Advanced Topics in Deep Learning

Instance segmentation of maritime environments for the autonomous navigation of vessels.

Spring Semester 2018

Adrian Llopart Maurin

Technical University of Denmark
Department of Electrical Engineering

Contents

Chapter 1

Introduction

Current trends in A.I and Deep Learning have focused on the autonomous navigation of cars on roads thanks to the huge investment of big companies such as Tesla, Toyota, Google, Facebook, etc . This means receiving sensory feedback from sensors mounted on the cars, analyzing the data, taking decisions and acting accordingly (Fig. 1.1). So far, this has become a mainstream topic with multiple solutions and approaches presented almost every day, specially in the automobile world. However, it is uncommon to find similar solutions for other types of vehicles or environments.

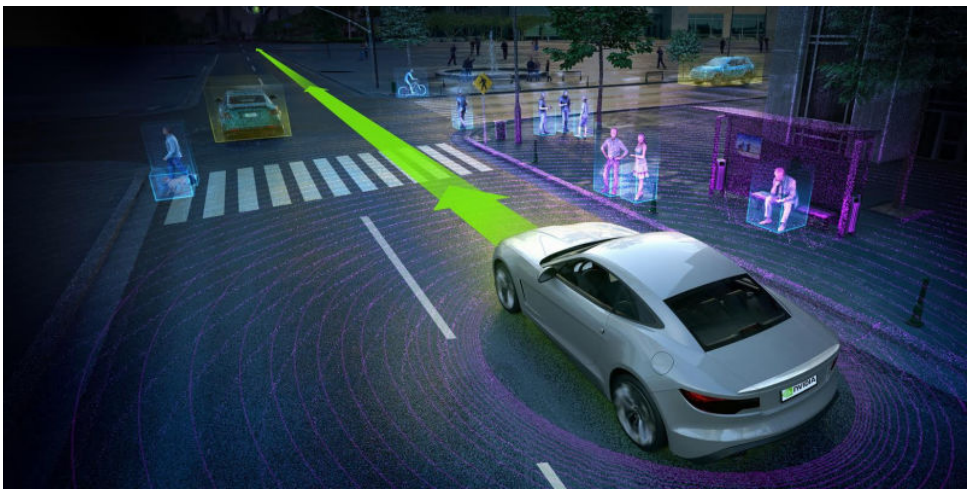

Figure 1.1: Concept art of the autonomous navigation of an on-road car

The presented work helps transfer the idea of vehicle automation to ships and other maritime vessels. This work tries to provide a state-of-the-art solution to the *Electronic Outlook* project, by Prof. Mogens Blanke from DTU Elektro, which has received funding from the Danish Maritime Authority. It will focus on semantically segmenting images captured by an on-board camera, to be able to detect and act upon its surroundings. Special emphasis is put on detecting neighboring ships, buoys and land, to avoid collisions and navigate safely.

Having segmented certain classes from an RGB image, it becomes easier to track moving obstacles and avoid them, or categorize types of buoy and understand what they mean (maritime buoys have different shapes and colors that signal how vessels should interact with them). Image segmentation will also be helpful for some other smaller sections of this project, for instance buoy categorization or estimation of the horizon line. The vessel will have other mounted sensors like Infrared (IR) cameras and a radar for when the visibility is low (i.e. fog or low light conditions). With an appropriate sensor fusion system, the data received by the sensors can be combined and a better and more realistic *image* of the environment can be created.

