IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 19, 2022, accepted February 23, 2022, date of publication March 16, 2022, date of current version March 25, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3159667

An Efficient Cascaded Model for Ship Segmentation in Aerial Images

CARLOS PIRES[1], BRUNO DAMAS[1,2], AND ALEXANDRE BERNARDINO[1], (Senior Member, IEEE)

[1]Institute for Systems and Robotics (ISR), 1049-001 Lisbon, Portugal
[2]Centro de Investigação Naval (CINAV), 2810-001 Almada, Portugal

Corresponding author: Bruno Damas (bdamas@isr.tecnico.ulisboa.pt)

This work was supported by the Portuguese National Project VOAMAIS under Grant 02/SAICT/2017/31172.

ABSTRACT In this work, we propose a new method for real-time ship segmentation during maritime surveillance missions using aircrafts with onboard video cameras. We propose a cascade model with a detection stage followed by a segmentation stage. The detection stage selects candidate regions (bounding boxes) likely to contain ships. These bounding boxes are passed to the segmentation stage, where the ship segmentation mask is then obtained. By focusing the segmentation effort only in the image regions recommended by the previous detection stage, it is possible to improve the overall image processing time and, simultaneously, to obtain a better segmentation score than the one obtained by monolithic segmentation models that are trained in an end-to-end way. Additionally, we test the viability of using a Conditional Random Field model as final boundary refinement stage: although such model can improve the segmentation results when a full segmentation approach is used, our experiments did not show any significant improvements when using our proposed cascade model. We trained the detection and segmentation models with aerial ship images from publicly available maritime datasets. We tested the cascade model on the Airbus ship detection challenge, showing real-time performance and accurate maritime ship segmentation, comparable to state-of-the-art results.

INDEX TERMS Maritime surveillance, ship detection, ship segmentation, real-time image segmentation, convolutional neural networks, deep learning.

I. INTRODUCTION

Maritime surveillance missions are of great importance for search and rescue operations in accident or natural catastrophe scenarios, and to track, prevent and discourage illegal activities such as drug trafficking, illegal fishing, and illegal cargo movement. For security, financial and environmental reasons, countries need to control the activities that affect humans and the ecosystem in the corresponding exclusive economic zones (EEZ).

An effective surveillance of such economic zones may be impracticable, due to the the lack of sufficient human and material resources, specially when these zones correspond to large maritime areas: Portugal, for instance, with a land area of 92 212 km^2, has the 5th largest EEZ within Europe, with 1 727 408 km^2. This motivates the use of unmanned aerial vehicles (UAVs) as an efficient and cost-effective solution to extend the surveillance capabilities, as these flexible and extensible systems can incorporate a vast number of sensors and can be operated autonomously [1].

While image detection addresses the problem of identifying the presence of certain classes of objects in the image, resorting to bounding boxes to locate the detected instances, image segmentation performs classification of the image on a per-pixel basis, *i.e.*, assigns a class to each image pixel. Segmenting possible ships in an aerial image provides information that can not be conveyed by an automatic detector's bounding box alone: besides ship size, the estimated silhouette can be used to calculate the ship orientation and route. Moreover, the ship shape, obtained from image segmentation, can be automatically matched to the ship Vessel Maritime Mobile Service Identity, given by the AIS signature provided by the ship transceiver. If all these calculations are performed onboard the UAV, without any human intervention, the bandwidth requirements for the datalink between the UAV and the Ground Control Station

The associate editor coordinating the review of this manuscript and approving it for publication was Vivek Kumar Sehgal.

can be considerably reduced, since only the most crucial information regarding the identified ships is relayed (*e.g.* suspicious situations), thus avoiding a full video stream transmission. As a consequence, the surveillance mission range can be significantly increased.

The main goal of this work is to develop a system for automatic ship segmentation in airborne color image sequences, taken from standard RGB cameras, that can be run in real-time in the UAV embedded hardware, thus extending the autonomy of maritime surveillance missions. Many UAVs used in this kind of missions also have mounted thermal cameras that can provide images in the near infrared spectrum, but for generality we only consider here the use of commercial off-the-shelf RGB cameras and embedded processing units. Segmentation of maritime surveillance images is a challenging task: often the ship represents only a small portion of the image, and it may be surrounded by distracting elements such as waves, wave crests and sun glare. In the past decade deep learning approaches have consistently achieved state-of-the-art results in image segmentation tasks, at a cost of large training times and heavy computations performed at inference time. Even if there are some specialized accelerated hardware solutions for image processing and for running deep neural networks in embedded systems (*e.g.*, Jetson GPU family [2]), image segmentation in real-time is still a demanding task in this type of hardware, as its computational power cannot be matched to the one provided by top desktop GPUs used in deep learning research. Therefore, it is essential to reduce the computational complexity of segmentation algorithms in order to process the images with the required frame rate onboard the UAV.

The main contributions of this paper are:

- An efficient algorithm to segment ships in airborne aerial images, using a cascaded approach composed of two distinct deep neural networks, that can run in real-time in standard embedded processing units;
- The evaluation of performance gain when a post-processing step based on a Conditional Random Field model is used.

The first network is a deep convolutional detection network that extracts image locations with high likelihood of containing ships, and currently many fast detection networks of this kind can achieve real-time framerates, even on embedded hardware [3], [4]. Typically, in this detection stage, most of the original image is discarded and only a small portion is provided to the second stage of the proposed cascaded approach. In the second stage, a segmentation network processes the image regions provided by the detector, thus significantly saving computation time, as the full image is no longer processed. Besides the speed gain, we also verified that this method also improves the performance with respect to end-to-end whole image segmentation, as will be discussed in the next sections. Finally, we also evaluated the segmentation performance when a Conditional Random Field model is used as a post-processing step: although such model has shown to improve the segmentation score when a full segmentation approach was used, it did not show any significant improvements when using our proposed cascade model.

The remainder of this document is organized in the following manner: Section II presents a review of the current state-of-the-art related to image detection and segmentation, with a focus on maritime scenarios. Then, in Section III, we explain the proposed methodology and the choice of the deep neural networks used in the proposed architecture. In Section IV we describe the experimental setup, the training procedure and the datasets used for training, testing and validation. Section V provides and discusses the results obtained by the proposed approach and finally Section VI summarizes the relevant aspects of this work and presents guidelines for future work.

II. RELATED WORK

Since 2012, when Krizhevsky *et al.* [5] trained a convolutional neural network (CNN) on ImageNet and achieved remarkable results in a image classification task, the use of deep learning methods for image analysis has grown dramatically. Progress in the development of very deep architectures for performance boost, like VGG [6], and residual networks like ResNet [7], for improved convergence, led to significant improvements in detection and segmentation tasks results.

Detection algorithms both localize and classify objects in images. Localization information is typically represented through a bounding box (BB) that encloses the detected objects. Several detection architectures were developed during the last decade: initial models, such as R-CNN [8], first generate bottom-up region proposals, then extract features on the region proposals using a CNN, and then classify the obtained feature vector with a Support Vector Machine (SVM). In recognition systems, Feature Pyramids Networks (FPN) are also useful components for object detection, providing a top-down pathway to construct higher resolution layers by generating multi-scale feature maps with better quality information: in [9] the authors introduced a new method for building them inside a CNN, combining low-resolution, semantically rich features with high-resolution, semantically weak features. More recent and efficient approaches like YOLO [4] entirely skip the region proposal stage from two stage detectors like R-CNN through the use of a fixed set of anchor bounding boxes. Features are extracted by a single CNN applied to the whole image, and then used to classify the objects in the anchor boxes and adjust their position and scale to fit tight to the objects of interest.

Segmentation algorithms assign a label to each pixel in the image. If the label corresponds to a class, this is denoted Semantic Segmentation. Instead, if the label is assigned to an instance of a class, this is denoted Instance Segmentation. In segmentation problems we want to generate an output image of the same size of the original image, containing the classification of each pixel. It is convenient that a segmentation network is able to process images of arbitrary size to prevent loss of resolution.

