

Informatik und Mathematik
Systems Engineering for Vision and
Cognition
Goethe-Universität Frankfurt



MLPR Group Project

Ship Detection on satellite images using neural networks

Kaggle Challenge: Airbus Ship Detection Challenge

Submission Date: October 16, 2018

Team members

Hesamedin Ghavami Kazzazi
Iurii Mozzhorin
Hevin Özmen

Supervised by:

Prof. Dr. Visvanathan Ramesh

Table of Contents

1	<i>Introduction</i>	1
2	<i>Methodology</i>	4
3	<i>Data Understanding</i>	5
4	<i>Pre-Classification</i>	6
4.1	Data Preparation	6
4.2	Data Modelling	7
4.3	Technical Specifications and Runtime	8
4.4	Data Visualization and Evaluation	9
4.4.1	Loss	9
4.4.2	Accuracy	9
4.4.3	Errors	10
5	<i>U-Net</i>	12
5.1	Data Preparation	12
5.2	Data Modelling	12
5.3	Technical specifications and runtime	14
5.4	Data Visualization and Evaluation	14
5.4.1	Loss	15
5.4.2	Accuracy	15
5.4.3	Kaggle Score	16
6	<i>Mask R-CNN</i>	16
6.1	Related work	17
6.1.1	R-CNN and paving the way towards the Mask R-CNN	17
6.1.2	Image Segmentation	19
6.2	Mask R-CNN Details	19
6.3	Data	21
6.4	Hardware Configuration	22
6.5	Framework Configuration	22

6.6	Data Preparation	23
6.7	Training	24
6.8	Detection	25
6.9	Evaluation	25
6.9.1	Loss	25
6.9.2	Kaggle Score	26
6.10	Summary	27
7	<i>Evaluation and Summary</i>	28
8	<i>References</i>	29
9	<i>Appendix</i>	VII
9.1	Code specifications for pre-classification	VII
9.2	Code specifications for U-Net	VIII
9.3	U-Net Architecture	IX
9.4	Mask R-CNN: Kaggle submission	X

List of Figures

Figure 1: Network architecture of CNN8.....	7
Figure 2: Loss (Cross-Entropy) over steps for balanced and unbalanced set.....	9
Figure 3: CNN_32 Unbalanced Loss (Cross-Entropy) over steps.....	9
Figure 4: CNN_8 Accuracy per epoch for balanced and unbalanced set.....	10
Figure 5: CNN_8 Errors per epoch for balanced and unbalanced set.....	10
Figure 6: CNN_8 Errors per threshold for balanced and unbalanced set.....	11
Figure 7: U-Net loss per epoch	15
Figure 8: Accuracy of U-Net Model B per epoch (1) True Positives, (2) Binary Accuracy and (3) Dice Coefficient.....	15
Figure 9: R-CNN Framework [7].....	18
Figure 10: Fast R-CNN Framework [3]	18
Figure 11: Faster R-CNN Framework [4]	19
Figure 12: Task of Computer Vision [10]	19
Figure 13: Mask R-CNN Framework [6].....	20
Figure 14: Example of Run-Length-Coding mask	22
Figure 15: Image with mask.....	23
Figure 16: Image with bounding boxes and respective mask	24
Figure 17: Images with anchors and RoIs	24
Figure 18: Prediction masks (sample).....	25
Figure 19: Loss (Cross-Entropy) for Training (left) and evaluation (right).....	25
Figure 20: Mask Loss (Cross-Entropy) for Training (left) and evaluation (right)	26
Figure 21: Class Loss (Cross-Entropy) for Training (left) and evaluation (right)	26
Figure 22: Bounding Box Loss (Cross-Entropy) for Training (left) and evaluation (right).....	26

List of Tables

Table 1: Confusion matrix for Preclassification.....	10
Table 2: Overview of runtime for prediction of entire test data and preclassified test data (thresholds 0.4 and 0.5)	14
Table 3: Overview of Kaggle submission files, the underlying U-Net model and achieved Kaggle Score.....	16

List of Abbreviations

FCN	Fully Convolutional Network
FPN	Feature Pyramid Network
IoU	Intersection over union
ReLU	Rectified Linear Unit
RPN	Region-based Proposal Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine

1 Introduction

In light of the increasing size and demand in the ship transport sector the need for maritime monitoring services becomes pressing. This ensures the reduction of ship accidents as well as the mitigation of illegal activities such as piracy, drug trafficking, illegal fishing and movement of illegal cargo. There are a myriad of stakeholders interested in functioning maritime monitoring, such as government authorities, insurance companies as well as environmental protection organizations.

One organization specializing in solutions for advancing maritime monitoring, threat anticipation and triggering and improving overall efficiency is Airbus. One of the attempted advances initiated by Kaggle is the automatic ship detection from satellite images. The objective here is dedicated improvement in the dimensions of speed and accuracy, as these remain challenging in automatic object extraction from satellite imagery. This problem is published as a challenge on the machine learning competition community platform Kaggle Inc.¹ Airbus provided original satellite images and corresponding masks.

The authors of this report took part in this Kaggle competition as part of the machine learning project. The following report summarizes the results for ship detection from satellite images using different deep learning approaches.

The implemented models and programmes as well as their results are stored in the team's GitHub Repository under the URL: <https://github.com/mozzhorin/ML-PR18>. For the final models as well as documentation please refer to the submission folder.

Evaluation

This competition is evaluated on the F2 Score at different intersection over union (IoU) thresholds. The IoU of a proposed set of object pixels and a set of true object pixels is calculated as:

$$IoU(A, B) = \frac{A \cap B}{A \cup B}$$

The metric sweeps over a range of IoU thresholds, at each point calculating an F2 Score. The threshold values range from 0.5 to 0.95 with a step size of 0.05: (0.5, 0.55, 0.6, 0.65,

¹ <https://www.kaggle.com/c/airbus-ship-detection#description>, Accessed: September 03, 2018

0.7, 0.75, 0.8, 0.85, 0.9, 0.95). In other words, at a threshold of 0.5, a predicted object is considered a "hit" if its intersection over union with a ground truth object is greater than 0.5.

At each threshold value t , the F2 Score value is calculated based on the number of true positives (TP), false negatives (FN), and false positives (FP) resulting from comparing the predicted object to all ground truth objects. The following equation is equivalent to F2 Score when β is set to 2:

$$F_{\beta}(t) = \frac{(1 + \beta^2) \cdot TP(t)}{(1 + \beta^2) \cdot TP(t) + \beta^2 \cdot FN(t) + FP(t)}$$

A true positive is counted when a single predicted object matches a ground truth object with an IoU above the threshold. A false positive indicates a predicted object had no associated ground truth object. A false negative indicates a ground truth object had no associated predicted object. The average F2 Score of a single image is then calculated as the mean of the above F2 Score values at each IoU threshold:

$$\frac{1}{|thresholds|} \sum_t F_2(t)$$

Lastly, the score returned by the competition metric is the mean taken over the individual average F2 Scores of each image in the test dataset.

Prizes

- 1st place - \$25,000
- 2nd place - \$15,000
- 3rd place - \$5,000
- Algorithm Speed Prize (Post competition prize) - \$15,000

The goal of the Algorithm Speed Prize is to encourage participants to develop the fastest running algorithm that will still produce accurate results. The speed prize will be open to the top performing teams on the leaderboard.