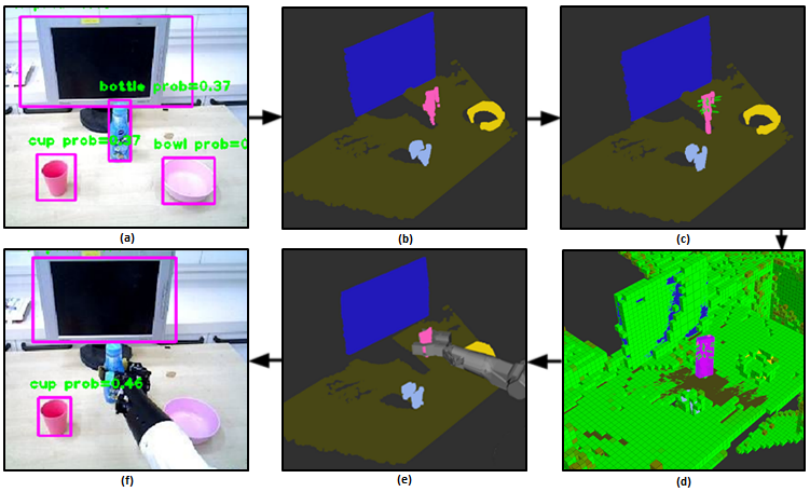

Figure 1.2: Procedure based on the YOLO network to detect object in real time. It needs a post processing algorithm to correctly extract the data from the objects.

1.1 Evaluation of Deep Learning architectures

The proposed work requires that, given a single RGB image, a computer is able to semantically segment per-pixel diverse classes it has been trained to detect and recognize, outputting that same image but with colors corresponding to each class layered on top.

Initially, though, being able only to generate bounding boxes around the classes seemed like a good idea; however, after implementing such a system, it was seen that the results were not as stable and optimal as expected. Llopart et. al. [1] proposes the interaction between robots and objects in a generalizable system that employs two main methods that work together: the former, the YOLO system ([2] and [3]), a CNN trained on the COCO dataset (80 classes) that generates bounding boxes in real time; and the latter, a segmentation procedure for pointclouds in the 3D space domain. Using these methods, a robot is able to detect, recognize and build 3D models of many objects (Fig 1.2). Despite the fact that this system is slightly different than the proposed maritime vessel project, a key idea could be extracted from it: in a real scenario, if a system needs to interact with its environment, a bounding box is not precise enough. Some post processing will be needed to be able to extract the contour of the object/class, loosing processing power, increasing the detection time and, generally, having lower performance.

Indeed, since the *Electronic Outlook* project started recently and there are several people collaborating in it, it has been proven that ship and buoy detection can be accomplished using only bounding boxes (with, actually, as few training images as 100 per class) and some post-processing algorithms (Fig 1.3). This has been achieved with the Matlab implementations of *Fast-RCNN* [4] and *Faster-RCNN* [5], which already shaped the path the presented work would follow.

The system *Mask-RCNN* [6] is selected as the best solution for real time semantic segmentation of images. The main idea, thus, is to be able to increase performance and boost the results compared to the bounding box concept already established, whilst still being able to function in real time.

Figure 1.3: Matlab CNN outputs bounding boxes around classes (ship and buoy)

Chapter 2

Mask-RCNN

Mask-RCNN is a continuation of the work proposed in *Fast-RCNN* and *Faster-RCNN*. This algorithm focuses on instance segmentation, that is, semantically segmenting an image per-pixel (Figure 2.1). The next sections describe briefly the previous work on which *Mask-RCNN* is cemented on, how the network models have changed over time and what results are to be expected for each case.

Figure 2.1: Mask-RCNN results based on RGB input images.

2.1 Preceding architectures: Fast-RCNN

Fast-RCNN is a network architecture that focuses on object detection from RGB images, that is, generating bounding boxes around recognized classes and displaying their name/label and probability. The model is composed of 3 main steps:

1. The input (RGB image) of the network is processed with a FCN (Fully Convolutional Network) to extract its full feature map. The feature extractors can be modified, but some popular examples are VGG and ResNet. Additionally, the generated feature map will be re-used for different regions of interest.

2. To find the Regions of Interest (RoI) that possibly contain objects, a Selective Search is applied. Neither *Faster-RCNN* nor *Mask-RCNN* use this concept though. By applying these generated RoIs to an RoI-Pooling layer, the corresponding parts of the full-feature map are extracted.