In [10] a Fully Convolutional Network (FCN) is presented that can segment images of different input sizes. However, in this architecture some resolution is lost because max pooling and sub-sampling operations reduce the feature map resolution, impacting the final segmentation result. Segnet, on the other hand, is a deep neural network architecture that introduces the encoder-decoder concept on CNNs to recover the resolution of the input image [11]. In this architecture the low resolution encoder features are mapped by the decoder to full input resolution feature maps, by using pooling indices, computed in the max-pooling step of the corresponding encoder, to perform non-linear upsampling. U-Net is an alternative architecture that was developed for biomedical image segmentation, based on a fully convolutional network modified and extended to converge with fewer training images and to yield more precise segmentations [12]. Fig. 1 shows the U-Net architecture. The left side represents the encoder composed of a contracting path to compute features. The right side represents the decoder that has an expanding path to localize patterns. U-net uses the feature maps computed in the contraction path to expand the encoder representation into a segmented frame. Thus, the structural integrity of the image is preserved, which decreases the output distortion.

As shown in Fig. 1, the contracting path applies multiple pairs of two 3*3 convolutions, each followed by a ReLU layer. After each group of two convolution layers it is implemented a 2*2 max-pooling operation with stride 2 for downsampling. At every downsampling step the number of channels duplicate. The expansive path performs the opposite operations: it uses 2*2 up-convolutions and decreases the number of channels to half. Plus, it concatenates high-resolution contracting path features followed by two 3*3 convolutions, each with a Relu. Since the network uses feature maps from the initial layers, the loss of pattern information is prevented. Besides, with a large number of feature channels, context information can be propagated to layers with higher resolution. The U-Net final layer consists in a 1*1 convolution that generates a segmentation map from the features maps. U-Net does not have fully connected layers, which leads to fewer model parameters to train; also, it can achieve competitive results due to the skip connections, that gather information from multiple scales of the image, allowing data from the upper and deeper layers of the architecture to be directly used in the decoder layers, thus also enabling a better flow of the gradients during the training phase.

Segmentation networks can be applied to multiple scenarios, and in this paper we focus on maritime scenarios and aim at segmenting ships from aerial images. This problem can also be addressed with synthetic-aperture radar (SAR), that allows creating two-dimensional images or reconstructing a three-dimension object, and it is quite popular in ship detection. In [13], the authors present an application of a Faster R-CNN in SAR ship detection. The Faster R-CNN generates bounding boxes, and those with a low score pass through a constant false alarm rate (CFAR) detector. CFAR is a pixel-based detection method that calculates the probability of false alarm associated with a detection threshold. Image acquisition and processing with SAR sensors requires additional computation resources and has limited data acquisition speed and real-time capabilities compared to RGB images acquired from UAVs. [14] presents a framework called Rotation Dense Feature Pyramid Networks (R-DFPN) capable of detecting ships in the ocean and in ports with RGB images. The Feature Pyramid Network gathers multilevel information and achieves state of the art results in small object detection tasks, and in R-DFPN this architecture is improved by adding connections between feature maps. The authors of [15] use recorded data to help identify a vessel. They investigate how temporal features could improve ship detection in video sequences captured by a small aircraft. They build a convolutional long short-term memory (LSTM) to learn those features and increase ship detection rates. In [16] the authors propose a method capable of performing ship detection in challenging surveillance conditions. They track vessels with a correlation filter complemented with image segmentation. To compensate for drifts in the correlation filter, they apply blob analysis to re-center the target in the tracking window. To perform segmentation, the authors use the Otsu's method [17] to segment the image into two classes, bright and dark parts of the image, that will correspond to the ships and the ocean surface, respectively. In [18], a robust detector for aerial images captured during maritime surveillance missions is presented. They modify a CNN and create tracks with the successive detections to predict the position of the objects in future frames. That method combines a CNN that generates detection proposals, bounding box-shaped, with a Multiple Hypothesis Tracker (MHT). The MHT increases robustness by creating associations between detections and tries to predict the future position of the vessel.

The Kaggle Airbus Ship Detection Challenge provides a good testbed for ship segmentation in aerial images, as in this competition it is required to locate ships in images and to put an aligned bounding box segment around the located ships [19]. Although the ground-truth segmentation masks in this challenge consist simply in rotated rectangles, a feature that some competitors can exploit, it nevertheless provides a large annotated maritime dataset for training and testing of deep learning algorithms, allowing for comparison of results in an independent test set with different approaches and algorithms. This independent test set, not available to the competitors, comprises 93 % of the test data: the segmentation score on this dataset is called the challenge private score. The remaining 7 % of the test data are available to the competitors: the results obtained using these data are denoted the public score.

In [20], the authors present an algorithm to segment ships from aerial images. They implemented the U-Net network and achieved a private score of 0.845 with an inference time of 1.21 frames per second, which comes quite close

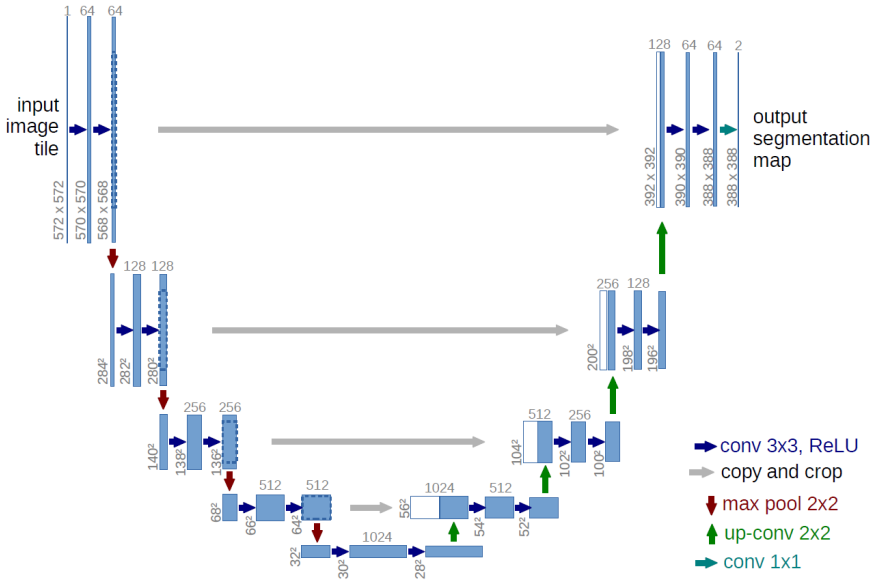


FIGURE 1. U-Net architecture. Figure taken from [12] with permission from authors.

to the results of the winner of the contest (private score of 0.854).

Our main goal is to develop a segmentation algorithm that still maintains a good segmentation performance, *i.e.*, that can achieve a competitive score on the Kaggle Airbus Ship Detection Challenge, while significantly lowering the inference time required to process an image in order to meet real-time demands. We implement a cascade model capable of identifying ships in aerial images captured by a UAV. The algorithm has two different stages: first, we use a detection network to search for possible ship location regions, and then we pass these regions through a segmentation network based on the U-Net, thus narrowing the image region where segmentation is performed, which significantly improves the overall image processing speed. We train the two different stages of this cascade model using the Seagull [21] and Kaggle [19] datasets. Additionally we evaluate the possible performance gain if conditional random fields are used to improve the segmentation results, using it as a post-processing step after the segmentation network.

III. METHODOLOGY

The embedded computers on the UAVs assigned for maritime surveillance missions by the Portuguese authorities typically have limited computational power for computer vision tasks, and current state of the art segmentation networks like U-Net are not compatible with real-time processing on this kind of hardware. Since during these missions the ships we intend to segment only represent a small part of the image, we propose a cascade approach to reduce the computation time and make it compatible with real-time demands, composed of the following stages:

- Detection: possible ships locations are identified on the whole image;
- Segmentation: Ship segmentation is performed solely on the locations provided by the previous detection phase;
- Post-processing: as a final pos-processing phase, the individual segmentations carried out in each detection bounding box are merged by the Bounding Box Mask Aggregator to provide a single segmentation mask for the image. In this stage the final segmentation can also be improved by using post-processing techniques like Conditional Random Fields models.