Timeline

- July 30, 2018 - Start Date.
- September 27, 2018 - Entry deadline.
- September 27, 2018 - Team Merger deadline.
- October 4, 2018 - Final submission deadline.
- October 18, 2018 - Final submission deadline for the Algorithm Speed Prize.
- November 22, 2018 - Announcement of the winner of the Algorithm Speed Prize.

Because of a data leak, the deadlines were extended. Details on the data leak are provided in Section 3 (Data Understanding).

2 Methodology

Since the idea of the challenge was not only to achieve a good accuracy in ship detection and localization but also develop a quicker high-performance algorithm, we have suggested using some extra technic for this purpose. We wanted to compare two models, which are widely used for image segmentation and object localization: Mask R-CNN and U-net. U-Net was developed for the biomedical data but recently has shown good results for satellite images also. Mask R-CNN is a successor of Fast R-CNN and Faster R-CNN and now is officially a part of a Facebook's project Detectron. Both of these models have a deep architecture and many specific heuristics to improve their precision. Both models are characterized by a big number of parameters, what makes them relatively slow in training and predictions. Thus, we have tried to combine them with more simple and fast (pre)classification method. In the first stage this method should scan through the images and reject the empty ones, so on the second stage, our large models will process only the selected images, which contain ships with some high probability. Consequently, we aimed to compare the accuracy and performance of Mask R-CNN and U-net on the whole test data and only on the selected during the pre-classification part. Due to some technical difficulties, we have not combined pre-classification and localization blocks in one model, so the estimations of the performance improvements are based on the ratio of images, selected during the pre-classification, with the whole amount of the test data. That should be considered as a low boundary of the performance improving because does not include possible parallel processing of the data through these two stages.

3 Data Understanding

The original train dataset contained 104700 satellite images with 131030 ships on them. Test set contained 88486 images. 15 images were corrupted (1 from the train and the rest from the test). All images have resolution 768x768 px and were cut from some bigger images. No location or time data were provided in the challenge. Kaggle had reassured that there is no overlapping in the images.

Besides ships and water, the images also contain clouds, haze, wakes behind the ship (that will not count as part of the ship), coastal areas, docks, marinas, reflections, waves and other floating objects such as buoys, barges, wind turbines, etc. There is also some image cutting artifacts.

Most of the images are empty, only ~39% of them contains ships. You can see on the histogram that images with only one ship are dominating.

After a few weeks of running a severe data leak in the challenge was found: test images occurred to be just shifted train images, there were also overlaps inside train set. Then the challenge hosts provided masks also for the test set, so it can be also used for training. The new test set should be provided on the first week of October. Until that we have focused on training and validating the old data².

The new test set was published on 8th October. It contains only 15606 images.

² <https://www.kaggle.com/c/airbus-ship-detection/discussion/64388>, Accessed: September 05, 2018

4 Pre-Classification

We aimed to compare several different methods for pre-classification but, because of some technical difficulties and a shortage of time, we have focused only on simple models of convolutionally neural networks.

4.1 Data Preparation

There was no complete data loader for these challenge based on Pytorch, which includes all the required functions, so we developed our own using examples from the practice week and some Kaggle kernels. During the preprocessing, the images were resized and augmented. The corrupted images and images with the size smaller than 40 KB (~200 images) were dropped. 10% of the train data set was used only for validating during training.

Resize factor

In order to accelerate the pre-classification we have used instead of original 768x768 px images the resized versions of them:

- factor 2 (384x384 px)
- factor 4 (192x192 px)
- factor 8 (96x96 px)

Since some ships take only a few percents of the original images square, with the resize factor 8 they will take only several pixels and will be lost by classification. So after a short check, we rejected this factor for the detailed research. With the resize factor 2 both models have converges to either always 1 or always 0.

Data Balancing

One of the technics, which we have tried, is the balancing the ratio of negative examples using only 30% of empty images from train data and all the images with ships on them. In this case, the amount of positive and negative examples are approximately equal. That heuristic can be considered as not clean in data science because the structure of the train data supposed to be similar to test data, but in combination with a specific threshold on a softmax function, it gave us a suitable result for the resize factor 4. Unfortunately, for the resize factor 2 that did not help with models convergence to extreme states.

4.2 Data Modelling

We tried different architectures and settled on two models with two convolution layers with kernels 5x5 and one linear layer:

- 8 and 16 features (CNN_8)
- 32 and 64 features (CNN_32)

Each convolutional layer was followed by ReLU function and max-pooling with kernel 2x2. We have used cross-entropy as a loss function and stochastic gradient descent for optimization. After few tries, we chose the learning rate 0.001, the momentum 0.9 and the weight decay 0.0005. We have trained the models until the loss stops descending, that was after about 50 epochs.

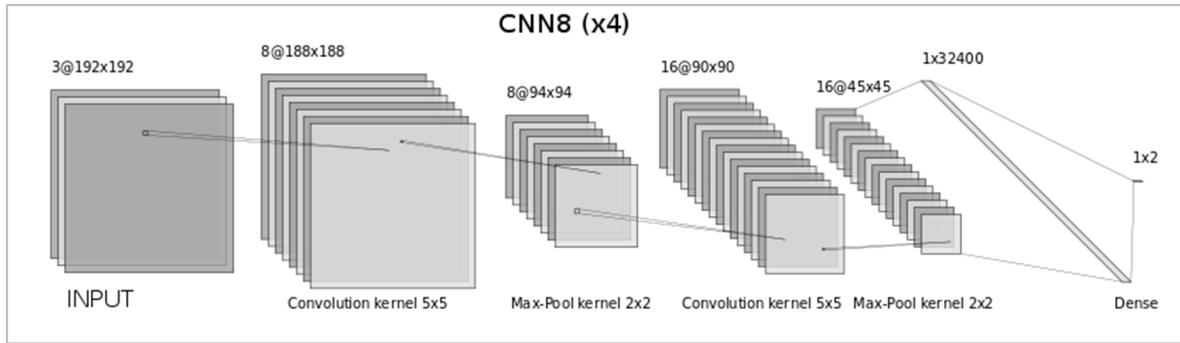


Figure 1: Network architecture of CNN8

We have also compared the usage of these models with the different image resize factors and balance of positive and negative examples for the training. And in the end, we have compared several of these combinations setting different thresholds on softmax functions. For this work, we have used Pytorch.

Training

Here we show some curves of the loss function and accuracy. We have used a simple binary accuracy: 1 if the prediction is right and 0 else. For our task it is not the optimal metric, so we have mostly focused on the confusion matrices. You can see how their values change during the training. Here we have only two types of errors:

- false positive, when the model predicts ships on empty images,
- false negative, when we lose some images with ships in the next stage.

The first brings down the efficiency of the algorithm and the second - its precision. So in any case, our task is to find a trade-off between these two characteristics. We assume that the minimization of the false negative error has here a higher priority. Previously we have predicted a class of an image just taking one of them with a maximal weight. For more precise models tuning we have passed the weights through the softmax function, which returns us values in $[0,1]$, and compared results for different thresholds. The previous predictions will be similar to a threshold 0.5. We have decided that the optimal ratio between two errors gives us CNN_8 model with resize factor 2 trained on balanced data with the threshold 0.4.

