Ship Detection from Satellite images using Computer Vision techniques

George Mandilaras - DS1190012

September 2020

Abstract

In our time, an abundance of satellite images have become available, from projects such us Copernicus[1] and companies like AIRBUS. Combining satellite images with computer vision techniques, has enabled us to develop a plethora of applications extending from precise agriculture to urban planning. In this project I focus on ship detection from satellite images using artificial neural networks. This is an important task which can entail into significant applications that will improve marine trade, by better monitoring and protecting the ships. To encounter this task, I use state-of-the-art models for semantic segmentation, which are FCN8s, U-Net, PSPNet and Mask-RCNN, alongside with computer vision techniques. In the end I achieve to detect the ships and extract relatively good segmentation masks. You can find the code of the project in its github repository[2]

1 Introduction

In our current days, the sky is filled with drones, and the space with satellites. Those drones and satellites are equipped with high resolution digital cameras, capable of capturing images and videos of high definition. These images and videos are very useful for a variety of application such us traffic monitoring, wild fires early detection, urban city planning, high precision agriculture and many security oriented services. However, their large abundance makes it almost impossible for humans to manually inspect them and take decisions based on them. Therefore, nowadays the industry, working together with the scientific community, has employed artificial intelligence and computer vision techniques in order to automate the process of extracting knowledge from such images.

In this work, I focus on ships detection captured from satellite images. This is a closed Kaggle Challenge[3] from AIRBUS[4]. Developing algorithms able of detecting ships can results to numerous important applications that can improve trade, security and defence. Such applications are ship monitoring, piracy confrontation, accidents prevention, surveillance, spying, etc.

This task can be described as an object detection task, where the target is to detect minimum bounding boxes that contain the detected ships. It can also be described as a semantic segmentation task.

[1] https://www.copernicus.eu/el
[2] https://github.com/GiorgosMandi/AIRBUS-ShipDetection
[3] https://www.kaggle.com/c/airbus-ship-detection/overview
[4] https://www.airbus.com/

Semantic segmentation describes the process of associating each pixel of an image with a class label, thus it is a pixel-wise classification and it also known as dense prediction. So, for an input image, the goal is to create a mask by colouring each pixel with with a colour that corresponds to the class, that the respective pixel of the image is part of. The Figure 1 is a a great example that explains the task of semantic segmentation.

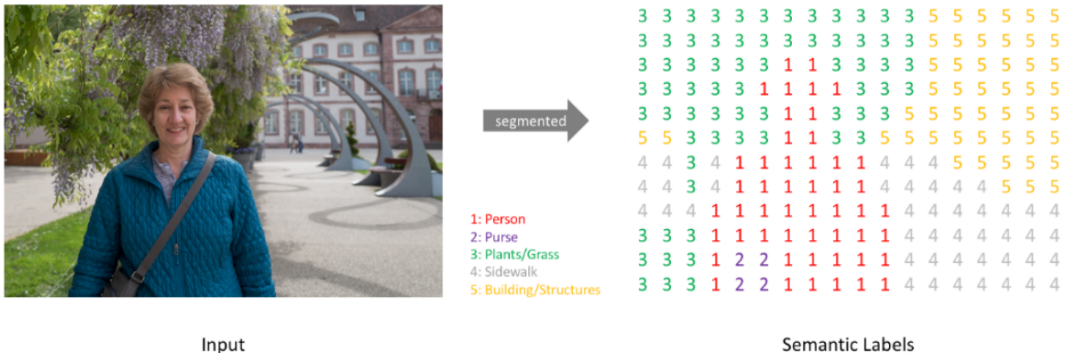


Figure 1: Semantic Segmentation Task

An example of Semantic Segmentation: All the pixels associated with lady's purse, are painted with the colour purple, which corresponds to the class *purse*

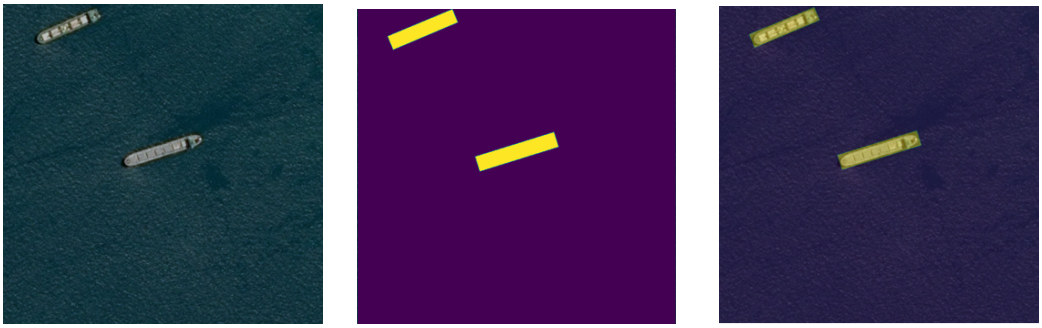
2 Related Work

A lot of work has been done in the filed of urban planning, where they employ semantic segmentation models to extract building footprints from satellite images. Such examples are [13] and [5], where in both cases they have used some variations of the U-Net model, which is a fully convolutional network. Furthermore a lot of work has been done in the field of agriculture where they apply deep learning techniques in satellite images for many purposes. For example many companies such as the company Planet[5] specializes on monitoring crops growth, while also detect anomalies and try to prevent plants' destruction. Additionally, deep learning is also used in crop type classification from satellite images [8].

A very interesting work is [15], where they trained deep learning models to predict survey-based estimates of asset wealth across 20,000 African villages from publicly-available multi-spectral satellite imagery, in order to get the economic well-being of countries in Africa. These local-level measurements of human well-being are important for informing public service delivery and policy choices by governments. This approach is revolutionary as the traditional data collection methods were expensive and rare, while most African households appears on average at least weekly in cloud-free imagery from multiple satellite-based sensors.

An EU project that focuses on deep learning on satellite images is the Extreme Earth[6]. Extreme Earth is a Horizon 2020 project that focuses on the development of Artificial Intelligence and Big Data technologies able to scale up to the extreme scale of Copernicus data. Extreme Earth applies

[5]https://www.planet.com/markets/monitoring-for-precision-agriculture/
[6]http://earthanalytics.eu/index.html



