 pixel values and counting the cardinality of the cited group of pixels allows to calculate the 'area' of the total rectangles.

The final metric value is obtained, according to the definition, dividing the intersection area by the union area. For the metric definition, note that better performances are acquired when this distance is close to 1 with respect to close to 0.

5 Main

Thanks to how the presented BoatDetector class has been built, it is possible to use it for different purposes: in particular it can be exploited to build the dataset necessary for the training of a neural network (called *training* task), to evaluate the performances of a pre-trained network on one or more images when the relative masks are known (called *evaluation* task) or to predict the boat localization in one or more images for which the ground truth is not known (called *prediction* task). Depending on the type of task chosen by the user through command line (with the input folder or image path and the eventual masks folder), different steps will be performed.

Since the tasks options available for the user are multiple, some checks must be performed to have a subsequent correct execution. As a matter of fact, the user must input a single task, provide an existing folder for the input images and, if evaluation or training are chosen, the masks folder path must be passed too. Note that in all tasks it is possible to provide the path to the images folder or the path to a single image: in both cases, however, only the masks folder (and not the single mask path). For sake of simplicity, the ground truth has been completely built in order to create the mask files with the same of the relative image. This assumption is kept also in this application and the mask is needed but not found for an image, an error is generated.

In case of evaluation or prediction tasks, the application should also correctly load the trained model that must be already saved in *model/boat.pb*.

After the cited checks, the paths of all images inside the folder are extracted and the program will cycle over all of them to analyze each image independently. If the path to a single image is passed in input, the execution will generate a single iteration in which the input image is analyzed. At each iteration, the image corresponding to the path under consideration is created and an instance of BoatDetector class is generated for its analysis. The image preprocessing is always performed and, depending on the chosen task, different functions are then called:

- For training, image and mask processing are performed and both the results are then saved in the correct folder (generated inside the input one);
- For evaluation, image and mask processing are performed; subsequently the whole set of methods for neural network prediction, processing, visualization and evaluation are executed. For this task, in addition, at the end of the cycle over all the images, an average Intersection over Union value over the dataset is provided to the user to measure the mean ability of the network to predict the masks;
- For prediction, after image and mask processing, only the neural network prediction, analysis and results visualization are performed. In this case it is not completely possible at all to provide a measure of the goodness of the prediction.

The implementation of the main can be found in *src/main.cpp* and how the code can be executed is provided in the *README.md* file.

6 Performances Evaluation

The performances of the presented code are evaluated thanks to the last class method when the evaluation task is main is requested. An initial evaluation can be done with the training steps of the network: in particular, during the different epochs, the validation loss reached values of 0.0825, demonstrating so a good ability of the network to learn from the dataset. Other tests results are reported for the two datasets cited as test sets in the project specification.

6.1 Kaggle test set

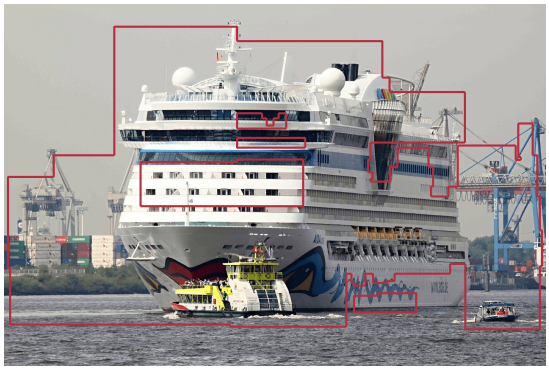
The results for Kaggle dataset are reported in figures 2: the value of IoU metric for each image is reported in its caption. The prediction for figures 2a, 2b, 2d, 2f and 2g is accurate and so the model is able to well detect the boat: in these images, the boats are in general well distinguishable from the background and few amount of glare is present. In figures 2b, 2e (even if the boat is very small), 2h and 2j, the model is predicting as boat quite all the image, demonstrating that in this case it is not able to distinguish it; probably an higher value applied for the thresholding would or some ad-hoc preprocessing steps allow the mask to perform a better distinction, avoid the high number of false positives pixels.

The mean IoU value for the Kaggle dataset is 0.3733: it indicates a good boat detection algorithm. This mean value is highly influenced by the two images in which no boats were predicted and by the fact that the masks are created completely rectangular around the boats, while the predictions could be more able to follow the shape

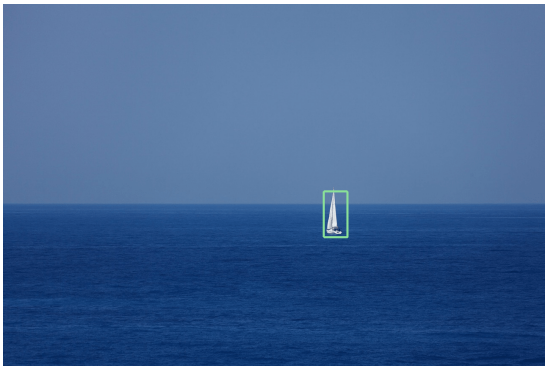
6.2 MAR test set

The results for Mar dataset are instead reported in 3: in this case the mean IoU obtained is 0.2837. The model, in this case, is making more effort to detect only the boats due to the presence of the buildings in the background. As a matter of fact, they are often detected as boats, probably due to their similarity with cruise ships, a type of boats present in many images inside the training dataset. Good results are however obtained in 3f, 3h, 3k and 3l.