Predicting

After choosing the best model and resize factor we have run the preclassification on the old test data (~88k images) for two thresholds: 0.4 and 0.5. As we have previously described a data leak occurred in the challenge, so the hosts have provided masks also for this test set. According to these data we have evaluated our predictions one more time more accurate. Using 0.4 threshold we lose only ~5% of the images with ships on them but rejecting ~35% of images, thus, the performance of the next stage can be improved minimum at 1.53 times comparing to processing the whole dataset.

We did the same also for the new test set (~15k images), but in this case, the quality of classification can be observed only indirectly through the score after the second phase.

4.3 Technical Specifications and Runtime

The model has run on an Amazon Web Service (AWS) p2.xlarge instance with the following specifications using GPU (CUDA) acceleration:

- 4 Core CPU (Intel Xeon E5-2686 v4 Broadwell)
- 61 GB RAM
- 1 GPU (NVIDIA K80 with 2496 cores and 12 GB memory)

One run on the old test data set (88486 images) takes on average 9:25 (min:sec) and on the new one (15606 images) - only 85 seconds, which is much faster than both models need on the second stage.

The code specifications for data preparation, modelling and visualization are provided in the Appendix (see 9.1 Code specifications for pre-classification).

4.4 Data Visualization and Evaluation

4.4.1 Loss

The loss was recorded during training and validation for both models respectively.

- 8 & 16 features (CNN_8)
- 32 & 64 features (CNN_32)

The loss function for CNN_8 for balanced (cnn_8_x4_b) and unbalanced (cnn_8_x4_nb) datasets is depicted in Figure 2.

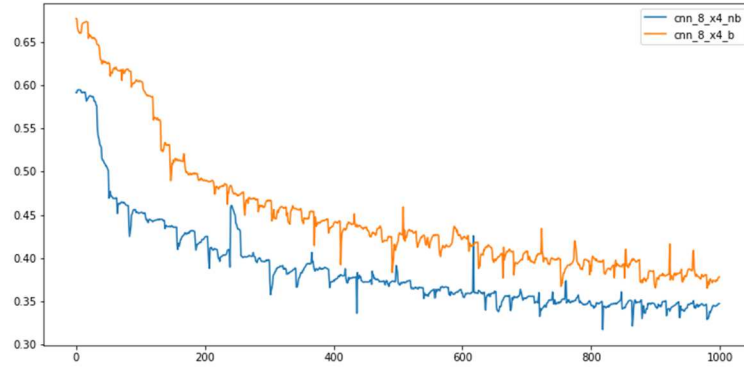


Figure 2: Loss (Cross-Entropy) over steps for balanced and unbalanced set

The loss function for CNN_32 for unbalanced dataset is displayed in Figure 3

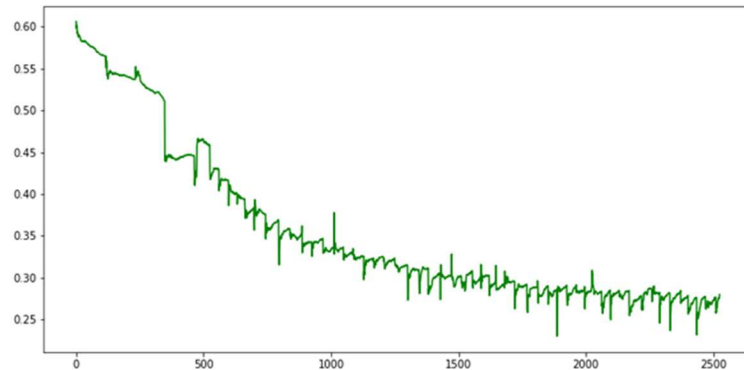


Figure 3: CNN_32 Unbalanced | Loss (Cross-Entropy) over steps

4.4.2 Accuracy

The accuracy for CNN_8 for each epoch is presented in Figure 4.

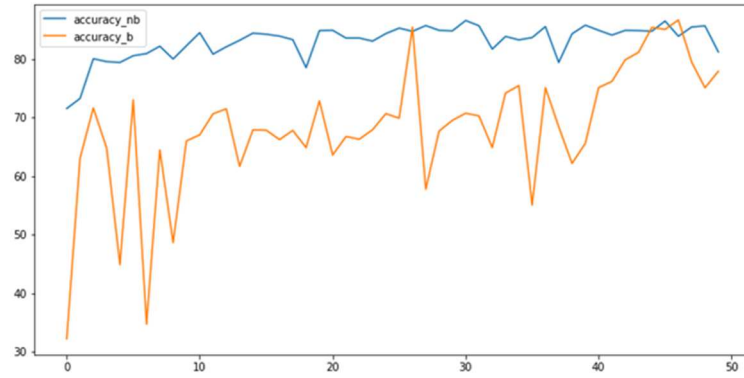


Figure 4: CNN_8 Accuracy per epoch for balanced and unbalanced set

4.4.3 Errors

Errors in predictions are shown in the following. Figure 5 plots these errors (false positives and false negatives). Blue and orange represent the functions for false positives for unbalanced and balanced respectively, and green and red depict false negatives for unbalanced and balanced respectively.

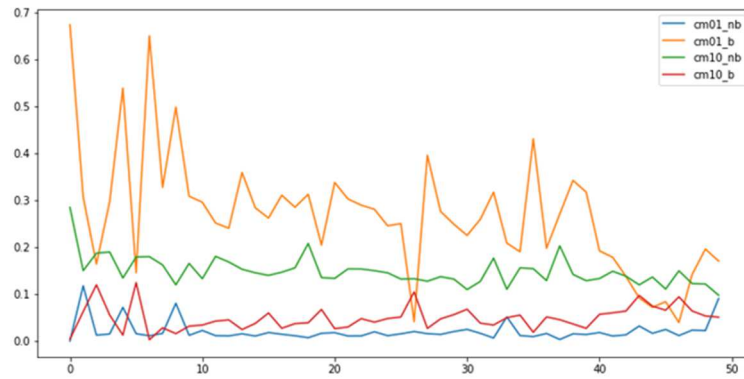


Figure 5: CNN_8 Errors per epoch for balanced and unbalanced set

The Confusion matrix displayed in gives further insight into the depicted errors.

Table 1: Confusion matrix for Preclassification

Predicted		CNN8_nb		CNN8_b		CNN32_nb	
		0	1	0	1	0	1
True	0	0.62	0.09	0.54	0.17	0.69	0.02
	1	0.10	0.19	0.05	0.23	0.11	0.17

Accordingly, the softmax thresholds for the errors are presented in Figure 6. The blue and orange line represent the errors for the unbalanced set, for false positives and false negatives respectively. The green and red line plot the balanced error function for false positives and false negatives.