3. To obtain the class probability, the feature map of every RoI is fed into a Softmax layer since it produces

a probability output (from 0 to 1). The class with the highest probability is the label which that object will have. The bounding boxes are adjusted to achieve better results by calculating the bounding-box offset and then refining them. The architecture overview is shown in Fig. 2.2.

2.2 Preceding architectures: Faster-RCNN

Faster-RCNN follows the *Fast-RCNN* model precisely, with just one exception: it removes the Selective Search (from step 2) because it is one order of magnitude larger than the rest of the processes. By removing this algorithm, the network becomes much faster. To compensate for the removal of the Selective Search, the Region Proposal Network is added.

The Region Proposal Network creates bounding boxes by sliding an anchor point across the image and generating multiple rectangular proposals per anchor that vary in ratio and scale. Each proposal includes also the likelihood of an object being present. After filtering out those proposals with low probability, the remaining RoI are processed in the same fashion as *Fast-RCNN*. Fig. 2.2 shows how the *Fast-RCNN* architecture changes to the *Faster-RCNN* by simply substituting the Selective Search with the Region Proposal Network.

2.3 Mask-RCNN overview

Mask-RCNN tries to push the boundaries of image segmentation to a new level: semantic instance segmentation. To achieve this, an additional output branch is added in parallel to *Faster-RCNN*. A Fully Convolutional layer processes the previously generated fixed-size RoI feature maps. This layer learns to upscale and segment portions of the ROIs through intensive training.

Additionally, the RoI-Pooling procedure from step 2 in *Fast-RCNN* is modified because it is not precise enough. The RoI-Align algorithm is used instead since it does not have quantization and, thus, does not create misalignment results on the boundaries. Both methods upgrade the *Faster-RCNN* architecture to the *Mask-RCNN* and are displayed in Fig. 2.2.