(a) Satellite Image **(b)** Image's Mask **(c)** Mask over Image

Figure 2: Image with its Mask

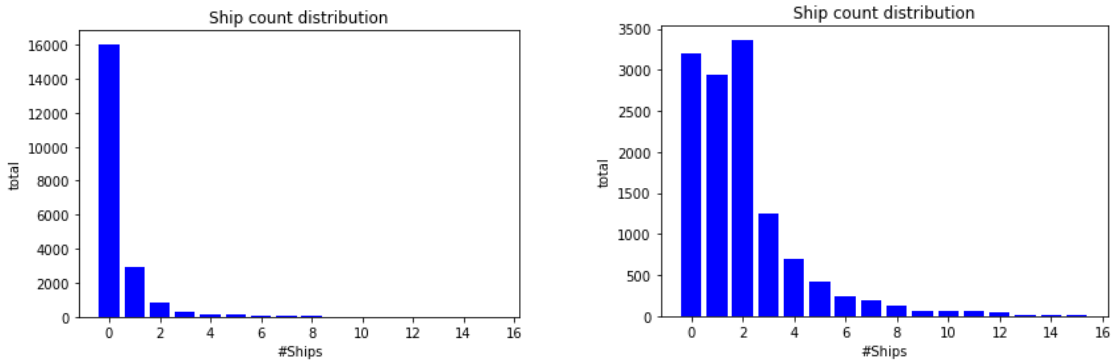
these technologies in two of the thematic exploitation platforms of the European Space Agency: one dedicated to Food Security and one dedicated to the Polar regions. Its goal is to develop techniques and software that will enable the extraction of information and knowledge from big Copernicus data using deep learning techniques and extreme geospatial analytics. Regarding the polar use case, the goal is to interlink the in-situ ice observations with satellite images in order to build training sets with satellite images associated with high quality ground observations, and deep learning models able to determine for the state of ice in the polar cycle. Ice monitoring is a very crucial task for the safely navigation through the Arctic, especially nowadays where the ship traffic in the Arctic has increased.

3 Problem Definition

The creators of the ships detection problem, defined it as a semantic segmentation task in which the goal is to predict a mask with two classes, the class *Ship* and the *Background* class that consists of anything else but ships. The data they provide are numerous satellite images of shape (768, 768), and a CSV document containing the masks of all the ships existing in the images. Each row of the document corresponds to a single ship mask, hence images that contain multiple ships are represented by multiple rows in the documents.

The masks are stored in the form of the Run Length Encoding[7] (RLE) which is good a way to compress data where sequences of the same data value occurs many consecutive times. In our case, most of the images contains a few ships and as a results the masks are very sparse arrays, containing mostly the *Background* class value. Therefore, using the RLE, the mask array is represented by a very small string, and by using the appropriate decoder you can easily retrieve the mask array. Similarly using an RLE encoder from the mask array you can generate the RLE. An example of an image and its mask is displayed in Figure 2.

[7] https://en.wikipedia.org/wiki/Run-length$_e$*ncoding*



(a) Before re-balancing

(b) After re-balancing

Figure 3: Ships Distribution in images

The distribution of ships in the images, before and after data re-balancing. Both plots were produced with the same sample of images

4 Development and Pre-process

4.1 Data Re-balancing

The very first thing I noticed when I downloaded the data, is that there was a great imbalance between the images that contain the ships and those that don't. The large majority of the images consisted of images that does not contain ships, and were just plain images of the sea. Figure 3a depicts the number of ships in a random sample of the images, and as you can see the data are very skew and need to be re-balanced. So I over-sampled the under-representing images that contain ships by replicating them, and I under-sampled the ships-free images by keeping just a small fraction of them. This way I adjust the data in a more preferable distribution, which is displayed in Figure 3b.

However, this procedure was developed before executing larger experiments, consisting of more images. The images without ships were significantly more than those with ships, and due to the lack of available resources and the deescalation of the experiments execution time, I decided to remove them from the training procedure. I believe that this does not impact negatively the overall performance of the models as the parts of the sea are well represented by the images that do contain ships.

4.2 Data Loader

This re-balancing strategy is implemented inside a data loader class, which its primary purpose is to return an image with all of its masks as one. Its first task is to read the CSV document and aggregate all the masks of the images, so each record will contain all the masks of the image it points to. Another function of this class is to split the dataset into training set and validation set, based on a given ratio. During this split the re-balancing of the images is taking place.

It is very important to mention that the data loader never loads all the images in the memory. The training and the validation set it returns, consist of records that point to the corresponding image, and the set that is provided to the models is a python generator which loads batches of images only

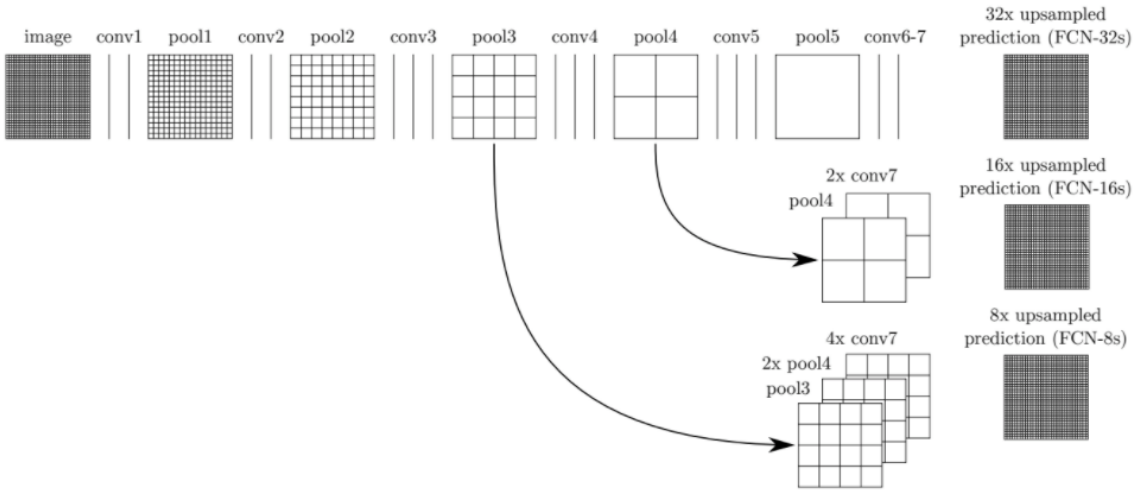


Figure 4: Fully Convolutional Networks

This image describes the architectures of the FCN32s, FCN16s and FCN8s.

when they are requested and right after they are no longer needed, they are removed from the memory. This way I accomplish to minimize the memory footprint which is essential when you have to work with images. During this images loading procedure, their masks are decoded and merged as single one, the values of the image are normalized in the range of [0, 1], and also both the image and mask are down-scaled to the shape (256, 256). This down-scaling probably has an negative impact on the performance of the models, but as I have already mentioned, I had to adapt to the available resources. Furthermore, there is also the option of applying filters to the images in order to enhance the contrast of the image and make the edges more distinct.

This produced generator that loads images on the fly, is then forwarded not directly to the model but to another generator responsible of generating more images through the process of data augmentation. This generates multiple images from a single image by randomly applying shifting horizontally and vertically, rotating, flipping, and zooming to the initial image.

5 Models

In order to predict the segmentation mask, I employed four well known models specialized for semantic segmentation. These are the Fully Convolutional Networks (FCN), the U-Net, the Pyramid Scene Parsing Network (SPSNet) and the Mask-RCNN. In this section, I'm going to explain their architecture and how I implemented them.

5.1 Fully Convolutional Networks (FCN)

Before the appearance of the first FCNs, convolutional networks were used for semantic segmentation tasks such those in [3] and in [10]. Common elements of these approaches includes patch-wise training,

post-processing by super-pixel projection which reduces the computational complexity by segmenting the image into homogeneous regions, and ensemble models such as in [4]. However, such approaches result to complex models, which sometimes were not able to be trained in an end-to-end manner.