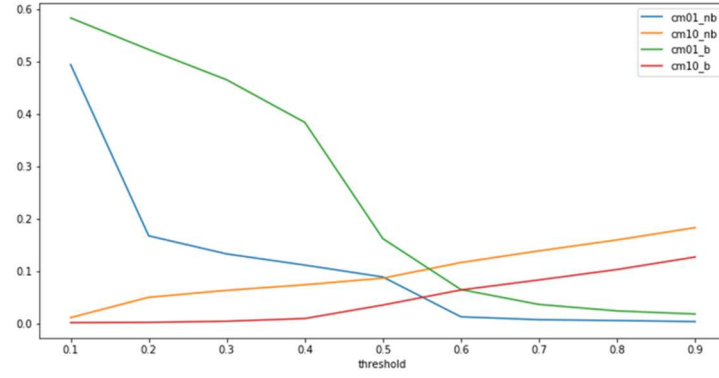


Figure 6: CNN_8 Errors per threshold for balanced and unbalanced set

5 U-Net

5.1 Data Preparation

The definition of hyperparameters is performed prior to the data preparation. The definition is highly dependent on technical characteristics of the used architecture. The used modules are general statistical libraries and modules such as numpy, pandas and os. As the model is performed on the Keras Python Deep Learning API with TensorFlow backend.

The data preparation phase included the following steps: run length encoding and decoding, transformation of masks to images, preparation of the train and validation set, data balancing and data augmentation.

5.2 Data Modelling

The selected model architecture is a U-Net, a convolutional neural network (CNN) with skip connections located prior to the bottleneck layer. It consists of Convolution Operation, Max Pooling, ReLU Activation, Downsampling/Concatenation and Upsampling/Expanding Layers. [1]

The architecture is based on Fully Convolutional Networks (FCN). U-net is symmetric and the skip connections between the downsampling path and the upsampling path apply a concatenation operator instead of a sum (compared to FCN). These concatenations transfer local information to the upsampling block while upsampling transfers global weights. Due to its symmetry, the feature maps tend to be large.

The architecture can be divided into three parts, the downsampling path, the upsampling path and the bottleneck. The former consists of four blocks, each represented as two 3x3 convolution layers and a corresponding activation function and 2x2 Max Pooling. The images and corresponding masks are ingested into the model, starting with 64 feature maps for the first block. The number of feature maps is then doubled at each pooling. The downsampling path is used to encode the context of the input as the basis for segmentation. This context is passed to the upsampling path using skip connections between down- and upsampling blocks.

The upsampling path consists of two 3x3 convolution layers with corresponding activation functions, concatenation with the feature map from downsampling, and a deconvolution

layer of stride 2. This path enables precise localization going up the architecture and combined with the contextual information from concatenation, transferring the information's state before being passed to the bottleneck. Finally, the bottleneck is located in between both sampling paths and is built on two convolution layers. The general idea behind this symmetric architecture is to concatenate higher resolution features from contracting and combine them with the sampled features for better pixel-based localization. The basic architecture of U-Nets is presented in the Appendix (see Section 9.3).

The U-Net model was implemented using the Keras Sequential model [2]. Keras is a deep learning library based on Python programming language and can run on TensorFlow and Theano, among other backends. The implementation described here used TensorFlow backend. The model is implemented by calling the **Sequential()** model, and then simply adding the U-Net layers using the **.add()** method. After modelling is finished, the learning process can be implemented using the **.compile()** function. The training and validation is based on the following parameters:

- Loss function: Intersection over Union
- Optimizer: Adam
- Accuracy Meter: Keras Binary Accuracy
- Additional accuracy metric: Dice Coefficient

The model was trained using a balanced train sample. The predictions were then made in two phases:

- I. Prediction for the entire test data set.
- II. Prediction only for the images, that were identified as images containing ships in the Pre-classification Model (Section 4), additional prediction input: csv file).

Parameters such as the learning rate were automatically adjusted using the method **ReduceLROnPlateau()**. This function automatically adjusts the learning rate after each epoch, if the measured metric (in this case validation loss) has not improved. In this model, the minimum learning rate was set to **1e-8**.

Another implementation was performed using different parameters. Due to technical disruptions disconnecting the network, the weights of the trained model were saved in .hdf5-format. The parameters are the following:

- Maximum epochs: 50
- Optimizer: SGD
- Loss function: Binary Cross-Entropy
- Steps per epoch: 250
- Batch size: 48

The loss and accuracy showed more stability, a better loss decrease and a better accuracy development. Thus, the Evaluation (Section 5.4) presents the loss and accuracies of the latter model, referred to as U-Net Model B hereinafter. Submissions for this file were however not made to Kaggle and therefore don't have a score.

5.3 Technical specifications and runtime

The model was implemented on an a cluster using GPU (CUDA) acceleration with the following specifications:

- 6 Core CPU (Intel ® Core™ i7-5939K CPU, 3.50GHz)
- 128804GB RAM
- 1 GPU (NVIDIA-SMI with 8105MB memory)

Table 2: Overview of runtime for prediction of entire test data and preclassified test data (thresholds 0.4 and 0.5)

ID	Description	Runtime (HH:MM:SS)
T1	Test size: 15,606 images	01:02:49
P4_1	Pre-classified test Size with threshold 0.4: 10,167 images	00:43:45
P5_1	Pre-classified test size with threshold 0.5: 5,514 images	00:21:33

5.4 Data Visualization and Evaluation

The following Visualizations are presented from the second implemented Model U-Net Model B, as the loss and accuracy function was more stable.

5.4.1 Loss

The following The selected loss function was the binary Cross-Entropy loss. The decreasing trend of the loss function in the U-Net Model for Training and Validation is depicted in Figure 7.

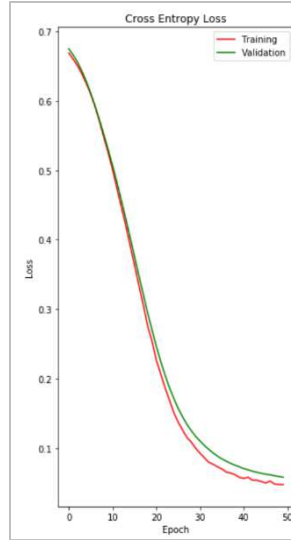


Figure 7: U-Net loss per epoch

5.4.2 Accuracy

The accuracy of the U-Net is displayed in the middle plot in Figure 8.

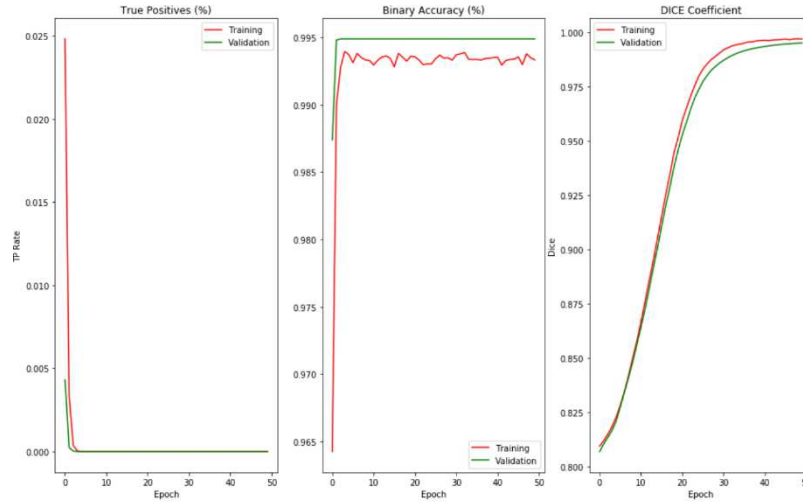


Figure 8: Accuracy of U-Net Model B per epoch (1) True Positives, (2) Binary Accuracy and (3) Dice Coefficient

5.4.3 Kaggle Score

The submission files for is implemented to output three csv-files, one for the prediction of the entire test dataset and two for the prediction of the pre-classified datasets using the thresholds 0.4 and 0.5 .