With this approach we expect the processing time to be significantly reduced, since detection using state of the art detectors is a much faster operation than segmentation, and the segmentation network only runs on a small part of the image, provided by the detection stage, therefore avoiding large areas of the image corresponding to the ocean, sky or coastline. As a final stage the individual segmentation masks for each detection are aggregated to provide an output mask with the same size as the input image. Fig. 2 show the global system architecture. The implementation of the cascade model was made in C and Python and is available on GitHub.[1]

A. DETECTION NETWORK

There are currently many efficient and fast detection algorithms such as R-FCN [22], RetinaNet [23], and YOLO [4], to name just a few: their performance on a detection task on the COCO dataset is summarized in Table 1.

Since detection speed is the main concern we will use the YOLOv3-tiny model in our work. The YOLO network is a one stage detector that splits the image into a grid,

[1] https://github.com/Cpires97/Ship-seg

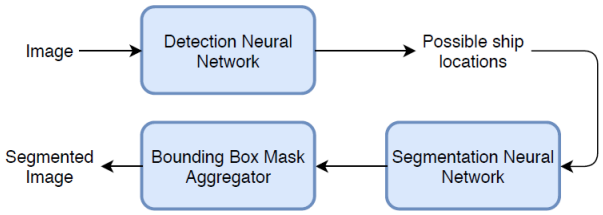


FIGURE 2. Segmentation global system architecture.

TABLE 1. Accuracy and speed of some detection networks on the COCO dataset (values taken from [4] and [23]).

Network	Average Precision	FPS
R-FCN	51.9%	12
FPN FRCN	59.1%	6
RetinaNet-50-500	50.9%	14
RetinaNet-101-800	57.5%	5
YOLOv3	51.9%	20
YOLOv3-tiny	33.1%	220

where each cell of the grid is responsible for predicting the bounding boxes and the probabilities of containing an object. According to Table 1, YOLOv3 can process 20 frames per second on the COCO dataset, and its faster version, Tiny Yolo, can reach 220 frames per second, a consequence of having fewer parameters and a lower number of layers. The inference results presented in Table 1 are usually obtained using top performance graphic cards (in [23], for instance, an Nvidia M40 GPU is used to evaluate RetinaNet performance). We expect these numbers to substantially drop when using single-board computers like, for instance, the Nvidia Jetson Nano or Nvidia Jetson TX2, but, nevertheless, optimized versions of the Tiny YOLO network can achieve real-time performance on this kind of embedded systems [24].

YOLOv3-tiny model has 24 layers, most of them convolutional and max-pooling layers. It also contains two route layers that concatenate outputs from previous layers and merge them into one layer, thus mixing feature maps from earlier layers with semantic information from upsampled features. In the end, YOLO predicts a 3D tensor with the bounding box coordinates, the class predictions and the corresponding confidence score, indicating the probability that an anchor box contains an object predicted by the YOLO network. Although this network can be trained to detect multiple class objects, we will use it to predict only the "Ship" class.

Additionally, by adjusting the detection threshold so that only objects detected with a confidence score above that value are considered, we can tune the network sensitivity: for high threshold values the network will only detect objects with a high confidence score, potentially missing some target objects on the image, while for low threshold values the number of false positives will increase, potentially misidentifying ocean or coastal regions as a ship.

FIGURE 3. Example of possible detection scenarios: PR and GT bounding boxes in red and green, respectively. Top-left: The ship is there and the network detects it with IoU above the threshold; top-right: The ship is there but the IoU is below the threshold; bottom-left: there is no ship but the model detects one; bottom-right: the network does not detect the ship in the image.

The evaluation metric for the detection stage resorts to the standard intersection over union (IoU) score, given by

$$\text{IoU} = \frac{\text{PR} \cap \text{GT}}{\text{PR} \cup \text{GT}}, \tag{1}$$

where PR is a bounding box predicted by the YOLO detector for the Ship class and GT is a ground-truth bounding box. For a given annotated image, a detection is considered a true positive (TP) if a ship exists in the image and IoU > 0.5 for any ground-truth bounding box, otherwise it is considered a false positive (FP). A false negative (FN) occurs if a ship in the image is not detected, *i.e.*, if IoU < 0.5 for all the detections provided by the YOLO network for this particular ground-truth bounding box. In Fig. 3 we represent examples of TP, FP, and FN situations in maritime surveillance images: keep in mind that in one image it is possible to have multiple TP, FP and FN occurrences.

The precision of the detector is given by

$$\text{Precision} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}},$$

where the number of true positives and false positives is evaluated over a given set of images, while its recall is equal to

$$\text{Recall} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}}. \tag{2}$$

The detection threshold value chosen for the YOLO network influences these two metrics: if this threshold is low the detector will correctly identify almost all ships in the image, but the number of false positive cases will also increase, thus lowering the detector precision. On the other hand, increasing the threshold value will lower the false positive rate but will increase the probability that a target is missed, thus lowering the detector recall.

In this cascade architecture the goal of the detection stage is to discard large regions of the image, thus narrowing down

the size of the bounding boxes presented to the segmentation stage. This means that we want to have a detection stage with a very high recall, so that no ship goes undetected. To achieve such high recall a low detection threshold is desired: even if such approach results in a high number of false positives this is not a serious issue, as these bounding boxes will still be processed by the segmentation stage. The choice of the detection threshold will be discussed in the following sections.

B. SEGMENTATION NETWORK

After the detection stage the segmentation stage provides a pixel-wise classification of the candidate bounding boxes into two distinct labels: Background and Ship. Although other state-of-the-art segmentation networks exist, like Mask R-CNN [25], we choose to implement the U-Net architecture [12], as it has fewer parameters to train, allowing the training process to achieve competitive results using smaller datasets. Also, using a lighter network allows a faster segmentation during real-time operation.

For the encoder part of the U-net, one can apply standard convolutional operations for feature extraction or one can apply an existing backbone from other detection CNNs, like VGG [26], ResNet [7], or DenseNet [27]. In this work we choose this later option and we will study five different architectures for the backbone of the U-Net: VGG [26], ResNet [7], SEResNet [28], DenseNet [27], and InceptionResNet [29]. In the following sections experimental results will be provided regarding the assessment of the performance of these backbones in the context of ship segmentation using maritime surveillance images.

To assess the segmentation results, we compare, pixelwise, the ground-truth masks for the Ship class with the segmented masks provided by the U-Net for the same class, calculating the corresponding IoU score. By defining a segmentation threshold over the IoU value we can obtain the number of true positives, false positives and false negatives over a set of annotated images, and thus calculate the precision and recall values for the segmentation stage. Additionally, to compare the obtained results with those provided by the Kaggle Airbus ship detection challenge [19], we also calculate the F_β score, given by

$$F_\beta = \frac{(1+\beta^2) * \mathrm{TP}}{(1+\beta^2) * \mathrm{TP} + \beta^2 * \mathrm{FN} + \mathrm{FP}} \,. \tag{3}$$

The F_β score is the harmonic mean of precision and recall, and having $F_\beta = 1$ means that both precision and recall are equal to one. This metric is helpful when we are learning from imbalanced data, as its parameter β can balance precision and recall. The value of β defines how many times the recall is more important than precision: with $\beta > 1$, for instance, F_β score weights recall higher than precision.