Then the first FCNs were introduced in [9], and since then they have become a state-of-the-art in semantic segmentation. These networks are based on the downsampling and the upsampling processes. During downsampling, which is known as convolution, lower-resolution feature mappings are developed which are efficient at discriminating between classes. Then follows the upsampling, also known as deconvolution, in which these lower resolution feature representations are upsampled into a full-resolution segmentation map. Hence these FCNs consists exclusively from convolutional layers which gives them a significant advantage in performance, as they contain fewer parameters than those required in dense layers. Furthermore this back-to-then new kind of architecture enables to transfer success in classification to dense prediction by reinterpreting classifications nets as FCNs, by replacing dense layers with convolutional.

Moreover these FCNs presented by the authors are not plain sequential models. They managed to improve the performance by combining high layers information of the downsampling process with low layers information of the upsampling process. This way they achieved significantly better mIoU (mean of Intersection over Union). Another factor that plays a significant role in the performance is the stride parameter of the upsampling layers, where the smaller it is, the more complex the model becomes but results to higher precision output masks.

In my implementation, I have developed an FCN8s with a VGG16 [14] as its convolutional model backbone, using tensorflow and keras. The "8s" in the end of its name refers to the stride size of the last upsampling layer. Since there was limited resources, I reduced the size of the filters in order to capable of running in my PC. Its architecture is described in Figure 4.

5.2 U-Net

U-Net [12] is an end-to-end fully convolutional network (FCN), designed for bio-medical images segmentation. Its architecture consist of two paths. First path is the contraction path (also called as the encoder) which is used to capture the context of the image. The encoder is just a traditional stack of convolutional and max pooling layers. The second path is the symmetric expanding path (also called as the decoder) which is used to enable precise localization using transposed convolutions. Unlike the previous mentioned FCNs, in the upsampling part of U-Net there is a large number of feature channels, which allow the network to propagate context information to higher resolution layers. This results to higher precision segmentation masks, without the need of complex and heavy backbone models. Figure 5 depicts its architecture.

My implementation of U-Net is based on models I found online, adjusted to fit into my code.

5.3 Pyramid Scene Parsing Network (PSPNet)

Pyramid Scene Parsing Network (PSPNet) is an FCN model developed for scene parsing. Scene parsing is an extension of semantic segmentation where the label, location, as well as shape for each element need to be predicted. PSPNet achieves state-of-the-art performance and it is the champion of ImageNet scene parsing challenge 2016, and arrived the 1st place on PASCAL VOC 2012 semantic

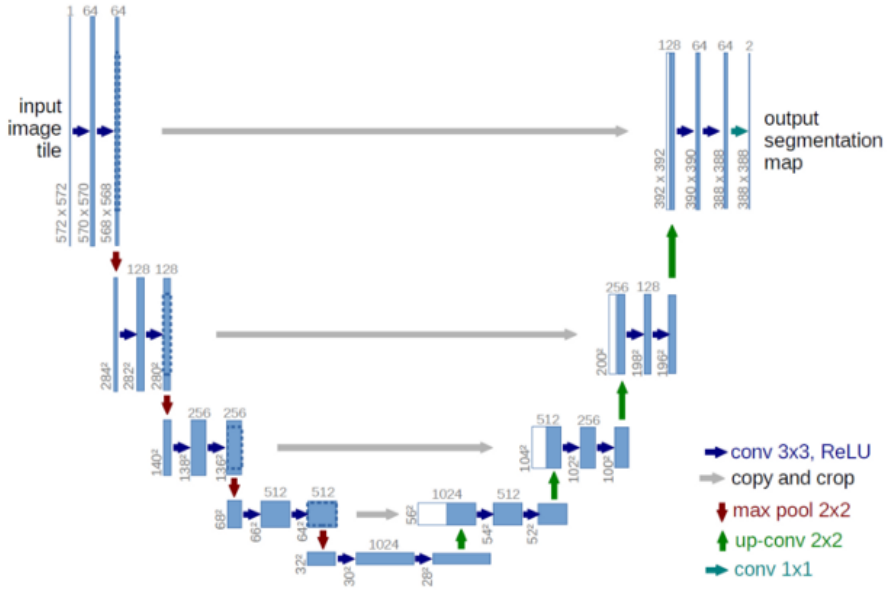


Figure 5: U-Net Architecture

The first path where the downsampling is occurring is the contraction path, and the path where the upsampling is occurring is the extraction path.

segmentation benchmark, and the 1st place on urban scene Cityscapes data. Consequently, it is a very powerful model.

Figure 6 depicts the architecture of PSPNet. PSPNet consists of three main modules. The first one is a convolutional network that extracts the feature maps from the input image. This can be any conv-net, but in the original implementation they use an extended version of ResNet [7]. Then follows the Pyramind Pooling module which fuses features under four different pyramid scales. In Figure 6, the coarsest level highlighted in red is global pooling to generate a single bin output. The following pyramid level separates the feature map into different sub-regions and forms pooled representation for different locations. The output of different levels in the pyramid pooling module contains the feature map with varied sizes. Then upsamples the low-dimension feature maps to get the same size feature as the original feature map via bilinear interpolation. Finally, different levels of features are concatenated as the final pyramid pooling global feature. Finally, the representation is fed into a convolution layer to get the final per-pixel prediction. The keypoint of the success of PSPNet is that the pyramid pooling module can collect levels of information, more representative than global pooling.

In my implementation, I used the code found in this Medium Article[8], which I slightly modified in order to fit in my code framework.

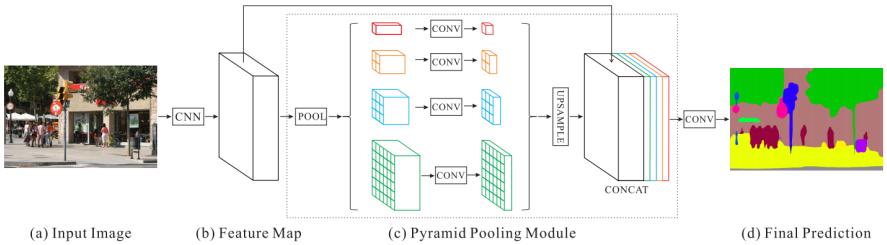


Figure 6: PSPNet Architecture

First a cov-net extracts feature maps from input image, then the pyramind pooling module separates the feature maps into different sub-regions and forms pool representations. This representations are upsampled in order to match the original size and then concatenated into one, which is forwarded into a convolution layer that gets the pixel-wise predictions.

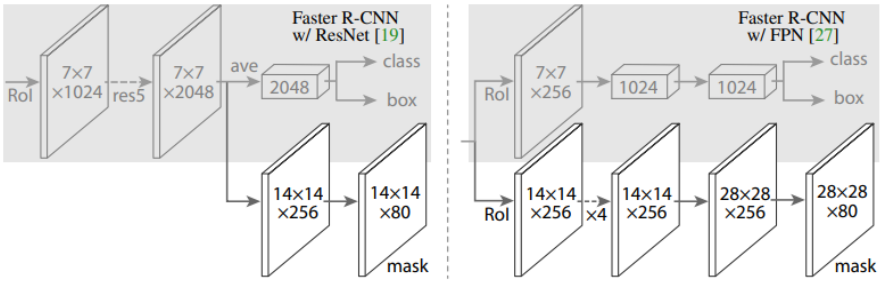


Figure 7: Mask-RCNN Architecture

The image depicts two version of the Mask-RCNN network based on two different implementations of Faster-RCNN.