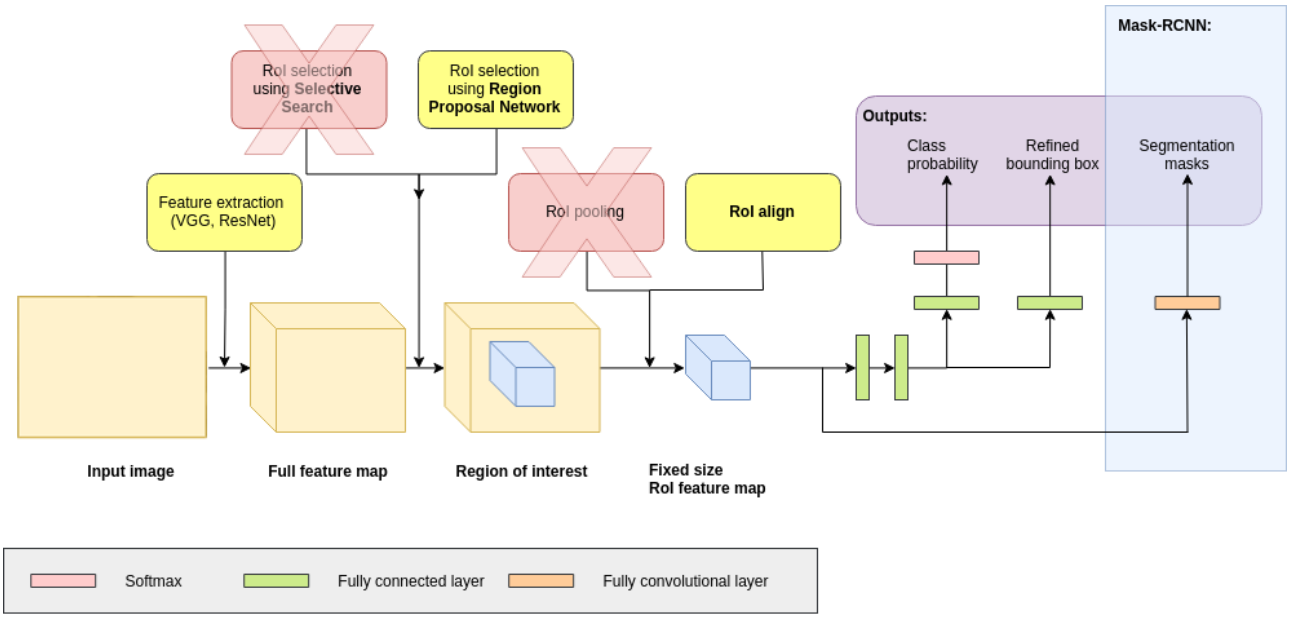

Figure 2.2: Overview of the evolution of the *Fast-RCNN*, *Faster-RCNN* and *Mask-RCNN* network models. *Faster-RCNN* simply changes the Selective Search algorithm for the Region Proposal Network. *Mask-RCNN* adds a parallel branch that takes the fixed sized RoI feature maps, processes them through a trained Fully Convolutional layer and outputs semantic instance segmentation; additionally, it substitutes RoI-Pooling with RoI-Align.
Courtesy of Szymon Prezemyslaw Kowalewski

Chapter 3

Dataset

The annotated images selected for training are almost as important as the Deep Learning architecture employed. For this specific case, no dataset was found that incorporated correctly all the required information. This is because *Mask-RCNN* is a supervised learning model that needs the masks and name labels of all classes from an RGB image for training.

Only one dataset was found, the *ARGOS-Venice* [7] dataset, but, even though it incorporated multiple types of boats from Venice, it did not have any information on the masks of the ships, ships that were meant for open sea sailing, buoys or land. Thus, this dataset was deemed not particularly useful.

This meant that the dataset had to be generated from scratch, which would be highly time-consuming. This includes gathering several hundred images, labeling correctly the classes and the masks and preparing the dataset for correct training. The new dataset will be called *DTU Maritime*.

3.1 Image gathering

At the time of making this project, the on-board cameras have not yet been mounted. This means all images were taken manually using a DSLR camera: a person would go on board of a ferry that cruises between Helsingor and Helsingborg (this is the original route the system is proposed for, in future iterations, routes in the south of Fyn will be taken into account). A couple hundred pictures were taken in the span of 3 days. This includes and extremely foggy day and two sky-clear sunny days. The difference between images of these days is appreciable and perfect to be able to train a more generizable system. However, as will be discussed in th results, the limited number of training images (176) and the fact that the images tend to show a recurrent landscape (its the same route over and over again), will make the system overfit.

3.2 Image labeling

The labeling of images was done using the *LabelMe* software that allows for easy labeling and the generation of a *.json* file that is necessary later for training. Even though the software is very simple to use, manually creating contour labels for up to 5 different classes (buoy, land, sea, ship, sky) is very tedious. Around 3 days were necessary to label 176 images. The process could have been crowdsourced using, for example, the Amazon Mechanical Turk (MTurk), but due to the limited number of images it did not make that much sense.

3.3 Dataset preparation

Prior to training, the image dataset has to be prepared. To start off with, a *.txt* file is created with the names of all the images that will be used for training. All images will be converted to the *.png* format, to simplify the data loading process.

As aforementioned, the *LabelMe* software generates *.json* files for every labeled image. These files have a lot of information in them, however, what is mostly important is the number of instances a specific class appears in each image, the class name and the polygon points that limit the mask of each instance. This information is essential to building masks of each class that appears in the image. A *JSON_parser.py* file is generated to adequatelly extract the information from the *.json* files. Depending on the software used to label the images, the

Figure 3.1: Hand-made label results of the images using the *LabelMe* software

.json file information setup might differ. Specifically, if the COCO dataset was used, the way the information is encoded in the *.json* file will be very different as the ones from *LabelMe*. Thus the parser should be modified.