The detection network presented in this paper is designed to have a high recall, providing to the segmentation stage many bounding boxes with just background pixels; additionally, the dataset used in the Kaggle Airbus ship detection challenge is highly unbalanced, with a lot of images without ships. This competition uses the F_β score, with $\beta = 2$, calculated at different IoU thresholds as a metric to evaluate and score the different segmentation algorithms competing in this challenge. To obtain a final score, the threshold value is varied from 0.5 to 0.95, with a step size of 0.05, and the arithmetic mean of the F_2 score for these different threshold values is obtained, *i.e.*,

$$\text{Kaggle Score} = \frac{1}{\#T} \sum_{t \in T} F_2(t), \tag{4}$$

where T represents the set of all thresholds for which the F_2 score is calculated.

C. CONDITIONAL RANDOM FIELDS

In complex segmentation scenarios, where small or partially occluded objects are present in the image, U-Net and other CNN based segmentation networks can output a very coarse segmentation boundary between different classes. To overcome this problem we can use post-processing techniques over the results provided by the segmentation stage like Conditional Random Fields (CRF), a very popular graphical model used in computer vision [30].

A CRF uses neighboring context, such as the relationship between adjacent pixels to assign a class to a given pixel. This model considers a cost, to be minimized, that depends both on each pixel prediction class probability (unary potential), as given by the segmentation stage, and on the similarity of predictions between pairs of pixels (pairwise potential), to enforce a correlation between the pixel predictions given by the previous segmentation stage, thus leading to the smoothing of the segmented objects boundary. For the pairwise potential we can use the similarity between all pixels in the image (dense CRF) or, alternatively, we can consider only the similarity between adjacent pixels (grid CRF).

In [30], using grid CRF leads to over-smoothing around the object boundaries, while it is shown that dense CRF can recover narrow edges in the image, thus improving the segmentation performance. In our work we implement the dense CRF algorithm, as it allows long-range interactions between pixels, no shrinking bias, and a probabilistic interpretation for segmentation. We follow the dense CRF model detailed in [30], that uses the energy function given by

$$E(x) = \sum_i \theta_i(x_i) + \sum_{ij} \theta_{ij}(x_i, x_j) \tag{5}$$

as a cost to be minimized, where x represents the set of pixel labels for all the pixels in the image and x_i and x_j represent labels for individual pixels i and j, respectively. The first term on the right side represents the unary potential, where we have

$$\theta_i(x_i) = -\log p(x_i), \tag{6}$$

with $p(x_i)$ corresponding to the label probability at pixel i provided by the segmentation stage. The second term in Equation 5 denotes the pairwise potential, given by

$$\theta_{ij}(x_i, x_j) = \mu(x_i, x_j)\left(w_1 K_1(i,j) + w_2 K_2(i,j)\right), \tag{7}$$

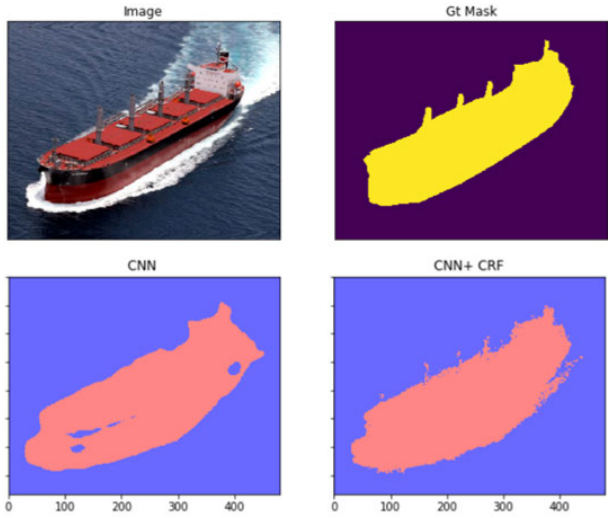


FIGURE 4. Upper left: original image; upper right: ground truth segmentation; lower left: segmentation using U-Net; lower right: segmentation using U-Net and CRF.

FIGURE 5. Example images from the Seagull dataset [21] and corresponding detection bounding boxes (in red).

where $K_1(i,j)$ and $K_2(i,j)$ are Gaussian kernels that measure the similarity between pixels i and j, w_1 and w_2 are the corresponding weights, and where $\mu(x_i, x_j)$ is an indicator function, denoted label compatibility, that only considers pairs of pixels that have different class labelings when calculating the cost. It is given by

$$\mu(x_i, x_j) = \begin{cases} 1 & \text{if } x_i \neq x_j\,, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

$K_1(i,j)$ is the appearance kernel, given by

$$K_1(i,j) = \exp\left(-\frac{\|p_i - p_j\|^2}{2\sigma_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\sigma_\beta^2}\right), \tag{9}$$

which depends on pixel positions and color, denoted by p and I respectively, and assumes that nearby pixels with similar colors are likely to be in the same class. The parameters σ_α and σ_β, respectively, control the degrees of nearness and similarity in this kernel.

$K_2(i,j)$ is the smoothness kernel that discourages labeling very small isolated regions of the image as the same class. It is given by

$$K_2(i,j) = \exp\left(-\frac{\|I_i - I_j\|^2}{2\sigma_\gamma^2}\right), \tag{10}$$

where σ_γ is a pixel similarity parameter.

Fig. 4 illustrates the potential benefits of using CRFs as a post-processing stage for ship segmentation on aerial images.

Note that the CRF model used in this work has 5 parameters that need to be set beforehand: w_1, w_2, σ_α, σ_β and σ_γ. These parameters values are defined using the available maritime images, as will be explained in the following sections.

IV. EXPERIMENTS

A. DATASETS

Machine learning models require a diversified set of training data to create a robust algorithm capable of segmenting vessels in multiple maritime scenarios. We use a combination of images from two distinct datasets: Seagull [21] and Kaggle [19]. The Seagull dataset contains RGB and multispectral video sequences acquired from a fixed-wing drone surveying maritime areas at altitudes around 150-300m, labeled with frame-based bounding boxes (Fig. 5). The Kaggle dataset, on the other hand, is composed of single-frame aerial images from different sources, with pixel-based labels, *i.e.*, segmentation masks (Fig. 6).

To train the detection network we selected 90 000 images, 50% from the Seagull dataset (RGB images only, from several videos, all with bounding box annotations) and 50% from the Kaggle dataset (the first 45 000 images from the training folder, which has pixel-level annotation). Since Kaggle has pixel-level labels, we had to create a bounding box to wrap the segmentation mask. For testing, we used 197 and 200 images from the Seagull and Kaggle datasets, respectively. Again, we had to generate the ground truth bounding boxes for the Kaggle images.

To train the network for the segmentation stage we selected a total of 20 000 images: 19 800 images from the Kaggle dataset and 200 images from the Seagull dataset. In this latter dataset, we had to manually create the ground truth mask for the images. This was done with LabelMe: Image Polygonal Annotation with Python [31], a graphical image annotation tool written in Python capable of creating semantic and instance annotations.

Since the input to the segmentation stage consists in image patches delimited by the bounding boxes provided by the previous detection stage, we passed the aforementioned 20 000 images through the detection network and then we used the resulting bounding boxes to train the U-Net with