5.4 Mask-RCNN

Mask-RCNN [6] is an extension of Faster-RCNN [11] combined with an FCN and it is designed for instance segmentation, which target is the correct detection of all objects in an image while also precisely segmenting each instance. Mask-RCNN outperforms the winner of the COCO 2016 segmentation task, including the heavily engineered entries.

Mask R-CNN, extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression. The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. This additional branch add only a small computational overhead, and therefore is slightly slower than the Faster-RCNN. Its architecture is displayed in Figure 7.

Regarding my implementation, I followed the installation instructions of the original repository of Mask-RCNN[9], and then downloaded a frozen model, trained in the COCO dataset. Furthermore, I

[8]https://medium.com/analytics-vidhya/semantic-segmentation-in-pspnet-with-implementation-in-keras-4843d05fc025

[9]https://github.com/matterport/Mask_RCNN

followed their instructions, in order to be able to load and augment my images using this model.

6 Evaluation

6.1 Preliminary

In order to examine all the models under the same circumstances, I implemented a parent class in which I defined how the models will be trained and validated, and the only abstract fields are the segmentation model and other properties related to it. Almost all the models (except Mask-RCNN), are defined in classes that extend this parent class. This was applicable by choosing to implement all models under the same backend, which is tensorflow and keras. In this parent class, I define the optimizer, the loss function and the metrics, some callbacks to improve performance, the number of epochs, the batch size and the way I evaluate their performance during testing.

Regarding the optimizer, I chose to use ADAM optimizer as it is a very efficient optimizer that looks like RMSProp with momentum. The learning rate starts from $1*e^{-3}$ and I periodically decrease it, also the weight decay is set to $1 * e^{-6}$. Using callbacks, I have set a scheduler that halves the learning rate every five epochs, and also reduce LR by a small factor every time it finds itself in a plateau. Figure 9d shows the scheduled values of LR during training. Another callback that I use is the checkpoint callback, which stores the weight of my models every time it detects a local minimum (i.e. the metric is maximized or the loss minimized) and an early stopping. These callbacks are provided to the model during the training process.

For metrics I used the DICE coefficient and the intersection over union (IoU), which are the main widely used metrics for semantic segmentation. Given two sets A and B, DICE coefficient shows the similarity between the two sets and it is defined as DICE $= 2 * |A \cap B|/(|A| + |B|)$. Similarly IoU is another similarity function that shows how much too objects overlap. It is defined as $IoU = (A \cap B)/(A \cup B)$, and it is also widely used in object detection. As for loss function, I use the binary cross entropy subtracted by the DICE coefficient. As I read in forums[10], DICE or IoU as loss functions does not produce good gradients leading to slow and unstable training, so in a lot similar projects a combination of DICE and cross entropy was suggested.

These configurations apply to all the models except the Mask-RCNN. My code regarding Mask-RCNN concerns the loading of the model, configuring its hyper parameter, and extending its data loader class. Furthermore, I managed to pass the same callbacks to improve its performance. Regarding loss functions, Mask-RCNN uses a loss function which is an addition of smaller loss functions. Those are the followings:

- rpn_class_loss : How well the Region Proposal Network separates background with objects.
- rpn_bbox_loss : How well the RPN localize objects.
- mrcnn_bbox_loss : How well the Mask RCNN localize objects.
- mrcnn_class_loss : How well the Mask RCNN recognize each class of object.
- mrcnn_mask_loss : How well the Mask RCNN segment objects.

[10] https://stats.stackexchange.com/questions/321460/dice-coefficient-loss-function-vs-cross-entropy

Model	DICE	IoU
U-Net	0.48	0.24
U-Net + Histogram Equalization	0.42	0.21
U-Net + Adaptive Equalizer	0.44	0.22
PSPNet	0.45	0.21
PSPNet + Histogram Equalization	0.38	0.19
PSPNet + Adaptive Equalizer	0.45	0.23
Mask-RCNN	0.45	0.31

Table 1: The performance of the Models, regarding DICE coefficient and IoU, over the testing set

6.2 Performance

Before training, I split the training dataset into training and validation sets. The initial training set consists around to 20K images, and around 2K of them form the validation set. The training function receives as input the generator that produces the augmented data, and trains for 20 epochs. Then to evaluate the models, I predict the masks of a testing set consisting of 50 images, and for each model I calculate the mean IoU and DICE. The results are displayed in Table 1. The experiments were performed into my laptop containing an intel i7 9th generation CPU, 8GB memory and a GTX1050 GPU. I also tried to perform the experiments in the Collab TPU cluster[11], but I had to have the images stored in my Google Drive and the network I/O for reading the images was significantly deteriorating the execution, erasing any advantage of the TPUs.

As you can see, none of the models manage to achieve high DICE or IoU over the testing set, however all of the models seems to be able to detect the ships but the predicted masks lack precision. In most cases the ships are very small and hence their masks are very small, while the predicted masks are blobby and lack boundary details. As you can see in Figure 9, the loss function have not converged after 20 epochs, which means that we probably can achieve better results by further training. On the other hand, Mask-RCNN is a big pre-trained model, and probably requires more training. By observing its loss during training in Figure 9c, we can reckon that the model has not converged after 20 epochs and requires further training, with even smaller learning rate. Furthermore, I believe that using gradient descent with momentum would also improve its performance, but Mask-RCNN has its own optimization technique and it is not open to change.

In Figures 10 and 11, you can see some examples of predicted masks.

Regarding FCN8s, the reason its results are not displayed, is because I was not able of training it. No matter the epochs and the values of the hyper parameters, the predicted masks were coarse and scattered across the image. You can see some results in Figure 8. The main reason is due to of its vast size, as it uses a VGG16 as its backbone convolution model, hence during the backward bass the gradient was not able to reach the first layers, and therefore they were never updated. This is known as *vanishing gradient*. Even though PSPNet also uses a big convolution network as its backbone (ResNet), it contains residual nodes that allow the flow of the gradient by skipping nodes.

The authors were aware of this issue as it is the same problem that occurs when you train a VGG16. In order to be able to train an FCN8s, I should have first trained an FCN32s and transfer some of its

[11] https://colab.research.google.com/notebooks/intro.ipynbrecent=true

weights to an FCN16s. Then I should have trained an FCN16s, by maintaining the transferred weights frozen (i.e. will not be trainable). Finally, by using some weights of FCN16s frozen inside FCN8s, would probably allow to train and to predict decent segmentation masks.