In this cases, a different preprocessing step should be take into consideration: for example, it could be useful to firstly approximately segment the water and subsequently highly smooth the parts that are not strictly near this part. This preprocessing could be useful for this type of images but it has not been adopted since it is not possible to apply it to all the images in the dataset (e.g. in some of them, only the boat and not the see is visible), leading so to a worst training.



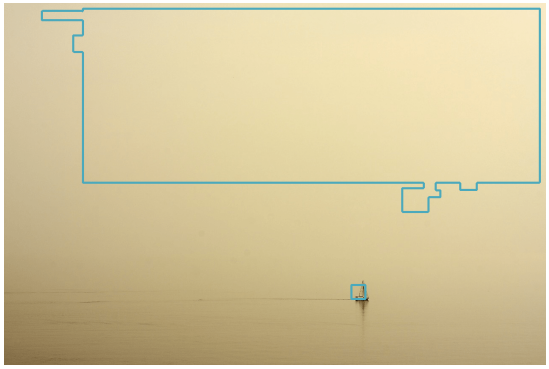
(a) IoU = 0.5788



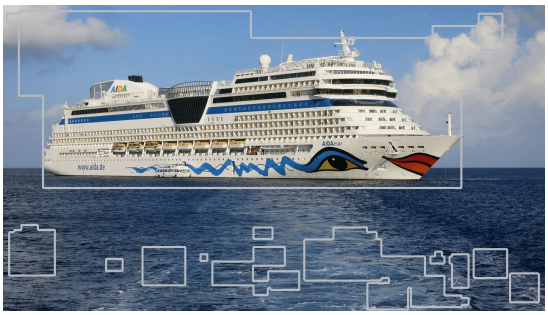
(b) IoU = 0.8090