Once the class and contour data is extracted, the file *loader.py* will generate one mask per instance, per class, per image. This mask will be a gray scale image, where all the pixels corresponding to a class will have the associated class number (from 1 to 5, since there are 5 classes) and the other pixels will have a value of zero (background). A way to view these masks is using the file *show_data.py* The number of instances for the whole *DTU Maritime* dataset is shown in table 3.1. Comparatively, the COCO dataset uses 35000 images for its up to 80 classes. Evidently, the results one can obtain with a tiny set of images like the ones from the *DTU Maritime* dataset, will be much poorer than those of the COCO dataset; in fact, a comparison might not even be possible. However, the idea behind this project is to be able to prove the concept of using *Mask-RCNN* on a brand new dataset with adequate results.

Class	Files containing class	Number of instances
Buoy	141	221
Land	147	171
Sea	176	181
Ship	45	74
Sky	162	162

Table 3.1: Number of instances per class in the whole dataset

3.3.1 Mean pixel value

Before the training occurs, *Mask-RCNN* requires the normalization of the network inputs. For this reason, *mean_pixels.py* calculates the mean values for each of the RGB channels, for all the set of training images. Normalizing allows for a better performance of the model.

Chapter 4

Results and discussion

The following chapter will showcase how to train the selected network on the *DTU Maritime* dataset and what conclusions can be extracted from the results.

4.1 Training

A new training configuration file is created called *Maritime.py*. Here, a series of parameters are tunned for better performance and overall good results. The used weights will have been pretrained on the COCO dataset. Since the images collected form the on-board cameras will be at a high resolution, the GPU used is a 1080 GTX with 16 GB of memory and following the recommendations of the *Mask-RCNN* programmers, the images_per_GPU will be set to 2. A value of 1000 steps per epoch is set and just one validation image is used because of the few images of the dataset. The number of classes will be 6: boy, land, sea, ship, sky and background (non-recognizable classes). Finally, a small learning rate of 0.001 is selected, with a learning momentum of 0.9.

One of the interesting features of *Mask-RCNN* is the ability to schedule the training, that is, train different parts of the model at different times. A good option, proposed for the *Mask-RCNN*, is training first the heads of the network (the first 3 layers), followed by a training of only layers 4 and up, and finally all he network together. Each of these 3 "levels" of training occurs for a set of epochs. It was found that, due to the small dataset available, the training always overfit and so it did not matter much how many epochs per level were selected. At the end, the values of 10 epochs for the first two levels and 20 for training the full model were set.

4.2 Experimental results

The Jupyter notebook code *maritime.ipynb* allows to generate the forward pass of the trained network on a single image. It shows the resulting bounding boxes and masks of each detected class. Figure 4.1 displays those results on images taken from the training set. Results are therefore expected to be extremely good. The fact that the masking corresponds practically to the ground truth and that the probabilities of all classes are 1.0 shows how the network is overfitting.

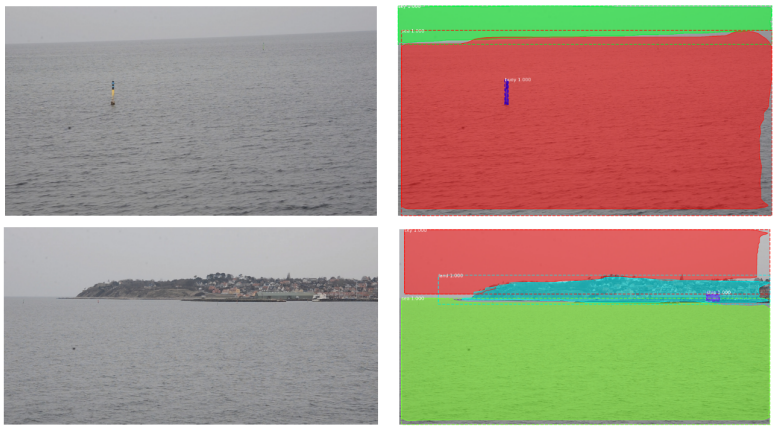

Figure 4.1: Forward pass of a couple of images (from the training set).