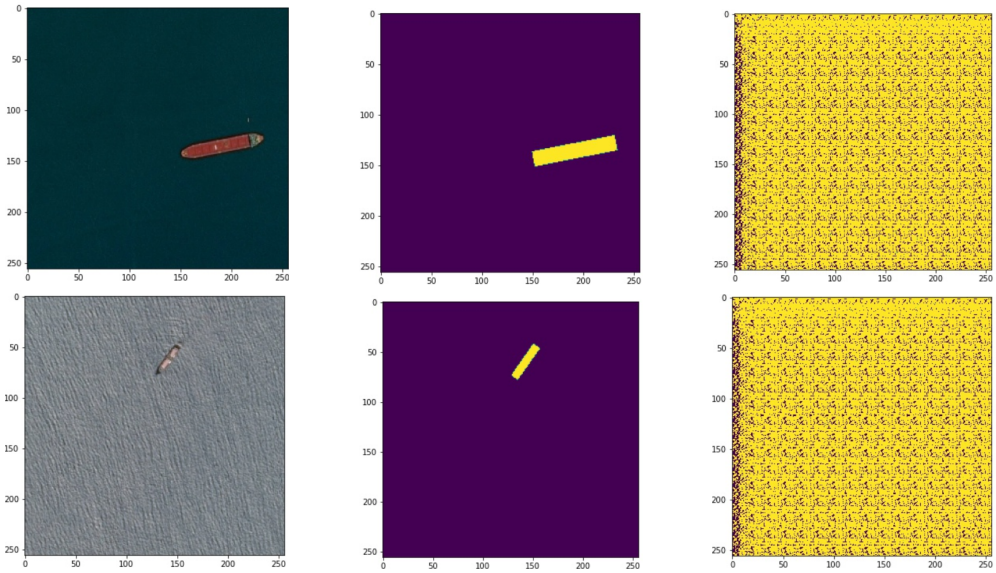
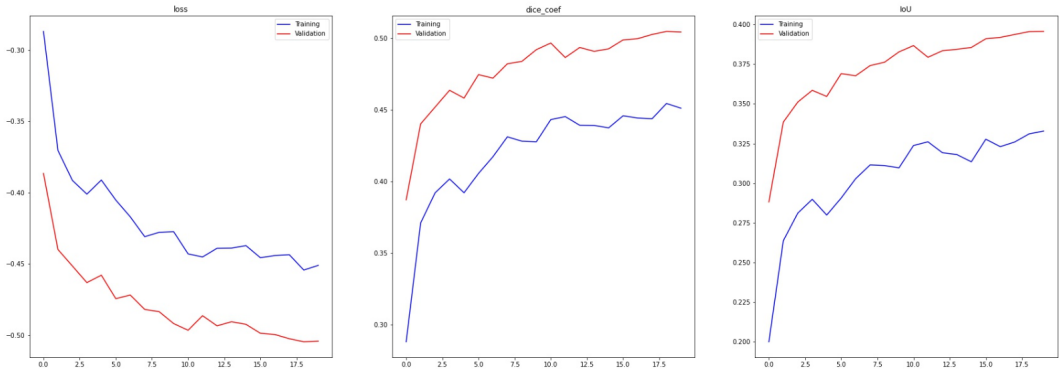
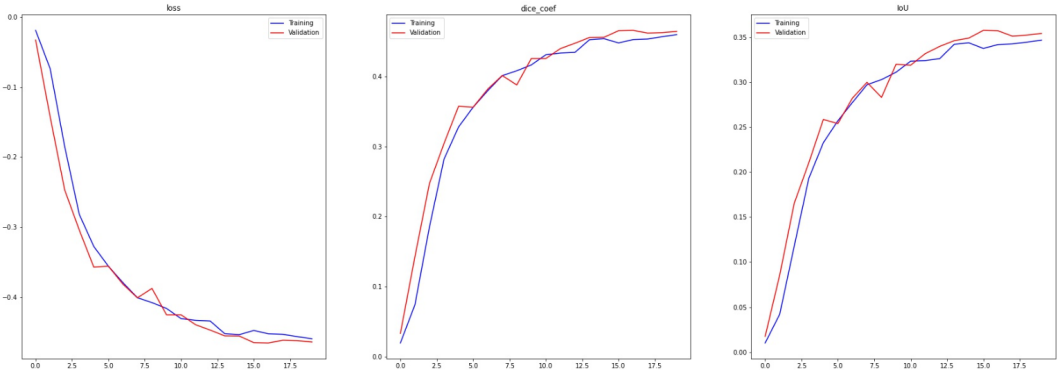


Figure 8: Masks predicted by FCN8s

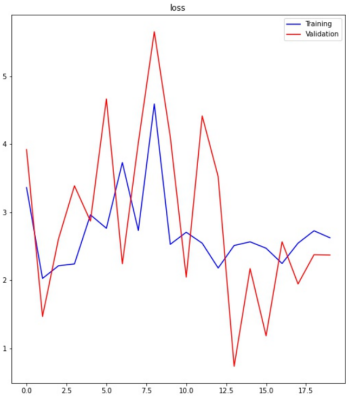
The results of FCN8s are coarse and scattered, due to the fact that it can not be trained using traditional ways



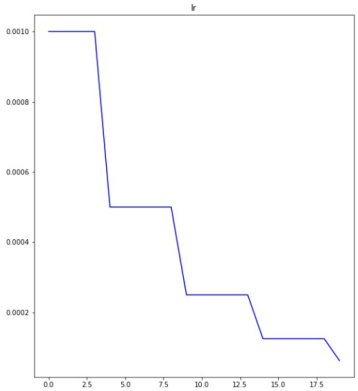
(a) U-Net Loss Function and Metrics during training



(b) PSPNet Loss Function and Metrics during training



(c) Mask-RCNN during training



(d) Learning Rate during training

Figure 9: Loss Function and Metrics of the models, during training

In the case of U-Net and PSPNet, both loss function and metrics have not converged, so we can further train them to achieve better results. Similarly, Mask-RCNN requires more training with more appropriate learning rate schedule.