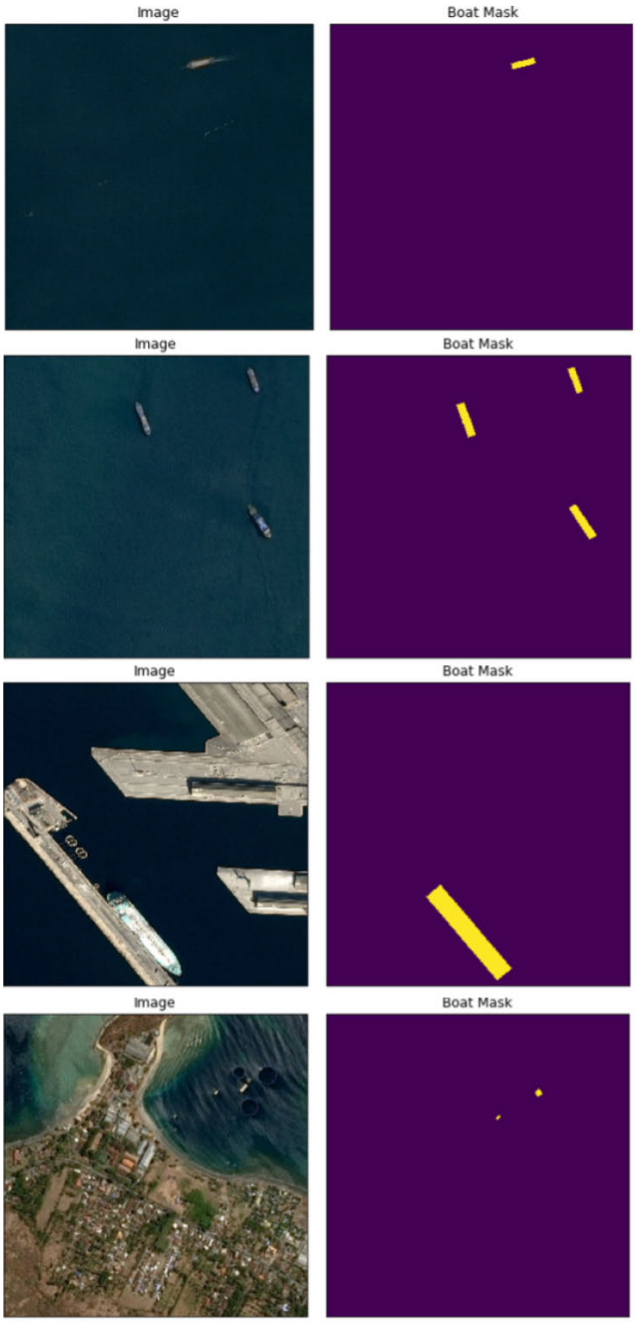


FIGURE 6. Example images from the Kaggle dataset [19] and corresponding segmentation masks. Left column: original image. Right Column: ground truth segmentation mask.

the ground truth labels. To evaluate the segmentation stage, we selected images from the Kaggle and Seagull datasets and built a local test set, where, as before, we had to manually annotate the Seagull images to provide the ground truth segmentation mask. Table 2 summarizes the set of images used for training and testing.

B. DETECTION STAGE

For the detection task we use the tiny-YOLO network from darknet, an open-source framework written in C and CUDA, which supports GPU and CPU computation [32].

TABLE 2. Training and test sets for the segmentation network. # Images denotes the total number of images in each dataset and # BB is the corresponding number of bounding boxes obtained after presenting these images to the detection stage.

Set ID	Dataset	Type	# Images	# BB
T1	Kaggle	Train	19 800	13 112
T2	Seagull	Train	200	710
S1	Kaggle	Test	100	382
S2	Seagull	Test	77	494

We use a pre-trained YOLO model, with a set of convolutional weights that result from the pre-training of the network using the ImageNet dataset. Since this pre-trained model is capable of detecting multiple objects categories, we replace the final layer with a simple binary output, since in our architecture we only intend to detect objects belonging to the Ship class. After that we fine tune the network, training this model with the aforementioned training images from the Seagull and the Kaggle datasets, keeping 70% of the 90 000 images for training the network and using the remaining 30% for validation. We use a batch size of 24, the sum of square error loss, and a learning rate of 0.001. We do not freeze any layer in the model during the training phase.

C. SEGMENTATION STAGE

For the segmentation task we used a recent implementation of U-Net where it is possible to customize some parameters like the loss function and the encoder architecture [33]. Since the input to the segmentation stage consists in image patches provided by the detection stage, there is a higher likelihood of an image pixel presented to the segmentation stage to belong to the Ship class, as most of the background pixels are discarded in the detection stage. The training set for the segmentation stage is different from the detection training set, to prevent biases in the detection stage from influencing the input distribution in the segmentation stage.

We passed the 20 000 images of the segmentation training set through the YOLO network, to obtain the bounding boxes for training the segmentation network. As a result, we got 61 775 bounding boxes, with only 6911 boxes having a ship. To create a balanced training set, we selected all the bounding boxes containing a ship and randomly picked another 6911 bounding boxes containing just background. This resulted in a training set with 13 822 bounding boxes. Then, the set was splitted in training and validation with a 70-30% ratio, respectively. During the training stage, we used a learning rate equal to 0.001 and the Adam optimizer. We trained the network during 50 epochs and we used the IoU as an evaluation metric.

During the training phase, we tried three different loss functions in the segmentation network: the dice loss, the cross-entropy loss, and the focal loss.

The dice loss is given by

$$\mathrm{DL} = 1 - (1 + \beta^2)\frac{\mathrm{Precision} * \mathrm{Recall}}{\beta^2 * \mathrm{Precision} + \mathrm{Recall}} \tag{11}$$

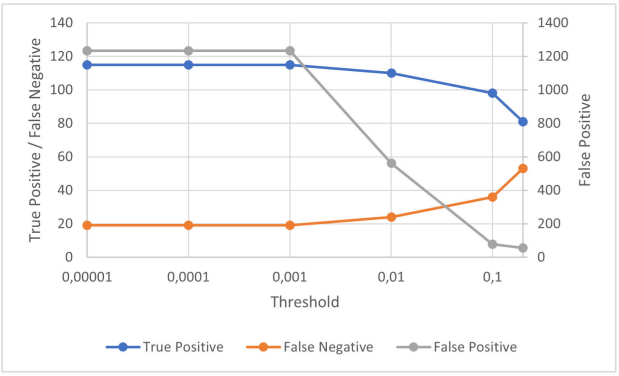


FIGURE 7. Number of false positive, false negative and true positive detections for the Seagull dataset as a function of the detection threshold.

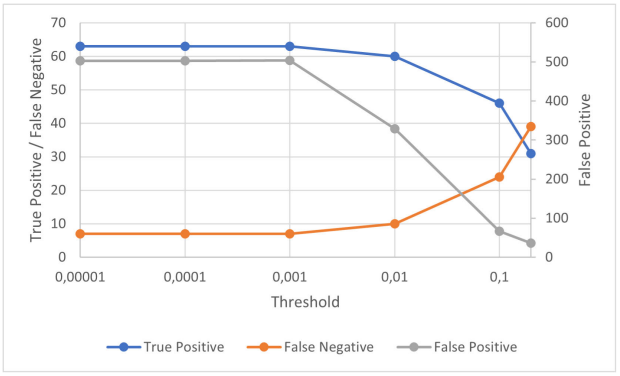


FIGURE 8. Number of false positive, false negative and true positive detections for the Kaggle dataset as a function of the detection threshold.

and in our work we set $\beta = 1$. The standard cross-entropy loss, on the other hand, is calculated pixelwise and is given by

$$\text{CEL} = \begin{cases} -\log p_x & \text{if } x = \text{Ship}, \\ -\log(1 - p_x) & \text{otherwise.} \end{cases} \tag{12}$$

Here x denotes the true class label for a particular pixel and p_x denotes the predicted probability that such pixel belongs to the class Ship, calculated by the segmentation network. The focal loss is a variation of the cross-entropy loss that addresses the class imbalance problem, down-weighting well classified examples so that in the training stage an increased focus on hard to classify pixels is achieved. It is given by

$$\text{FL} = \begin{cases} -\alpha(1 - p_x)^\gamma \log p_x & \text{if } x = \text{Ship}, \\ -(1 - \alpha)p_x^\gamma \log(1 - p_x) & \text{otherwise,} \end{cases} \tag{13}$$

where α and γ correspond respectively to a weighting factor and to a focusing parameter: based on [23] we use $\alpha = 0.25$ and $\gamma = 2$.

To assess the influence of the segmentation backbone in the overall performance, we trained the U-Net with several encoders: VGG, ResNet, SEResNet, DenseNet, and InceptionResNet. All of them are pre-trained in the ImageNet dataset [34] and available in [33]. Additionally, we also varied the batch size during the training phase to check its influence on the final evaluation score and on the training time.