(c) IoU = 0.1036

(d) IoU = 0.7774

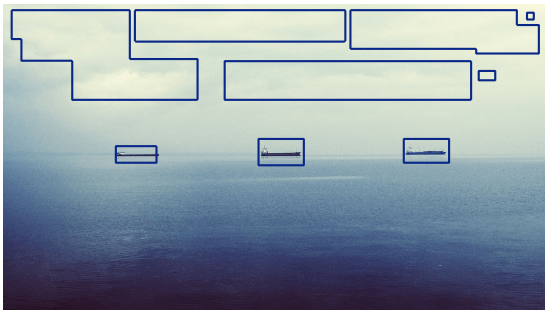


(e) IoU = 0.0018



(f) IoU = 0.7188

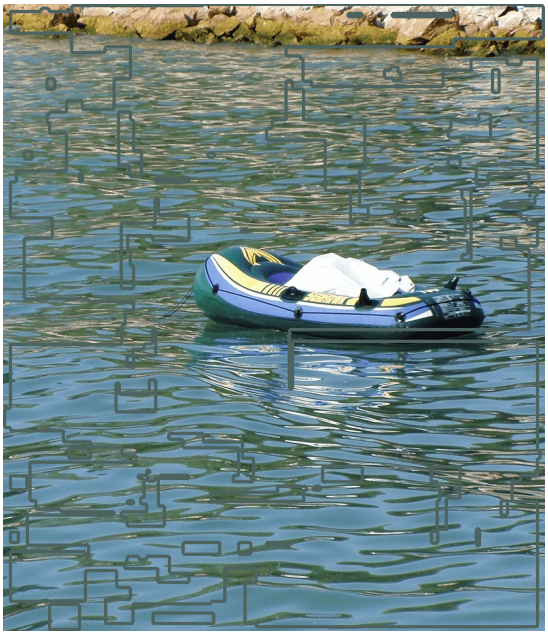
(g) IoU = 0.4848



(h) IoU = 0.0412

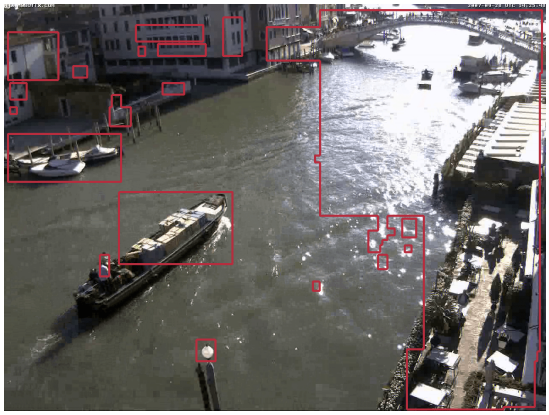
Fig. 2. Results for Kaggle dataset

(i) IoU = 0.1094

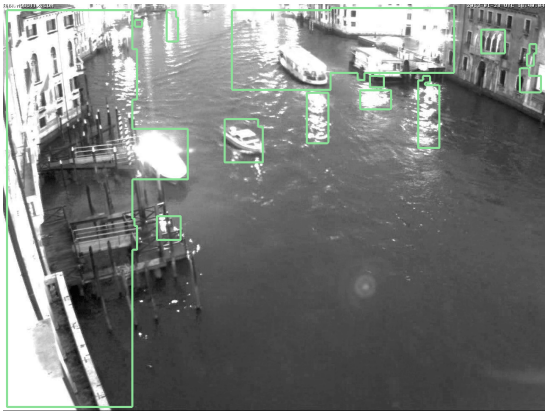


(j) IoU = 0.1079

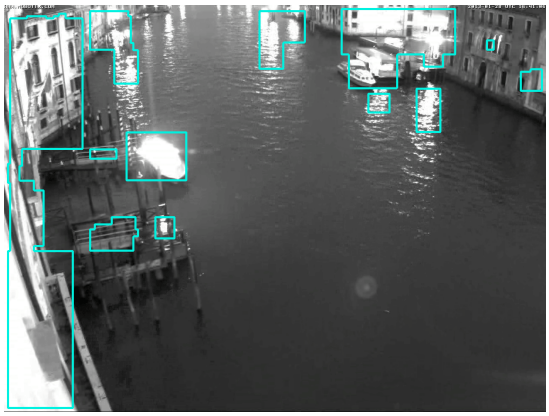
Fig. 2. Results for Kaggle dataset.



(a) IoU = 0.1174



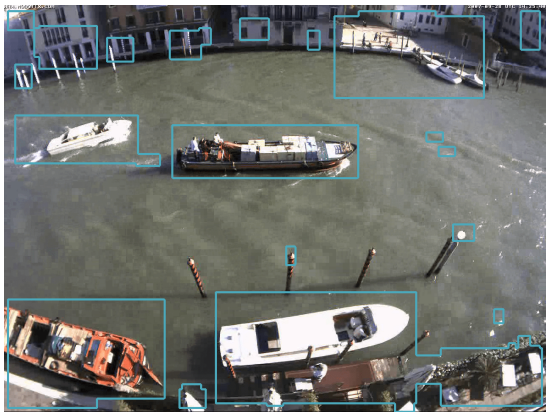
(b) IoU = 0.0881



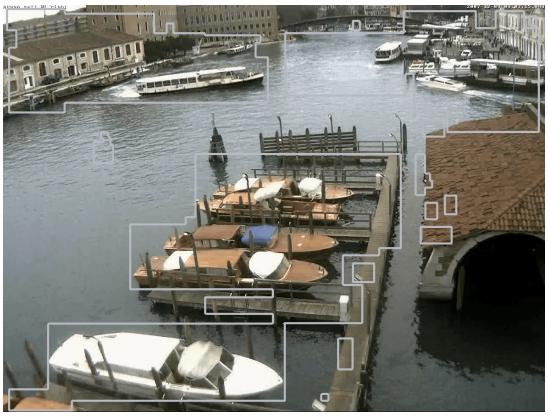
(c) IoU = 0.0833

(d) IoU = 0.1385

Fig. 3. Results for Mar dataset.