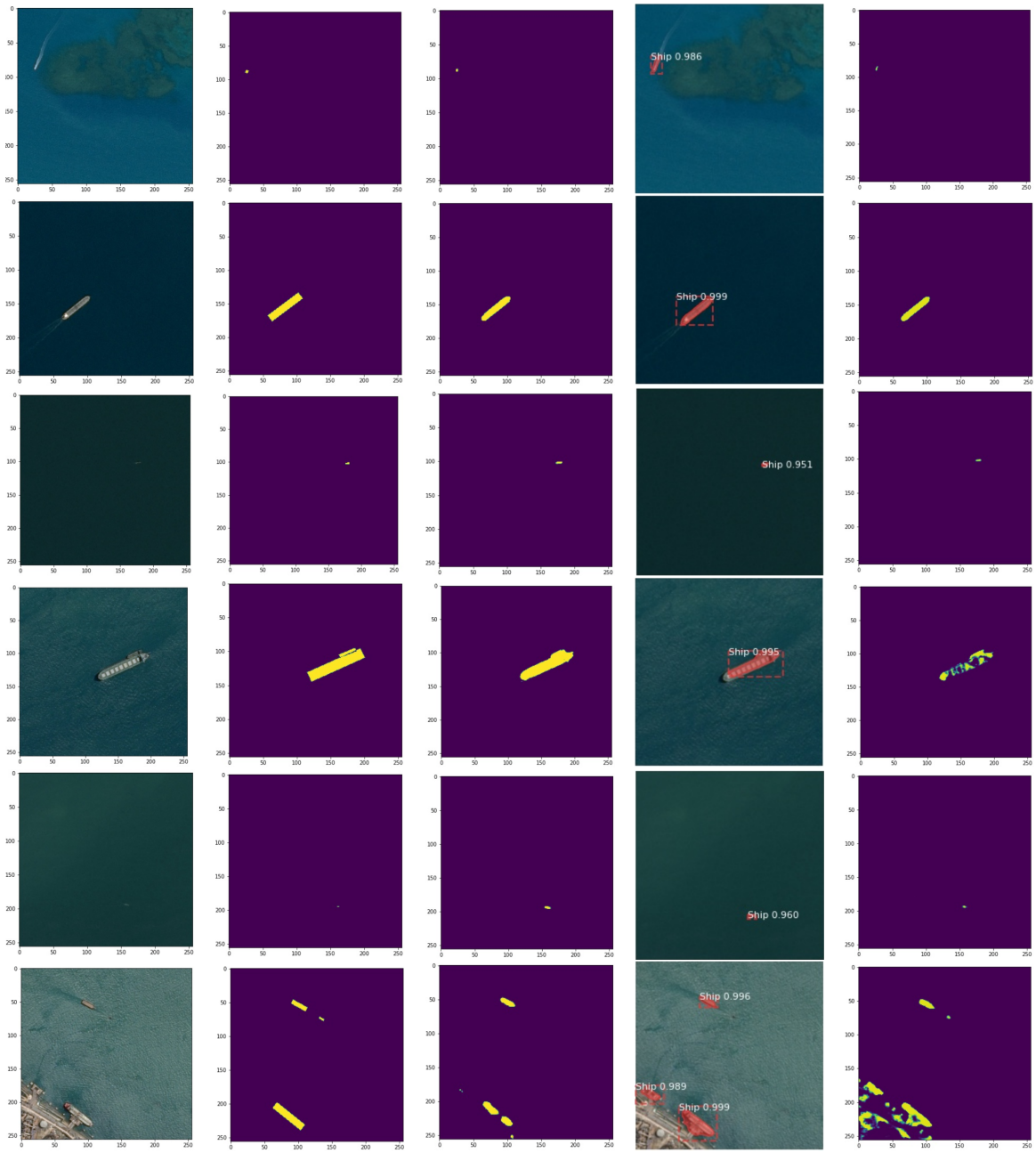


Figure 10: A sample of Model's results (1)

The first column is the images, the second is the true masks (ground-truth), the third is the masks predicted by U-Net, the fourth is the masks predicted by Mask-RCNN, and the last one is the masks predicted by PSPNet.

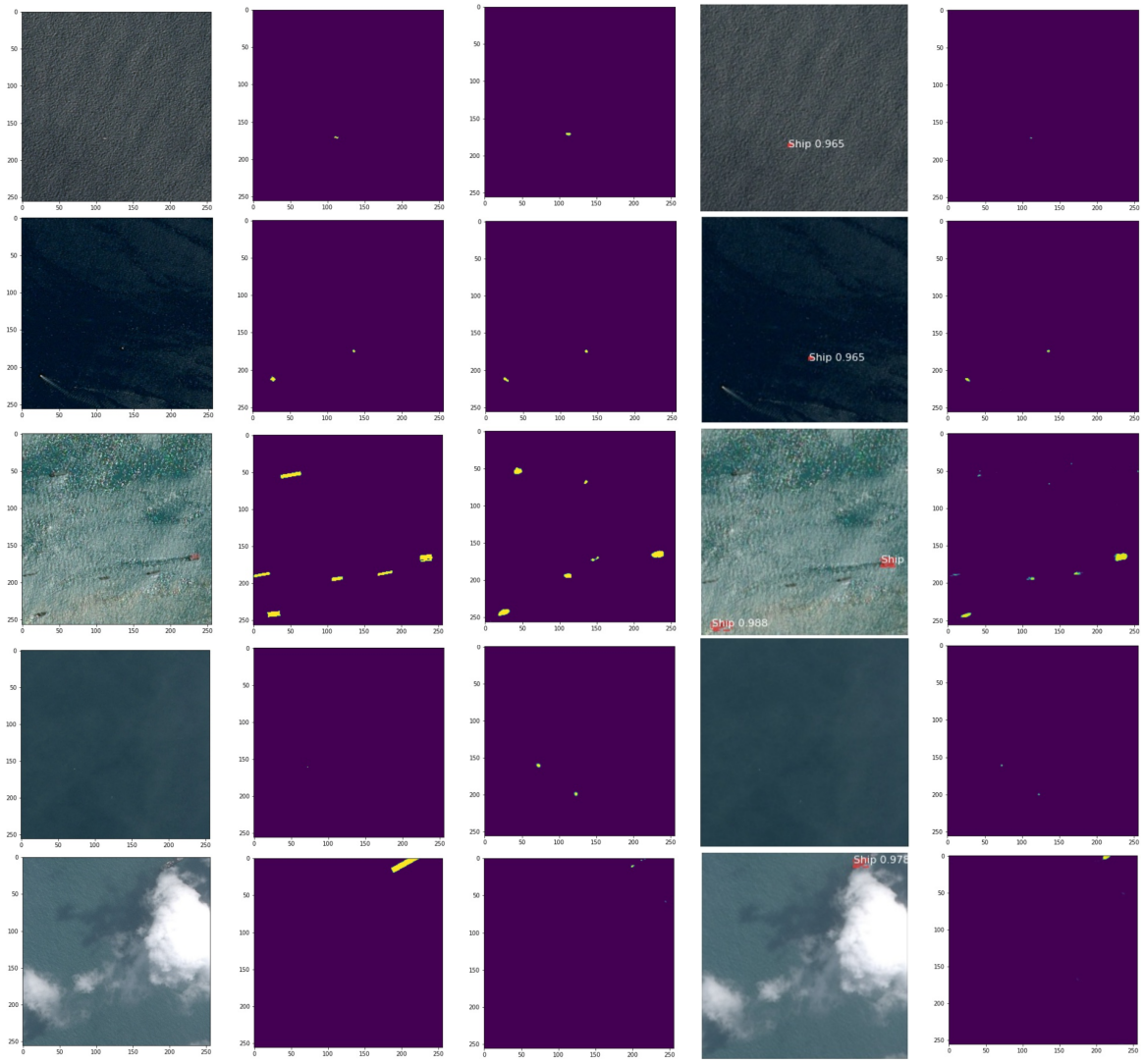


Figure 11: A sample of Model's results (2)

The first column is the images, the second is the true masks (ground-truth), the third is the masks predicted by U-Net, the fourth is the masks predicted by Mask-RCNN, and the last one is the masks predicted by PSPNet.