D. POST-PROCESSING STAGE

We implemented a dense CRF model and evaluated its contribution as a post-processing stage to the overall segmentation performance. In this situation, the segmentation network output probability map is provided to the CRF model, which tries to minimize the energy function presented in Equation (5), given the pixel prediction values and corresponding locations. Due to the cascaded implementation proposed in this work, this operation has a reduced computational cost, since it is only performed on the bounding boxes provided by the detection stage and not on the entire image. As a final post-processing stage the individual segmented bounding boxes are aggregated to provide the final full segmented image.

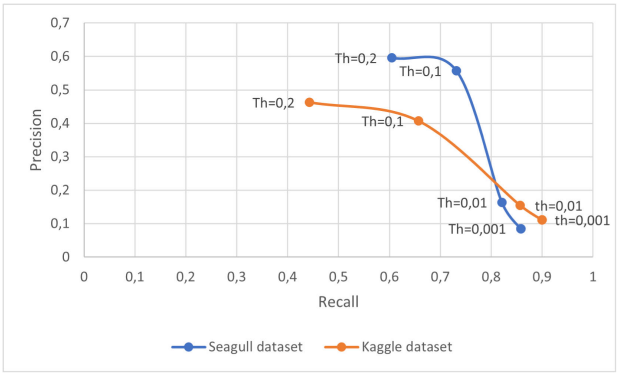


FIGURE 9. Detection precision-recall curves for the Seagull and Kaggle datasets.

V. RESULTS

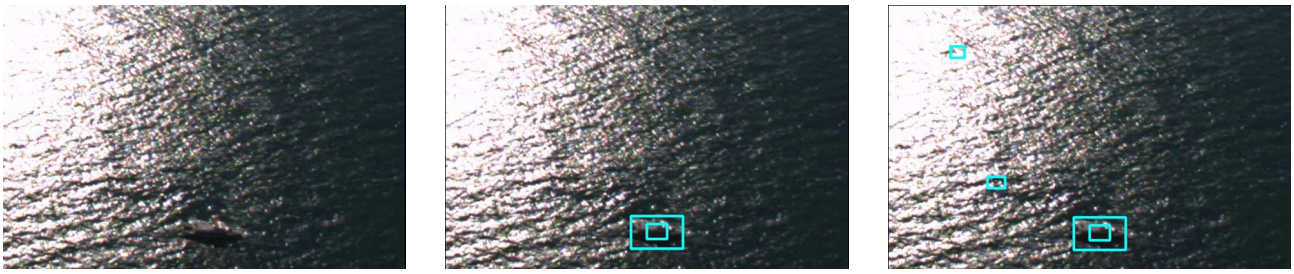
A. DETECTION STAGE

Since the segmentation stage is computationally demanding, in the detection stage we filter out unpromising image regions to reduce the size of the images presented to the segmentation stage, thus decreasing the overall image computation time. After training the YOLO network we tested it on both datasets for detection thresholds Th in the set $\text{Th} \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 0.2\}$. The number of false positive, false negative and true positive detections for the Seagull and Kaggle test datasets as this threshold is varied is presented in Fig. 7 and Fig. 8, respectively.

The corresponding precision-recall curves for both datasets are presented in Fig. 9, where we ommited the values corresponding to $\text{Th} \in \{0.00001, 0.0001\}$ for clarity, as they are very similar to the values calculated for $\text{Th} = 0.001$.

As discussed before, in the detection stage a high recall is desired, even if at a cost of generating a large number of false positives, as these false detections will have a high chance of being discarded at the segmentation stage. Fig. 10 illustrates the results obtained in the detection stage using three different threshold values.

Notice that, for a high detection threshold value, the ship in the lower part of the image would remain undetected and, as a consequence, the corresponding region of the



(a) Th = 0.2. (b) Th = 0.01. (c) Th = 0.001.

FIGURE 10. Detection network output for different threshold values.

TABLE 3. Detection network results with Th = 0.001.

Dataset	Recall	True Positive	False Negative	False Positive
Kaggle	0.9	63	7	504
Seagull	0.86	115	19	1233

image would not be processed by the segmentation stage. On the other hand, for very low values of this threshold, the YOLO network generates some false positive detections. This however, is not problematic, as these detections will still be processed by the segmentation stage.

Taking into account these results we set the detection threshold at 0.001: the corresponding detection results are summarized in Table 3.

Despite the low precision value — a high number of bounding boxes is provided by YOLO — we are able to discard between 95–96% of the background image, thus effectively greatly reducing the computational burden on the segmentation stage.

B. SEGMENTATION STAGE

To evaluate how the choice of a particular encoder in the U-Net influences the performance in the segmentation stage, we we tested different encoder backbones: VGG, ResNet and SEResNet. Table 4 shows the average IoU of the cascaded model (detection + segmentation) on the segmentation test sets of Kaggle and Seagull images (S1 and S2, respectively). Although the differences are not very significant, the ResNet-type networks seem to present a better overall performance, and ResNet-18 shows the best IoU score in both datasets.

In the following tests we selected the ResNet-18 network as the encoder of the segmentation network and searched for the best training loss function and batch size combination for this network. We tested the loss functions presented in Section IV.C both individually and in pairs, by averaging the corresponding scores. For the dice loss we also tried a weighted version where positive and negative examples are weighted differently, in order to balance the datasets – the weight is computed by the ratio of the number of pixels of each label in the training dataset. The tested batch size

TABLE 4. Segmentation IoU for multiple encoders with focal and dice losses, and batch size = 8.

Encoder	IoU	
	S1 (Kaggle)	S2 (Seagull)
VGG-19	0.81	0.91
ResNet-18	**0.91**	**0.94**
ResNet-34	0.91	0.93
SEResNet-18	0.90	0.93
SEResNet-34	0.87	0.92

TABLE 5. Airbus ship detection challenge score for the cascade model with the encoder Resnet-18 in the segmentation stage.

Loss function	Batch Size	Private score	Public score
FL and DL	8	0.79	0.61
FL and weighted DL	8	0.80	0.61
FL	16	0.79	0.62
DL	16	0.81	0.64
CEL and weighted DL	16	0.79	0.61
FL and weighted DL	**16**	**0.81**	**0.67**
FL and weighted DL	**24**	**0.81**	**0.67**

parameters were 8, 16, and 24. Notice that this study is not exhaustive and that we did not cover all the possible combinations of loss functions and batch sizes: instead, due to the limited computational resources available, we made a manually guided search, trying to change the parameters in order to achieve the best possible segmentation results in the Airbus ship detection challenge. Once again there is not an overwhelming difference in the results when we use different values for the loss function and the training batch size, but, nevertheless, Table 5 shows that the best results are achieved with a combination of the Focal and Weighted Dice losses, for large batch sizes. Fig. 11 presents some examples of segmented ships using the ResNet-18 encoder, trained with the average of Focal and Weighted Dice losses, in the segmentation network.

After selecting the encoder backbone, the batch size and the training loss function, we checked how it compared in the Airbus ship detection challenge to different encoder backbones. Additionally, we tried deeper encoders — ResNet-34, ResNet-50, DenseNet121 and InceptionResNetV2 — to see how encoder depth affected the performance of the global architecture. Table 6 presents the private score for six of

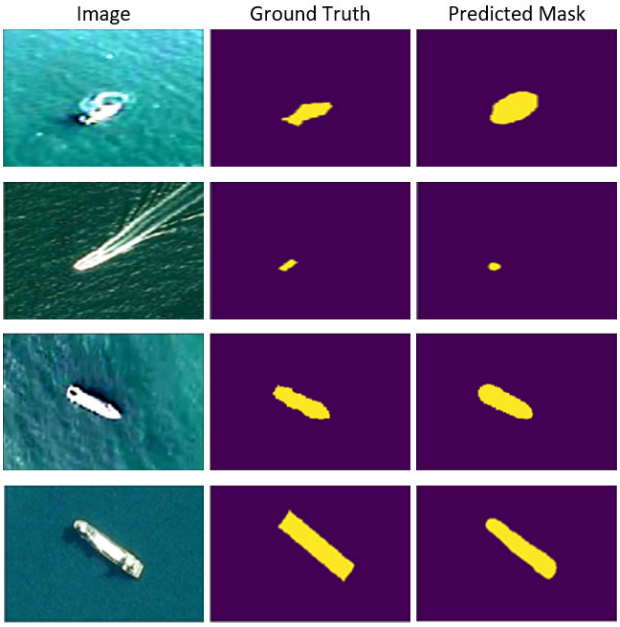


FIGURE 11. Segmentation results using the ResNet-18 encoder. Left column: original image. Middle column: ground truth segmentation mask. Right column: predicted segmentation mask.