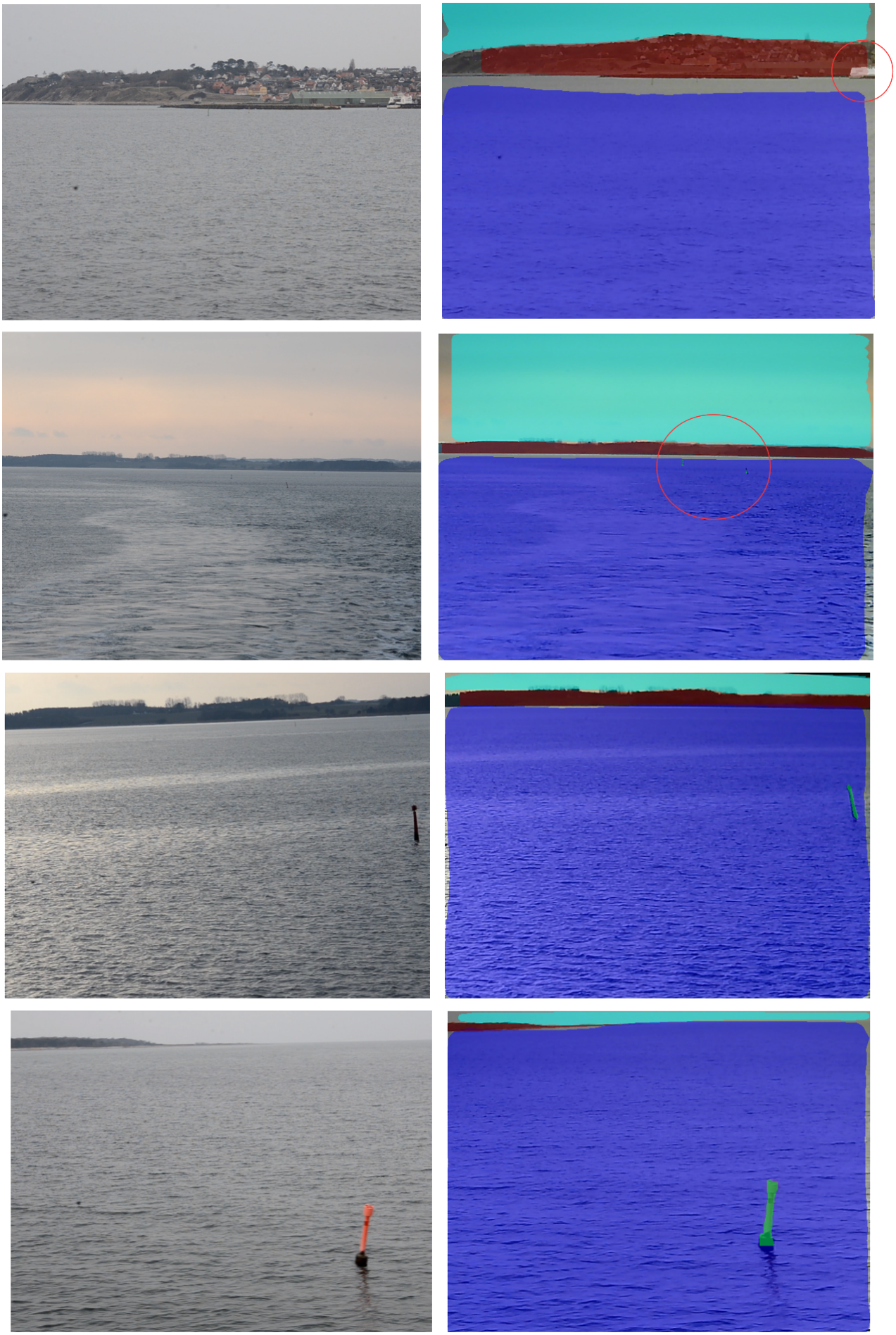

Figure 4.2: Final results of some frames from a video stream captured by a webcam