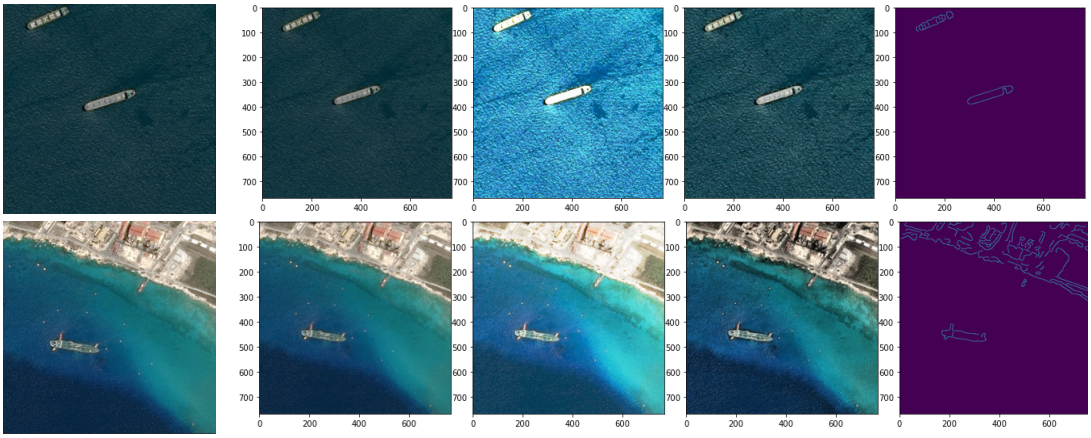


Figure 12: The Examined Filters

First column is the original image, second is the image after applying Histogram Equalization, second column is after applying Adaptive Equalizer, third is after applying Sobel operation, and the last is after applying Canny edge detector.

6.3 Filters

With the view to improve performance, I also tested the models after applying filters on the images. The purpose of the filters are to improve contrast by enhancing the edges, making the ships more distinct by the background. The filters I examined are:

- **Histogram Equalization**[12]: it's an image enhancement method. It provides a better quality of images without losing information and also increases contrast.
- **Adaptive Equalizer**[13]: It's a technique that improves contrast, even in regions that are lighter or darker than most of the image, an property that regular Histogram Equalization lacks behind.
- **Sobel operator**[14]: It's an edge detection algorithm that creates an image, emphasising edges.
- **Canny Edge Detector**[15]: It's an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. Even though it produces fine detailed edges, it dramatically reduces the amount of information of the image.

Examples of images after applying the filters are displayed in Figure 12. Initially I used all the filters but then I noticed that the use of of Sobel operator and Canny edge detector deteriorate the performance of the models. This is sensible for Canny edge detector as it significantly reduces the information of the images, essential for training such large networks. Furthermore, Sobel operator darkens the images and does not enhance the edges as much as the Adaptive Equalizer, for these particular images. For these reasons I decided to omit them and to not include them in the document.

[12] https://en.wikipedia.org/wiki/Histogram$_{e}qualization$
[13] https://en.wikipedia.org/wiki/Adaptive$_{h}istogram_{e}qualization$
[14] https://en.wikipedia.org/wiki/Sobel$_{o}perator$
[15] https://en.wikipedia.org/wiki/Canny$_{e}dge_{d}etector$

The performance of the models using Histogram Equalization and Adaptive Equalizer are depicted in Table 1, and some predicted masks are displayed in Figures 13 and 14.

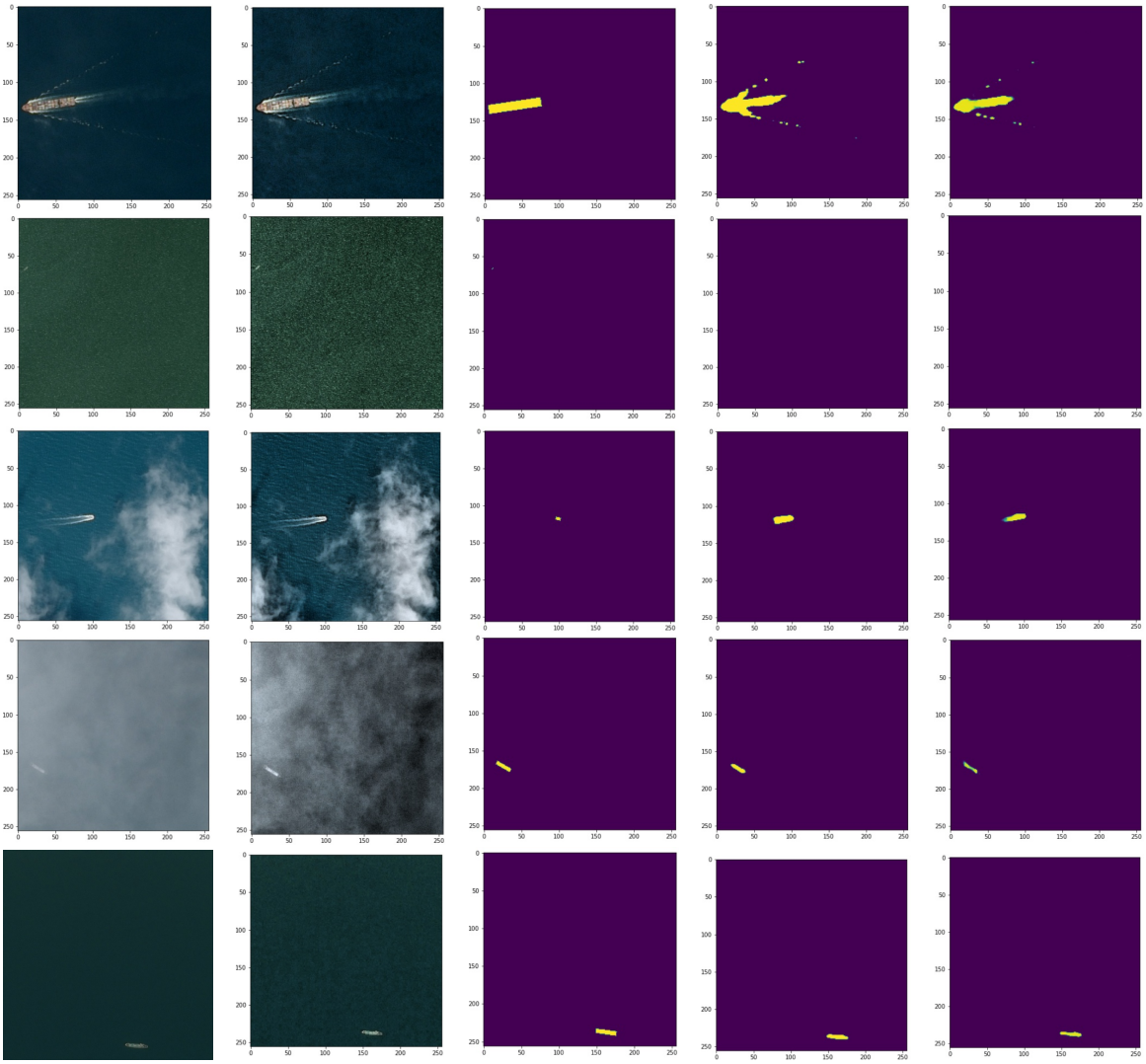


Figure 13: Segmentation after applying Histogram Equalization

The first column is the original images, the second is the images after applying Histogram Equalization, the third column contains the true masks, the fourth contains the masks predicted by U-Net and the last contains the masks predicted by PSPNet