TABLE 6. Airbus ship detection challenge results for multiple encoders using focal and weighted dice losses.

Encoder	Private score	Training time	Test time
ResNet-18	0.81	4h00min	1h20min
ResNet-34	0.80	5h13min	2h00min
ResNet-50	0.81	8h18min	2h10min
SeResNet-18	0.81	4h20min	1h37min
DenseNet121	0.82	11h36min	1h54min
InceptionResNetV2	0.82	13h00min	2h30min

these networks, together with the corresponding training and testing time. Although, as expected, the best score is achieved using Densenet121 and InceptionResNetV2 — the deeper encoders of the set of tested backbones — their training time is significantly higher when compared to the other networks. However, this difference is not so noticeable if we look at the test time: in the worst case scenario InceptionResNetV2 achieves a processing time of 2h30min for the full test set, against the 1h20min achieved by ResNet-18. In the following experiments we chose the DenseNet121 backbone for the segmentation stage, as it provides the best private score on the Airbus ship detection challenge at a decent processing time; keep in mind that this encoding network can be replaced, *e.g.*, by the ResNet-18 if the fastest processing time possible is desired.

C. POST-PROCESSING WITH CRF

To improve the segmentation accuracy we post-processed the segmentation probability map with a dense Conditional Random Field model whose energy function is given by Eq. (5), using a validation set of 200 images to tune its parameters in order to maximize the IoU score over that set. Parameters were optimized by grid search in the ranges: $\sigma_\alpha \in [1:5:100]$, $\sigma_\beta \in [1:1:20]$, $\sigma_\gamma \in [1:1:20]$, $w_1 \in [1:2:20]$, and $w_2 \in [1:2:20]$. The best values found were $\sigma_\alpha = 1$, $\sigma_\beta = 4$, $\sigma_\gamma = 3$, $w_1 = 1$, and $w_2 = 5$. Subsequently, we evaluated the results in the test sets S1 and S2. Table 7 presents the performance of the full system, comparing it with the results obtained without this post-processing stage. The final IoU score is marginally better when the post-processing stage is used, but this comes at a cost of an increased overall computation time. This will be discussed in the following section.

TABLE 7. Segmented bounding boxes IoU with and without CRF.

	IoU	
	S1 (Kaggle)	S2 (Seagull)
U-Net + CRF	0.941	0.910
U-Net (no CRF)	0.940	0.905

TABLE 8. Comparison of Cascaded models vs full image segmentation.

Method	Score	Time/image [s]
Full segmentation	0.71	1.47
Full segmentation + CRF	0.76	6.1
Detection + Segmentation	0.82	0.09
Detection + Segmentation + CRF	0.82	0.13

D. EVALUATION OF THE FULL CASCADED MODEL

To assess the computational advantages of the proposed cascade model, we compare it with the classical full image segmentation method. We trained a U-net for full image segmentation, using the same encoder and hyper-parameters as the ones used in the cascade model, and then evaluated its performance in the Airbus ship detection challenge. We also applied the CRF based post-processing step to the resulting output. Table 8 compares these different approaches with respect to the obtained private score in the Airbus ship detection challenge and the corresponding processing time. These results were obtained using a single GTX 1070 Ti graphics card, and show that the CRF post-processing stage can significatively improve the segmentation score when full image segmentation is performed (in line with results reported in [30]), but that such improvement is no longer visible when a cascade model is used. It also becomes aparent that the cascade model achieves the best results with respect to both segmentation performance and computation time; in particular, using the proposed cascaded model allows a real-time operation using UAV embedded hardware.

Fig. 12 shows some examples of segmented images taken from the Kaggle Airbus ship detection challenge dataset, using the proposed cascade model, where we can qualitatively notice that there is no significant difference when the CRF post-processing stage is performed. We also noticed that the cascade model segmentation accuracy sometimes drop when the ships are located near land (*e.g.* in harbours),

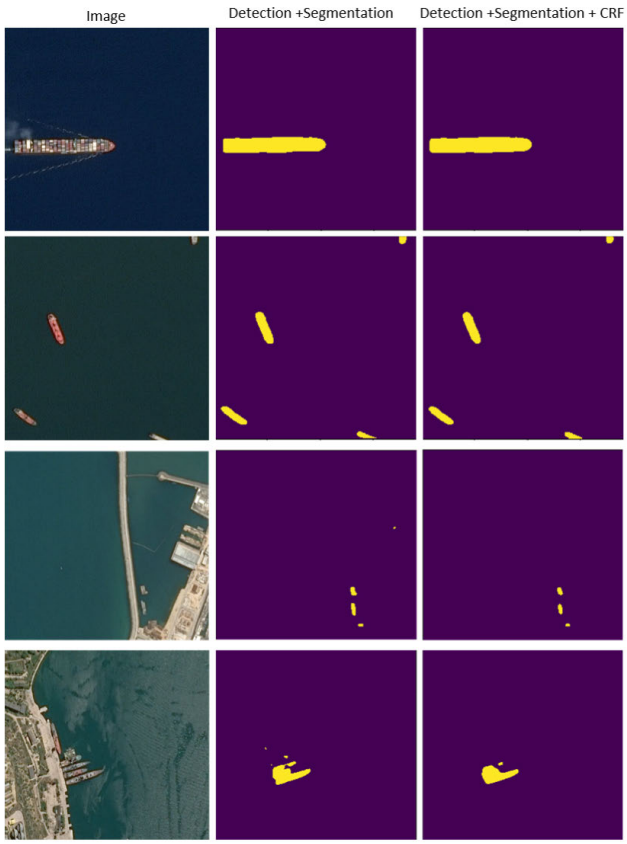


FIGURE 12. Segmentation using the proposed cascade model. Left column: original images. Middle column: segmentation without CRF post-processing. Right column: segmentation with CRF post-processing.

in which case sometimes it is difficult to distinguish the land background from the ship. This, however, is not a critical issue, as maritime surveillance missions typically occur in open sea scenarios.

According to [19], the winner of the Airbus ship detection challenge reached a private score of 0.85, against the 0.82 score achieved by our method. However, note that inference time was not taken into account to evaluate the performance of the algorithms competing at the Airbus ship detection challenge: since the proposed cascaded approach is intended to run on an UAV embedded hardware, real-time ship segmentation is the critical constraint in the proposed approach. Even if the average processing rate of 11 images per second achieved on a single GTX 1070 Ti graphics card will significantly drop on an embedded card like the Jetson Nano or Jetson TX2, this framerate, nevertheless, is still compatible with the maritime surveillance requirements.

VI. CONCLUSION

This work presents a new algorithm to perform fast and accurate maritime ship segmentation. The goal is to develop an autonomous system capable of doing maritime surveillance by using UAVs with onboard cameras. In the acquired maritime images the ships tipically represent a small portion of the image, and it can be challenging to recognize them when there are other elements such as waves, sun reflections, and ship wakes. Additionally, since all the computations are to be performed on onboard hardware with limited computational power, the final segmentation model must be fast enough to run on an embedded processor. To deal with these challenges we propose a cascade model with a detection and a segmentation stage: in the first stage we extract possible ship location regions, and then, in the following stage, we perform ship segmentation on these candidate regions. The initial stage discards parts of the image considered irrelevant, *i.e.*, with a very small probability of containing a ship according to the detector network, which greatly speeds up the segmentation stage. Our experiments show that this procedure also leads to a better performance, when compared to full image segmentation, even in challenging conditions, such as sun-glare and waves. We also tested a post-processing stage based on a CRF model: although, in Table 8, the experimental results show an improvement on the performance of a full segmentation model, this does not happen in the proposed cascade model, which presents the best segmentation performance at the fastest processing time.