Table 3: Overview of Kaggle submission files, the underlying U-Net model and achieved Kaggle Score

<i>Underlying Model</i>	<i>ID</i>	<i>Description</i>	<i>Kaggle Score</i> ³
U-Net Model A - Maximum epochs: 50 - Optimizer: Adam - Loss function: IoU - Steps per epoch: 50 - Batch size: 48 - Early stopping after 34 epochs	T1	Prediction on entire test dataset	0.546
	P4_1	Prediction on pre-classified dataset with threshold 0.4	0.580
	P5_1	Prediction on pre-classified dataset with threshold 0.5	0.593

5.4.4 Remarks

The achieved Kaggle Scores are expected to be higher using submission files from the trained Model B. This assumption is based on the positive loss and accuracy trends. However, only submissions from Model A were submitted in Kaggle.

³ Highest achieved Kaggle Score as of October 15, 2018: 0.733

6 Mask R-CNN

In this chapter, the Mask R-CNN or Mask Region-based Convolutional Neural Network is introduced as a framework to be used to overcome the Airbus ship detection challenge. Based on the defined classes or objects, the system detects the objects and delivers their respective masks. The image segmentation property of this framework makes it a powerful tool for the ship detection challenge. In this section, the background of the model is explored first. Afterward, the model itself together with its components will be introduced. How this model handles the data, the model and its data delivery will be inspected subsequently. Finally, the evaluation and improvement suggestion will be provided.

6.1 Related work

The computer vision section has improved rapidly in object detection and semantic segmentation thanks to the baseline systems, such as Fast R-CNN [3], Faster R-CNN [4] and Fully Convolutional Network (FCN) [5]. Compared to the mentioned computer vision tasks, the task of image segmentation has proven to be very challenging. Due to the challenging nature of instance segmentation, the development team of the Mask R-CNN has targeted a framework for this task, which is comparable to those mentioned earlier in performance, robustness and flexibility [6]. Before diving into the framework and its components, the image segmentation and R-CNN or Region-based CNN will be explored for setting the ground of the Mask R-CNN.

6.1.1 R-CNN and paving the way towards the Mask R-CNN

The Region-based Convolutional Neural Network approach generates category-independent region proposals, called Region of Interests or RoI, by using a region proposal module. Each region is then fed individually to a CNN (part of a large CNN) which acts as a feature extraction component. The CNN extracts a fixed-length vector from each region and produces feature vectors as output, which is also fed to set of SVMs⁴ for classification.

In other words, it trains the backbone convolutional networks separately (and end-to-end) on every single RoI to classify each RoI into object(s) of interest or background. R-CNN acts mainly as a classifier and make no prediction about object bounds [7].

⁴ SVM: Support Vector Machine

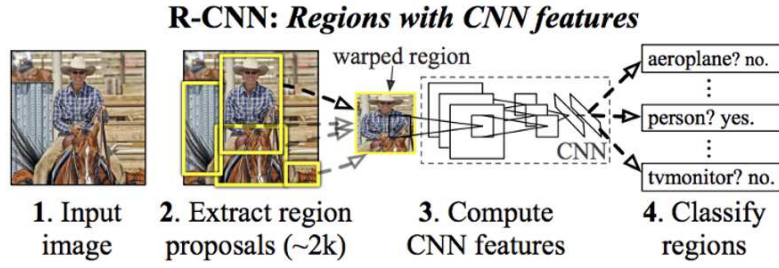


Figure 9: R-CNN Framework [7]

This method was further improved to enable the attending of RoIs on feature maps by utilizing the RoI Pool, which led to a better performance and accuracy, hence the name Fast R-CNN. In Fast R-CNN, the input of the system is an entire image and a set of object proposals or RoIs. This input is processed by a large CNN together with a max pooling layer. This process pools each RoI into a fixed-sized (convolutional) feature map (containing only RoIs). By using a RoI pooling layer and a sequence of fully connected layers, each RoI on the feature map is then mapped onto a fixed-length feature vector. Finally, a softmax layer predict the class of proposed region and the offset values for the bounding box [5].

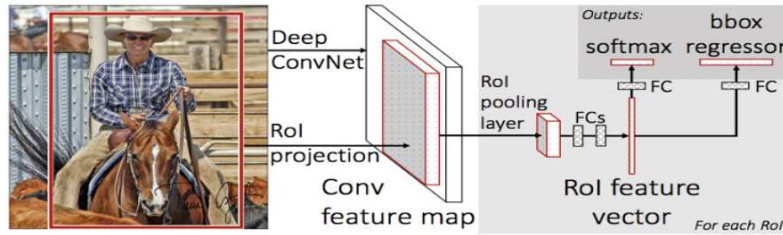


Figure 10: Fast R-CNN Framework [3]

The both mentioned methods uses selective search to generate region proposals, which is a time-consuming task. The Faster R-CNN speeds up the whole process by omitting the selective search algorithm and letting the network learn the region proposal itself. In Faster R-CNN, the system is comprised of a deep fully convolutional network which serves as region proposal module and a Fast R-CNN detector. In this case, the RPN⁵ module (utilizing the attention mechanism [8]) point the regions out to Fast R-CNN module. The RPN module uses sliding-window method and scans over the convolutional feature map, which is generated by the last shared convolutional layer. Multiple region proposals are predicted at the

⁵ RPN: Region-based Proposal Network

location of each sliding-window which are called anchors. Each anchor (region proposal) is provided with a score that estimates the probability of object or not object [4].

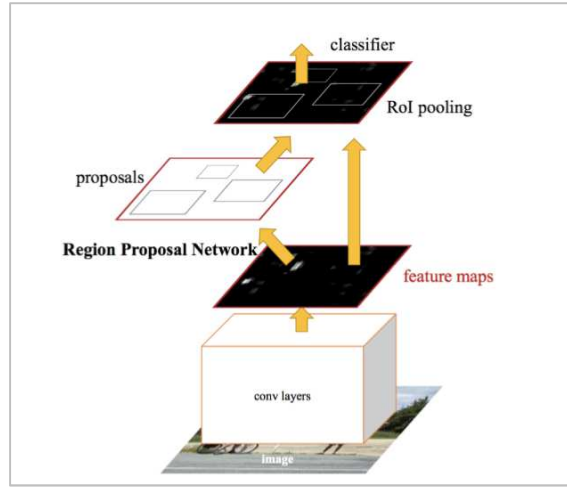


Figure 11: Faster R-CNN Framework [4]

6.1.2 Image Segmentation

Image segmentation is the process of dividing a digital image into multiple regions or segments, which correspond to different objects or parts of objects. Every pixel in the image is assigned a label which correspond to an object or a class. As a result, a set of pixels, which are labeled identically, shares a specific character. This provides a mean to represent a digital image into a much more meaningful and easier to analyze object [9].