To test how well the system performed under real conditions, the forward pass of the network was applied to a video stream captured from a webcam (to simulate the on-board cameras). The colouring schemantic was set so that the sky was always seen with a clear blue; the land with red; the sea with dark blue; the buoys with green and the ships with white. Running *mask_rcnn_webcam.py* will obtain frames from the camera and process them instantly. It is soon seen that the performance of the system allows for real-time detection and recognition, since it works at a frequency of around 8 Hz. Average-wise, the time it takes for the system to grab a frame is 0.00045 seconds, to process it 0.115 seconds and to display the results 0.0055 seconds. Figure 4.2 shows some frames from videos taken at the same time and location as the training images. The results are, of course, very good. But, to be fair, videos and training images are very similar; thus, considering that the network overfits, the system is only applicable to environments close to the training set. Which, in turn, is alright, because the system is meant to work on a ferry that travels the same route over and over again. It is worth noticing how the system also detects ships that are stationary in the harbor. Even more interesting is the ability of the system to not only recognize and contour buoys; but that it can do so at a distance where even the human eye might have difficulties (marked with a red circle).

The systems capabilities are then tested in completely different images. In this case, a random navigation video from YouTube. Figure 4.3 showcases how the system clearly overfits. The network is capable of understanding that in most pictures the sea is under the land which in turn is under the sky. Sometimes the land is present, sometimes not (this also happens with the sky), but the sea is always there. This is why the coloring schematic almost always follows the rectangle combination of dark blue (sea) under red (land) under clear blue (sky). Notice that in these figures, some parts of the sky were incorrectly recognized as sea. The reason is that there exists not one single image in the training data that shows a clear sky with contrasting clouds on it. Therefore, the system recognizes the clouds as waves and thinks the sky is the sea. The system however, seems to predict land very accurately.

Figure 4.3: Results from a Youtube video where the landscape is different than the training data

4.3 Conclusions

Given the lack of training data and the fact the system clearly overfits, the results are to be expected. The system predicts correctly landscapes similar to the training data, but is not able to generalize much more than that. Recognizing clouds as waves is a very clear example of that. In spite of this, this work's goal was a simple proof of concept for the maritime image segmentation in the *Electronic Outlook* project; specifically, in recurrent maritime routes where the only dynamic variables are possible ships and weather conditions. Thus, by increasing the training dataset, one would expect more generalizable results for a wider variety of scenarios.

All code and videos can be found on: https://github.com/Allopart/Mask_RCNN [8] and https://www.youtube.com/watch?v=_vmKbbW1FuM

Bibliography

[1] A. Llopart, O. Ravn, N. Andersen, and J.-H. Kim, "Generalized Framework for the Parallel Semantic Segmentation of Multiple Objects and Posterior Manipulation," in *Proc. IEEE International Conference on Robotics and Biomimetics (ROBIO)*, (Macau, China), December 2017.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," June 2016.

[3] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," Dec. 2016. arXiv:1612.08242.

[4] R. Girshick, "Fast R-CNN," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, (Santiago, Chile), December 2015.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN," *Advances in Neural Information Processing Systems*, pp. 91–99, 2015.

[6] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, (Venice, Italy), October 2017.

[7] D. D. Bloisi, L. Iocchi, A. Pennisi, and L. Tombolini, "ARGOS-Venice Boat Classification," in *Proc. IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, (Karlsruhe, Germany), August 2015.

[8] "Forked github from mask-rcnn with added training files and results for the dtu maritime dataset." https://github.com/Allopart/Mask_RCNN.