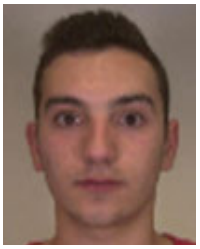
The annotated images from the Kaggle dataset used for training the segmentation network provide a rudimentar segmentation mask, simply consisting of rotated rectangles that fit the ships present in the image. We noticed that this issue may decrease the overall score obtained in the Airbus challenge, as the segmentation masks produced by the proposed method seem to provide a better approximation to the real ship contour than the rectangular mask used for evaluation — see, for instance, the last row of Fig. 11. Table 4 also seems to confirm this observation, since the evaluation of the proposed method on the Seagull dataset, where the ground-truth segmentation masks were manually annotated with a finer detail than those of the Kaggle dataset, leads to a better IoU score.

For future work we intend to study the use of temporal correlation with the previous video frames to improve ship segmentation. This can be done by introducing some sort of memory in the detection and segmentation stages (*e.g.*, by introducing Long Short-Term Memory (LSTM) layers in these networks) or by considering the time dimension in a post-processing stage (for instance, by using 3D CRFs [35]–[37] where the third dimension corresponds to time). This will require annotated sequences of images that the Kaggle dataset does not provide, which can make the training phase of a LSTM much more difficult due to the absence of this rich and diverse dataset. Furthermore, the processing time is expected to increase when this additional information is used, so a compromise must be made between segmentation accuracy and real-time processing requirements.

REFERENCES

[1] G. Pajares, "Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs)," *Photogramm. Eng. Remote Sens.*, vol. 81, no. 4, pp. 281–329, 2015.
[2] NVIDI. *Jetson Benchmarks*. Accessed: Jul. 6, 2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-benchmarks

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 91–99.
[4] J. Redmon and A. Farhadi, "YOLOV3: An incremental improvement," *CoRR*, vol. abs/1804.02767, pp. 1–8, Oct. 2018.
[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1–9.
[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.
[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2014, pp. 580–587.
[9] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2117–2125.
[10] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.
[11] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Jan. 2017.
[12] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Med. Image Comput. Comput.-Assist. Intervent.*, 2015, pp. 234–241.
[13] M. Kang, Z. Lin, X. Leng, and K. Ji, "A modified faster R-CNN based on CFAR algorithm for SAR ship detection," in *Proc. RSIP Workshop*, May 2017, pp. 1–4.
[14] X. Yang, H. Sun, K. Fu, J. Yang, X. Sun, M. Yan, and Z. Guo, "Automatic ship detection in remote sensing images from Google Earth of complex scenes based on multiscale rotation dense feature pyramid networks," *Remote Sens.*, vol. 10, no. 1, p. 132, Jan. 2018.
[15] G. Cruz and A. Bernardino, "Learning temporal features for detection on maritime airborne video sequences using convolutional LSTM," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 9, pp. 6565–6576, Sep. 2019.
[16] J. Matos, A. Bernardino, and R. ibeiro, "Robust tracking of vessels in oceanographic airborne images," in *Proc. Monterey*, 2016, pp. 1–10.
[17] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, no. 1, pp. 62–66, Jan. 1979.
[18] G. Cruz and A. Bernardino, "Evaluating aerial vessel detector in multiple maritime surveillance scenarios," in *Proc. OCEANS*, 2017, pp. 1–9.
[19] Airbus. *Airbus Ship Detection Challenge*. Accessed: Jan. 6, 2020. [Online]. Available: https://www.kaggle.com/c/airbus-ship-detection/overview
[20] V. R. and A. Metha, "Segmenting ships in satellite imagery with squeeze and excitation u-net," *CoRR*, vol. abs/1910.12206, pp. 1–12, Mar. 2019.
[21] R. Ribeiro, G. Cruz, J. Matos, and A. Bernardino, "A data set for airborne maritime surveillance environments," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 9, pp. 2720–2732, Sep. 2017.
[22] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 379–387.
[23] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.
[24] N. Ayoub and P. Schneider-Kamp, "Real-time on-board deep learning fault detection for autonomous UAV inspections," *Electronics*, vol. 10, no. 9, p. 1091, May 2021.
[25] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, May 2017, pp. 2961–2969.
[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015, pp. 1–5.
[27] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
[28] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, May 2018, pp. 7132–7141.
[29] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-V4, inception-resnet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artif. Intell. (AAAI)*. AAAI Press, 2017, pp. 4278–4284.
[30] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected CRFS with Gaussian edge potentials," *CoRR*, vol. abs/1210.5644, pp. 1–5, Jun. 2012.
[31] K. Wada. (2016). *LABELME: Image Polygonal Annotation with Python*. [Online]. Available: https://github.com/wkentaro/labelme
[32] J. Redmon. (2013). *Darknet: Open Source Neural Networks in C*. [Online]. Available: http://pjreddie.com/darknet/
[33] P. Yakubovskiy. (2019). *Segmentation Models*. [Online]. Available: https://github.com/qubvel/segmentation_models
[34] J. Deng, W. Dong, R. Socher, and L. Li, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Dec. 2009, pp. 248–255.
[35] K. Kamnitsas, C. Ledig, V. F. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert, and B. Glocker, "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation," *Med. Image Anal.*, vol. 36, pp. 61–78, Oct. 2017.
[36] M. Monteiro, M. A. T. Figueiredo, and A. L. Oliveira, "Conditional random fields as recurrent neural networks for 3D medical imaging segmentation," 2018, *arXiv:1807.07464*.
[37] S. Chen and M. de Bruijne, "An end-to-end approach to semantic segmentation with 3D CNN and posterior-CRF in medical images," in *Proc. Conf. Neural Inf. Process. Syst.*, Montréal, QC, Canada, 2018, pp. 1–4.



CARLOS PIRES received the M.Sc. degree in electrical engineering from the Instituto Superior Técnico, Lisbon, Portugal, in 2019. His research interests include machine learning and computer vision applied to robotic systems, with a strong emphasis on embedded systems and real-time maritime surveillance.

BRUNO DAMAS received the Ph.D. degree in electrical engineering from the Instituto Superior Técnico (IST), Lisbon, Portugal, in 2014. He is currently an Assistant Professor with the Portuguese Naval Academy and is also a Researcher at CINAV, the Portuguese Naval Research Center, and the Computer and Robot Vision Laboratory, Institute for Systems and Robotics, IST. His main research interests include machine learning, robotics, and computer vision, with a particular focus on real-time learning, online algorithms, reinforcement and self-supervised learning, autonomous underwater and surface vehicles, and maritime applications of autonomous robots.

ALEXANDRE BERNARDINO (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the Instituto Superior Técnico (IST), Lisbon, Portugal, in 2004. He is currently a tenured Associate Professor with the Department of Electrical and Computer Engineering and a Senior Researcher with the Computer and Robot Vision Laboratory, Institute for Systems and Robotics, IST. He has published over 70/210 research papers in peer-reviewed international journals/conferences in the field of robotics, machine learning vision, and cognitive systems, with more than 5900/3300 citations and H-index 35/23 (Scopus/Google Scholar). He has graduated 19 Ph.D. students and more than 120 M.Sc. students. He participated in more than 20 national and international research projects, being the principal investigator in five of them. He is the Chair of the IEEE Portugal Robotics and Automation Chapter, from 2015 to 2019. His main research interests include the application of computer vision, machine learning, cognitive science, and control theory to advanced robotics and automation systems.