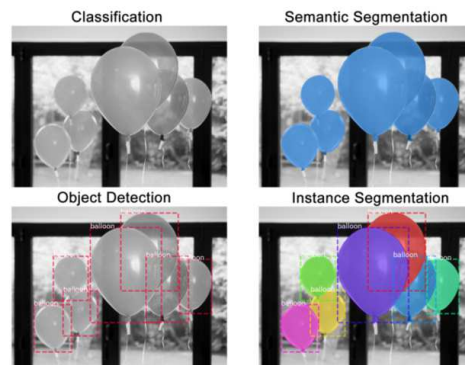


Figure 12: Task of Computer Vision [10]

6.2 Mask R-CNN Details

The Mask Region-based Convolutional Neural Network adopts the two-stages procedure of Fast R-CNN. It keeps the RPN stage untouched but adds a branch in parallel to the class and

boundary box prediction in second stage, which outputs a binary mask for each RoI [6]. The Mask R-CNN framework, which is used for ship detection during this project, was introduced by Kaiming et al., 2017 in Mask R-CNN paper.

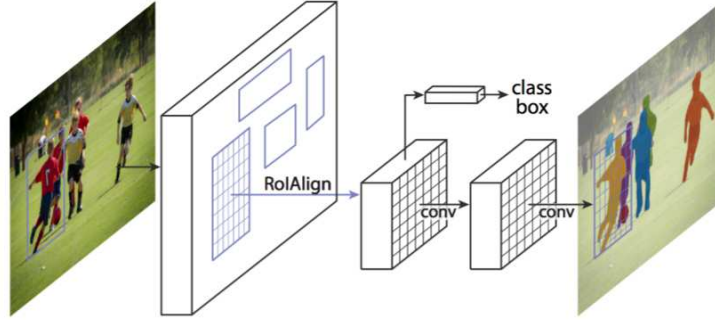


Figure 13: Mask R-CNN Framework [6]

The implementation of the model is provided by Waleed A. via his Github page [11]. The model is generated by using the Model Class API of Keras. This framework consists of the following components:

1. **Backbone** which is composed of a standard ResNet50 or Resnet101⁶ and is configurable in code. This component is extended via Feature Pyramid Network(FPN). The backbone is implemented in *Keras*. The ResNet has 5 stages. The first stage is a keras 2D convolution layer with output filters of 64, kernel size of (7,7), ReLU function as activation and max-pooling with a pool size of (3, 3). The second stage consists of a convolution block⁷ and two identity blocks⁸ each with kernel size 3. The third stage has one convolution block and three identity blocks each also with kernel size 3. The fourth stage makes the difference between ResNet50 and 101. In our case, namely ResNet50, this stage consists of a convolution block and five identity blocks. The last stage is like the second stage but with different output filters for convolutional layers of convolution and identity blocks.

⁶ Residual Network which employs deep residual learning to overcome the saturation of accuracy in much deeper networks when depth of network increase [10]

⁷ A residual block with shortcut which has a convolutional layer at the shortcut since input and output have different dimensions

⁸ A residual block whose shortcut can be directly used because input and output are of the same dimension

2. **Region Proposal Network** or RPN which slides a small neural network over the generated convolutional feature map and produces anchors at each sliding-window. This layer consists of 2D CNNs and uses soft max for activation.
3. **RoI Classifier and Bounding Box Regression** which runs over the RoIs generated by RPN and outputs two properties for each RoI, Class and Bounding Box Refinement. This layer is a fully connected layer(FC) and uses ReLU function for activation.
4. **RoI Pooling** which crops and resizes the corresponding part of the convolutional feature map to a fixed size for feeding into the classifier.
5. **Segmentation Masks** which is a convolutional network and generates the binary mask for each (positive) RoI selected by RoI classifier

6.3 Data

The train data set consists of 104070 Satellite images with the resolution of 768 x 768 pixels⁹. The training images include either ships or no ships which is indicated by masks via Run-Length-Coding . A CSV file was provided by Kaggle which contains the corresponding masks for each image. An Example of this file is shown in Figure 13. This data set was divided into 90% or 93663 images for training and 10% (10407 images) for evaluation during the training phase. Hence the number of steps per epochs and evaluation. The test data set has 15606 images with the same resolution as training data.

⁹ the data set used for training was provided prior to changes due to the data leak [12]

	A ImageId	A EncodedPixels
	192556 unique values	[null] 65% 438011 44567 4... 0% Other (81721) 35%
24	00052ed46.jpg	
25	000532683.jpg	458957 14 459725 14 460493 14 461261 14 462029 14 462797 14 463565 14

Figure 14: Example of Run-Length-Coding mask

6.4 Hardware Configuration

For the Maks R-CNN section of the project, a system with following configuration was employed:

- CPU: 12 Cores (24 Threads), each core running at max clock of 4 Ghz (AMD Threadripper 1920x)
- RAM: 32 GB DDR4 @ 3200 MHz
- GPU: Nvidia GTX 1080ti with 11 GB of GDDR5X video memory (Pascal with 3584 CUDA cores and boost clock of 1950 MHz)
- OS: Ubuntu 18.04
- Development Environment: Inside a Docker container generated via official TensorFlow docker image

6.5 Framework Configuration

As backbone, a ResNet50 was employed in Model. As a side note, it should be mentioned that we have implemented a ResNet20, a residual Network with 20 layers, for gaining a better performance in training. In comparison to ResNet50, we have improved the training performance per epoch on the configuration which was mentioned in previous section, but not significantly. Prior to these changes, the model with all its layer was already trained for 6 epochs with relative good results. Due to the insignificant improvement in performance and advancement in training, we decided to continue with the ResNet50. Arguably, a residual network is more suitable for detection of more complex objects and more number of classes. For this project, a much simpler neural network could be also utilized with even better performance. But this was also a motivation to observe if sacrificing some performance is an acceptable trade-off for gaining more accurate detection. Further configuration of the framework are as follows:

- Backbone: ResNet50
- Batch size: 1 (1 image per GPU because of the resolution of the images)
- Number of epochs: 30 (Early stopping after 6 epochs)
- Number of steps per epoch: 93663 (due to the batch size and 90% of training data set for training)
- Number of validation steps: 10407 (due to the batch size and 10% of training data set for evaluation)
- Number of RoIs per image: 200
- Positive RoI ratio: 0.33
- Learning rate: 0.001
- Learning momentum: 0.9
- Weight decay: 0.0001
- Detection minimum confidence: 0.90
- Mask resized: from 768x768 to 384x384
- Image augmentation: not employed

6.6 Data Preparation

Initially, the images are loaded together with their corresponding masks.



Figure 15: Image with mask

The bounding boxes are then computed from provided masks. To improve the training performance, the mask pixels inside the computed bounding box is stored rather than the mask of entire image. This mask is further resized to smaller size, in our case half of the original resolution. Choosing smaller masks leads to losing smaller objects. This is governed by *mini_mask* in config class.