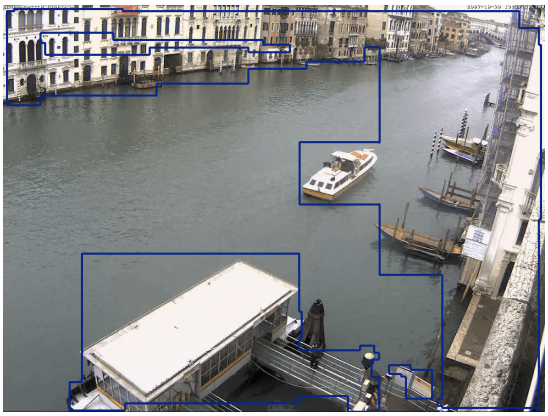


(e) IoU = 0.5413

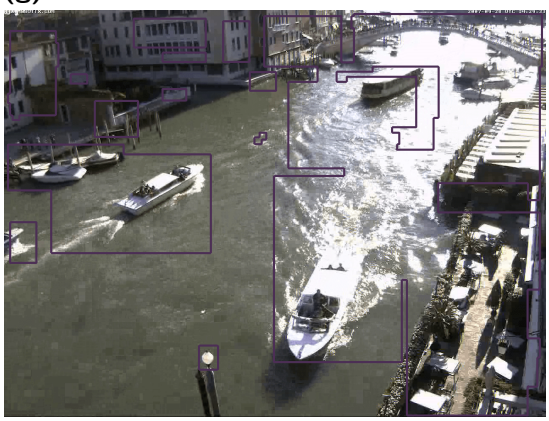


(f) IoU = 0.4479

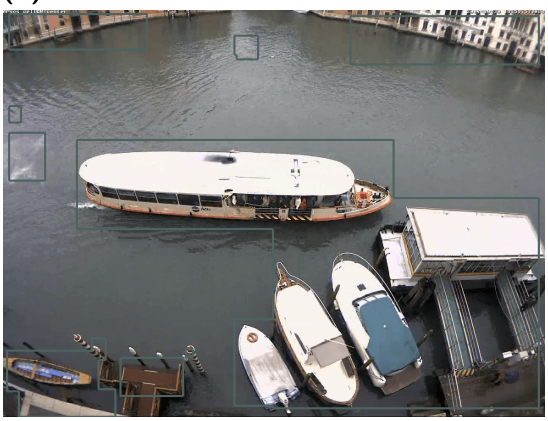
(g) IoU = 0.5562



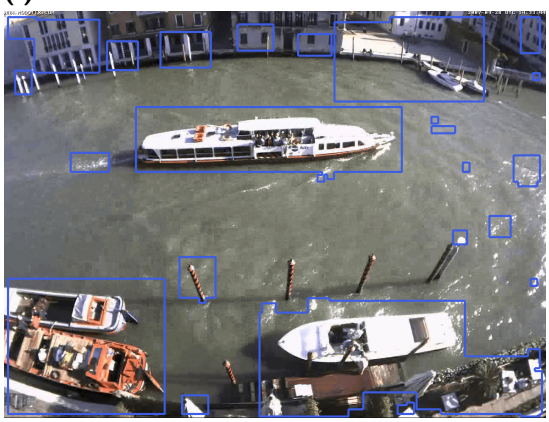
(h) IoU = 0.1071



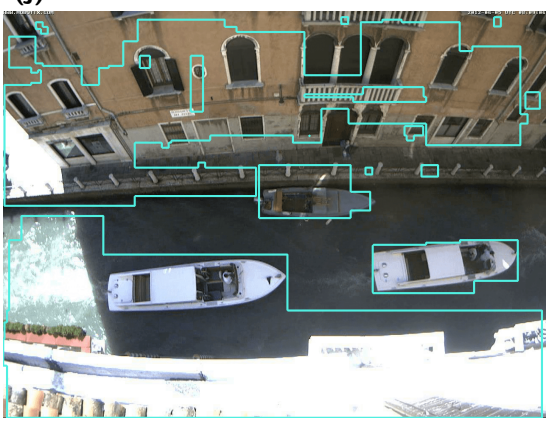
(i) IoU = 0.1705



(j) IoU = 0.5029



(k) IoU = 0.5148



(l) IoU = 0.1367

Fig. 3. Results for Mar dataset.

7 Conclusions

The task of boat detection is a very hard task due to the high variability of the images that can be input, both in terms of scale and rotation but also of shape and colour of each boat inside it. The images can then introduce other variability due to contrasts or saturation level. Even inside the same image, the boats can be really different and the algorithm must be able to well recognize all of them. In general, convolutional neural networks are well suitable for the image analysis but, in this case, it has been difficult to find a model able to well generalize the entire dataset. In order to make the prediction effective, so, it is necessary to have a meaningful pre-processing step, a big input dataset and important computational resource through which the neural network can learn all boats' characteristics to reduce the variability level.

Other difficulties have been introduced from the use of *dnn* in *OpenCV*: as a matter of fact, since it is a quite new implantation, some characteristics are not yet completely available and its usage is not always intuitive. An example of these limitations is the fact that Keras Dropout layer is not present in the current version of *dnn* and even if it would allow a better generalization during the training, it is not possible to use them.

The proposed work, however, is in general able to detect the presence of the boats in the images, demonstrating that computer vision and deep learning techniques can help one to other to reach a effective predictor. Possible future works for this task could exploit the extraction of different characteristics to be used as input of the neural networks and the use of region-based convolutional neural networks, that should be however fully accepted by *OpenCV*.

8 References

1. https:/github.com/giuliapezzutti/boat-detection.
2. http:/www.diag.uniroma1.it/~labrococo/MAR/classification.htm.
3. https://www.kaggle.com/clorichel/boat-types-recognition/version/1.
4. https:/github.com/tzutalin/labelImg.