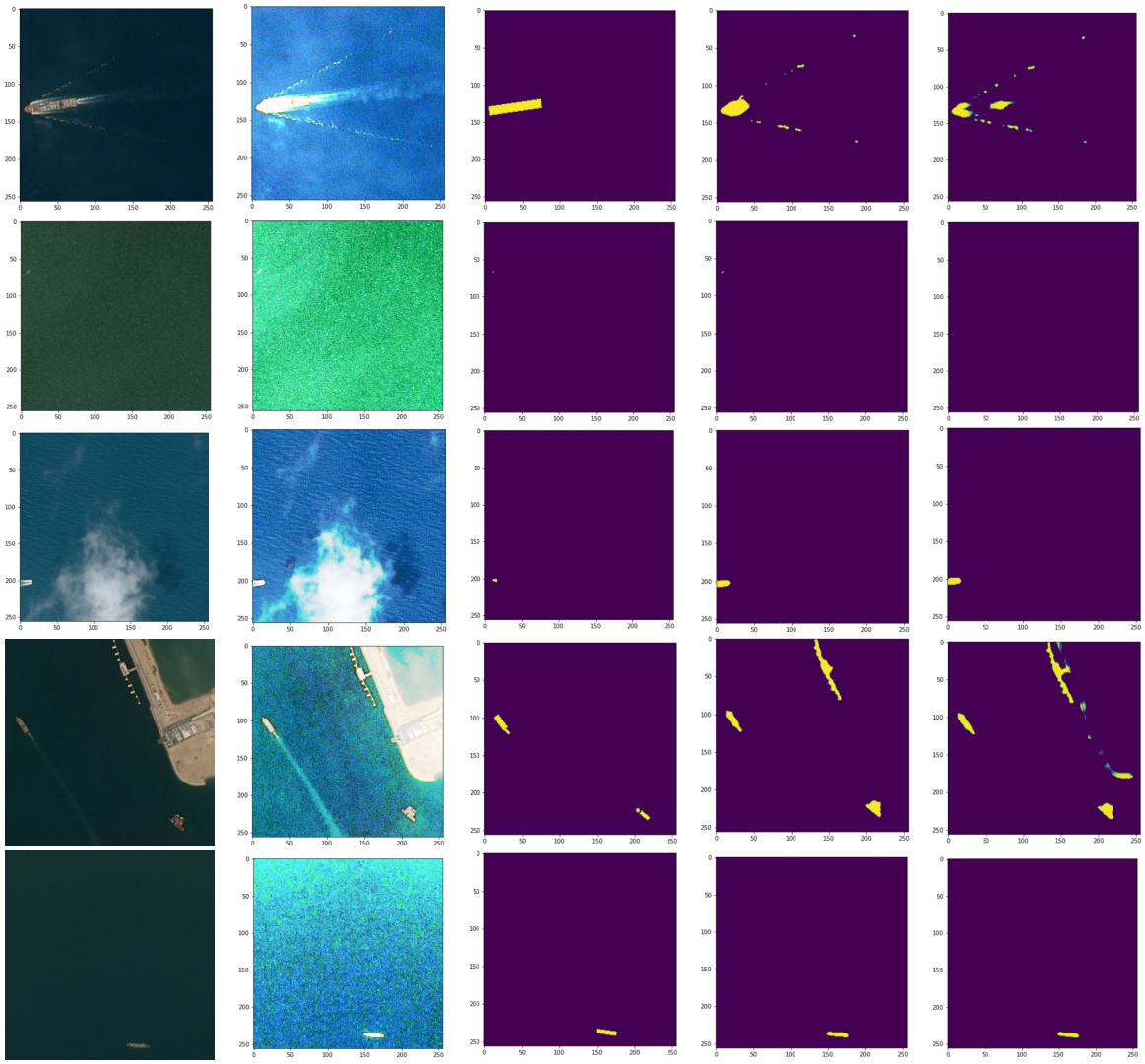


Figure 14: Segmentation after applying Adaptive Equalizer

The first column is the original images, the second is the images after applying Adaptive Equalizer, the third column contains the true masks, the fourth contains the masks predicted by U-Net and the last contains the masks predicted by PSPNet

7 Conclusion and Future Work

In this project I employed some state-of-the-art models in order to predict the segmentation masks of satellite images and detect ships. All the models produce fine results regardless the relatively low DICE coefficient. Furthermore by applying filters that enhances the contrast of images, we can improve performance. If I had to chose one of these models, I would choose either U-Net or U-Net combined with Adaptive equalizer, as it produces very good results and it is faster that the rest, in both training and inference.

However, I reckon that the main reason of the bad performance is the down-scaling of the images, as most of the ships are formed by just a few pixels. However, down-scaling was necessary in order to be able to train in my machine. So, for future work, in order to improve performance, I would not down-scale the images and also use more images for training. The original dataset contains approximately 200K images, while I used just 20K of them. Furthermore, training for more epochs with an appropriate learning rate schedule would probably also boost performance.

Additionally, I would be curious to examine the performance of other models as well. One of them is DeepLabV3 [2], another famous model used for semantic segmentation, which I tried to use but failed to integrate it into my code. Another model I would also considering to try is Random Walk Network [1] which produces fine-detailed segmentation masks, such as those expected from this project. However, its source code is not available yet.

References

[1] Gedas Bertasius, Lorenzo Torresani, Stella X. Yu, and Jianbo Shi. Convolutional random walk networks for semantic image segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6137–6145. IEEE Computer Society, 2017.

[2] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2018.

[3] Dan C. Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2852–2860, 2012.

[4] Yaroslav Ganin and Victor S. Lempitsky. N^4-fields: Neural network nearest neighbor fields for image transforms. *CoRR*, abs/1406.6558, 2014.

[5] Ryuhei Hamaguchi and Shuhei Hikosaka. Building detection from satellite imagery using ensemble of size-specific detectors. In *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 187–191. IEEE Computer Society, 2018.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2980–2988. IEEE Computer Society, 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[8] M. Lavreniuk, N. Kussul, and A. Novikov. Deep learning crop classification approach based on coding input satellite data into the unified hyperspace. In *2018 IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO)*, pages 239–244, 2018.

[9] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[10] F. Ning, D. Delhomme, Yann LeCun, F. Piano, Léon Bottou, and Paolo Emilio Barbano. Toward automatic phenotyping of developing embryos from videos. *IEEE Trans. Image Process.*, 14(9):1360–1371, 2005.

[11] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, 2017.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[13] Jacob Shermeyer, Daniel Hogan, Jason Brown, Adam Van Etten, Nicholas Weir, Fabio Pacifici, Ronny Hänsch, Alexei Bastidas, Scott Soenen, Todd M. Bacastow, and Ryan Lewis. Spacenet 6: Multi-sensor all weather mapping dataset. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, pages 768–777. IEEE, 2020.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[15] A. Yeh, C. Perez, and A. Driscoll. Using publicly available satellite imagery and deep learning to understand economic well-being in africa. 2020.