During the training, the method *data_generator* of the class model generates the following data for each instance of an object, in our case ship, on the image:



Figure 2-8: Image with bounding box

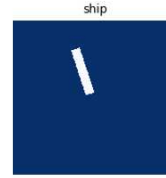


Figure 2-9: The respective Mask

Figure 16: Image with bounding boxes and respective mask

- The RPN generates the anchors using the sliding-window method. Initialized by method `build_rpn_model`, governed by ***RPN_ANCHOR_SCALES*** and ***RPN_ANCHOR_RATIOS*** of config class. The RoIs are generated(Figure 17 (2-11)) and consequently the positive anchors and their refinement (Figure 17 (2-10))
- Based on the positive RoIs, the classifier calculates the bounding box of an object (dotted box) and applies refinement (box with solid lines) to it (Figure 16 (2-8)). The classifier is initiated via the method ***fpn_classifier_graph()***
- The mask branch takes the positive regions selected by the RoI classifier and generates masks respectively(Figure 16 (2-9)). This branch is a convolutional network and is initialized by the method ***build_fpn_mask_graph()***



Figure 2-10: Image with positive anchors



Figure 2-11: Image with RoIs(showing 10 random RoIs out of 200)

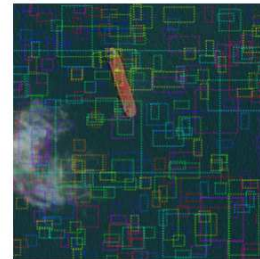


Figure 2-12: Image with negative anchors

Figure 17: Images with anchors and RoIs

6.7 Training

With the described configuration in earlier sections, each epoch took about 9 hours to complete. Due to the time constraint and relative good results, the training was stopped early after 6 epochs.

6.8 Detection

The detection method of the class model predicts/detects the ships and outputs the masks and bounding boxes respectively. Some examples of generated masks are shown in the following images:



Figure 2-13: Grey Image



Figure 2-14: Mask of image

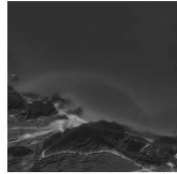


Figure 2-15: Image with small ship



Figure 2-16: Mask of smaller ships

Figure 18: Prediction masks (sample)

6.9 Evaluation

6.9.1 Loss

The losses of various layers of the model were recorded during training and validation and are shown in the following graphs.

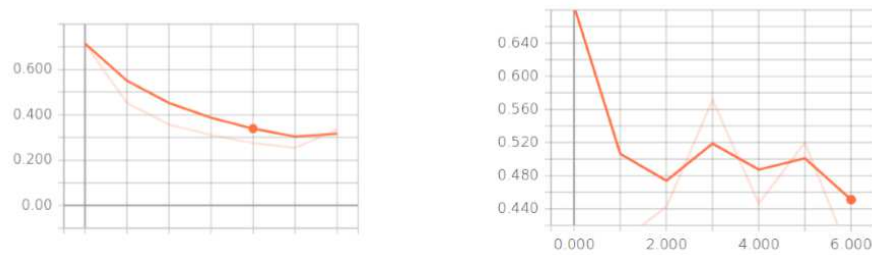


Figure 19: Loss (Cross-Entropy) for Training (left) and evaluation (right)

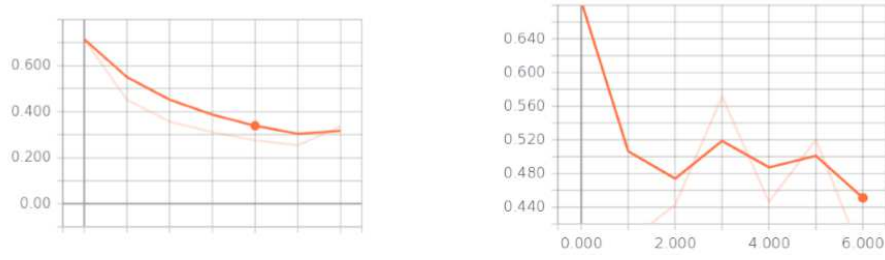


Figure 20: Mask Loss (Cross-Entropy) for Training (left) and evaluation (right)

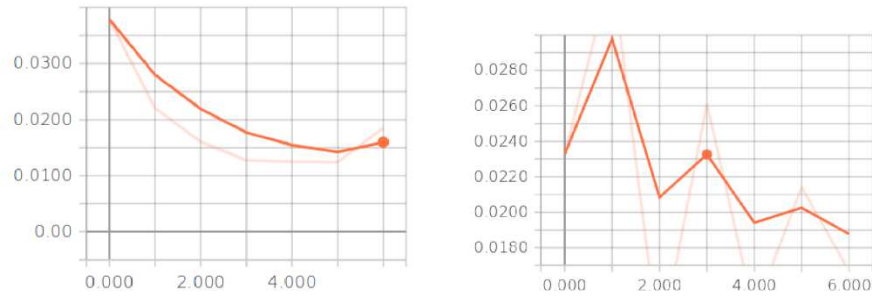


Figure 21: Class Loss (Cross-Entropy) for Training (left) and evaluation (right)

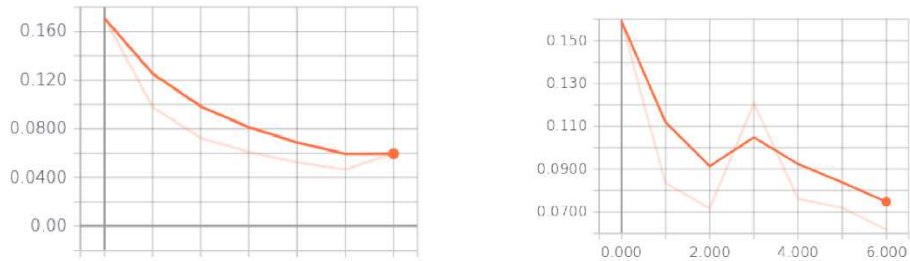


Figure 22: Bounding Box Loss (Cross-Entropy) for Training (left) and evaluation (right)

6.9.2 Kaggle Score

The predicted masks were encoded to Run-Length values and those values were saved in a csv file in the form of 2-6. The predictions were generated for 3 sets of test data and uploaded. The data sets and their respective results are as follow:

- The entire test data, namely 15606 test images. The prediction/detection took about 1822 Seconds. The prediction generated the score 0.671 and positioned us at number 117

- the pre-classified test data with threshold 0.4 consisting of 10164 images. The Mask R-CNN prediction process took 1217 Seconds and scored 0.674 which improved our ranking to 116
- the pre-classified test data with threshold 0.5 consisting of 5414 images. The prediction process took 648 Seconds and generated the same score 0.674

A screenshot of the submission is presented in the Appendix (see Section 9.4).

6.10 Summary

Based on the evaluation and loss values, it is obvious that this model can provide good results if it's trained adequately. After the 6th epoch, the values of evaluation losses have declined to a good extent, which indicates an improvement in learning the data.

7 Evaluation and Summary

The pre-classification of the test data set performed in Chapter 4 led to a performance improvement for the Kaggle score of both segmentation models, i.e. the U-Net and the Mask R-CNN.

<i>Underlying Model</i>	<i>Prediction basis</i>	<i>Kaggle Score</i>
U-Net Model	Prediction on entire test dataset	0.546
	Prediction on pre-classified dataset with threshold 0.4	0.580
	Prediction on pre-classified dataset with threshold 0.5	0.593
Mask R-CNN	Prediction on entire test dataset	0.671
	Prediction on pre-classified dataset with threshold 0.4	0.674
	Prediction on pre-classified dataset with threshold 0.5	0.674

During the model implementation, the team encountered various time consuming challenges, such as the following among others:

- The size of the dataset led to high requirements for the technical resources (especially a GPU), that standard notebooks do not provide.
- The complex data preparation phase as the basis for solid data modelling posed a time-consuming challenge in the process.
- The criticality of data balancing. Unbalanced datasets seemed to have a considerable impact on the prediction accuracy.

8 References

- [1] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*.
- [2] Documentation: Keras, “Keras Documentation: Keras: The Python Deep Learning library | Model class API,” [Online] Available: <https://keras.io/models/model/>. Accessed on: September, 2018.
- [3] R. B. Girshick, “Fast R-CNN,” *CoRR*, vol. abs/1504.08083, 2015.
- [4] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *CoRR*, vol. abs/1506.01497, 2015.
- [5] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” *CoRR*, vol. abs/1411.4038, 2014.
- [6] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, “Mask R-CNN,” *CoRR*, vol. abs/1703.06870, 2017.
- [7] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *CoRR*, vol. abs/1311.2524, 2013.
- [8] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-Based Models for Speech Recognition,” *CoRR*, vol. abs/1506.07503, 2015.
- [9] K. V. Kale, S. C. Mehrotra, and R. R. Manza, *Computer Vision and Information Technology: Advances and Applications*. I.K. International Publishing House Pvt. Limited, 2010.
- [10] W. Abdulla, *Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow*. Available: <https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46>.
- [11] W. Abdulla, *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*: Github, *GitHub repository*.
- [12] K. Inc, *Data Leak and Next Steps*. Available: <https://www.kaggle.com/c/airbus-ship-detection/discussion/64388>.

9 Appendix

9.1 Code specifications for pre-classification

Python programmms:

- `airbus_dataloader.py` - data loading and preprocessing
- `airbus_models.py` - used models (CNN_8 & CNN_32)
- `airbus_train_val_functions.py` - set of functions for training/validating and accuracy calculating
- `airbus_train.py` - train programm
- `airbus_compare_thresholds.py` - compare different thresholds on softmax function
- `airbus_preclassification.py` - generating the predictions

Jupyter notebooks:

- `data_exploration.ipynb` - dataset visualisation
- `preclassifier_predictions.ipynb` - generating the predictions
- `preclassifier_predictions_analysis.ipynb` - visualization of the results of predicting
- `cascade_classifier_datapreparation.ipynb` - datapreparation for a cascade classifier

Other files:

- `ccc_*.py` - versions for the CCC's cluster (different folders)
- `ccc_negative_exaples.txt`, `ccc_positive_exaples.txt` - dataset for the training a cascade classifier
- `*.model` - saved weights for models
- `filtered_ships*.txt` - results of the preclassification for the next stage

9.2 Code specifications for U-Net

The codes and results are stored in the U_Net.zip file in the submission folder in the Project repository.

The zip contains

- Folder: U_Net A (codes and results for model implementation for Model A)
- Folder: U_Net B (Code and weights model for U_Net Model B)
- 20181016_Unetnotebook.ipynb: Jupyter Notebook for model implementation and submission

The main folder that was the basis for the Kaggle submissions is folder U_Net A.

It contains

1. Python programmes:
 - training.py – Python programm for training and validation of U-Net Model A
 - predicting.py -Python program for predicting test images and creating 3 different submission files for trained Model A
2. Folder “Kaggle_submissions”
 - submission_T1.csv - – submission file for prediction on entire test data
 - submission_P4_1 - – submission file for prediction on preclassified test data with threshold 0.4
 - submission_P5_1.csv – submission file for prediction on preclassified test data with threshold 0.5
3. Folder “Results” – Visualizations and model outputs from python programmes
4. Folder “Screenshots” – Screenshots from Kaggle submission and terminal
5. Folder “Archive” – old submission files from previous runs

9.3 U-Net Architecture

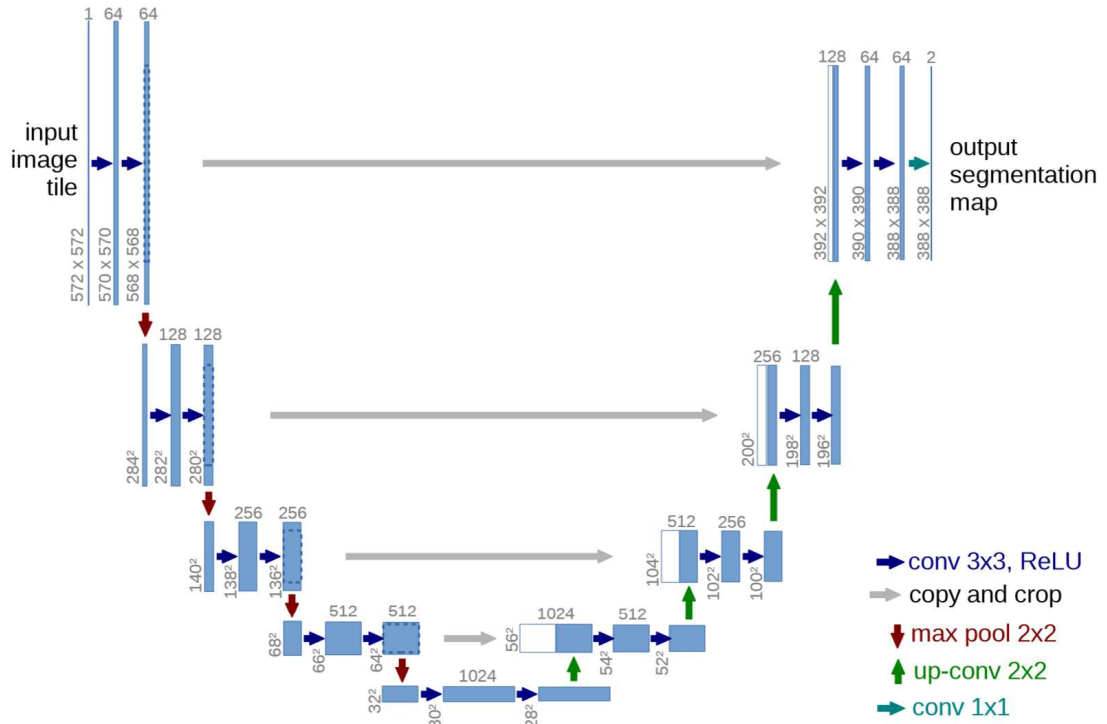


Fig. 1. U-net architecture (example for 32×32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Source: [1]

9.4 Mask R-CNN: Kaggle submission

16 submissions for jenkins_tu_tu		Sort by	Most recent
All	Successful	Selected	
Submission and Description		Public Score	Use for Final Score
submission_big_dataset_firstrun.csv 4 minutes ago by jenkins_tu_tu first run		0.671	<input type="checkbox"/>
data_submission_merged_t4.csv 5 minutes ago by jenkins_tu_tu preclassified_threshold_04		0.674	<input type="checkbox"/>
data_submission_merged_t5.csv 7 minutes ago by jenkins_tu_tu preclassified_threshold_05		0.674	<input type="checkbox"/>

Figure 2-25: Submission Scores




Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
115	new	Phat Hoang		0.677	8	8h		
116	new	jenkins_tu_tu		0.674	5	5m		
Your Best Entry 							Your submission scored 0.671, which is not an improvement of your best score. Keep trying!	

Figure 2-26: Kaggle Ranking