Machine Learning Engineer Nanodegree

Capstone Project

Oleg Polakow, October 6th, 2019

I. Definition

Project Overview

Shipping traffic is growing fast. More ships increase the chances devastating ship accidents, piracy, illegal fishing, drug trafficking, and illegal cargo movement. This has compelled many organizations to have a closer watch over the open seas. Comprehensive MARITIME MONITORING SERVICES support the maritime industry by increasing knowledge, anticipating threats, triggering alerts, and improving efficiency at sea.

This project is part of the AIRBUS SHIP DETECTION CHALLENGE and aims to support maritime monitoring services by automatically extracting objects from satellite images. In particular, it aims to increase the accuracy and speed of automatic ship detection by using the satellite imagery provided by Airbus. The result of this project is a deployed web application capable of locating ships on the images uploaded by the user.

Problem Statement

The task is to build a model that locates all ships in satellite images and puts an aligned bounding box segment around the detected ships. In order to locate ships and their boundaries in satellite images, one has to use semantic segmentation, which is the process of partitioning an image into multiple segments (SOURCE). More precisely, this is the process of assigning a label to every pixel in an image such that pixels with the same label are either ships or background. This image segmentation task is challenging, since ships within and across images may differ in size. Various scenes such as open water, wharf, buildings, clouds and fog also increase the difficulty of detection. Moreover, most of the images do not contain any ships, which results in an imbalanced classification problem.

Figure: Example of a ship mask

There exist many potential solutions to image segmentation problems, including traditional and deep learning methods; a look at the public kernels of this competition reveals the variety of solutions and their baselines. One of the most compelling approaches is to use a SOTA deep learning architecture such as U-Net or R-CNN. Such model produces an image where each pixel corresponds to either a ship or background, and is used to generate predictions for the test set of the competition as well as classify images in the web application. To tackle the data imbalance problem, the model is further supported by a classifier that is trained to detect whether the image has ships or not.

Metrics

The evaluation metrics are well defined on THE COMPETITION WEBSITE. Since this is a segmentation problem, the predicted segmentation and the ground truth must be compared pixel-wise.

The model's performance is evaluated based on two metrics:

1. the intersection over union (IoU), also known as the Jaccard Index, which is the area of overlap divided by the area of union between the predicted segmentation and the ground truth pixels;

$$IoU = \frac{TP}{(TP + FP + FN)}$$

1. the Dice score (F score), which is the area of overlap divided by the total number of pixels in both images.

$$Dice = \frac{2 \times TP}{(TP + FP) + (TP + FN)}$$

Both metrics are always positively correlated, but it may be the case then that the F metric favors classifier A while the IoU metric favors classifier B (SOURCE). Thus, multiple models can be easily compared. For more details on both metrics, see THIS article.

II. Analysis

Data Exploration

The data was obtained on the AIRBUS SHIP DETECTION CHALLENGE WEBSITE. The dataset for training comprises of 192,556 satellite images of resolution 768x768, located in the folder `TRAIN_V2` (with the total size of about 27GB zipped). Along with these images, there is a CSV file `TRAIN_SHIP_SEGMENTATIONS_V2.CSV` that provides the filename of an image and its ground truth in run-length encoding format. There is also a CSV file `SAMPLE_SUBMISSION_V2.CSV` that contains 15,606 filenames of test images located in the folder `TEST_V2` (about 2GB zipped) and used for generating predictions that are then submitted as part of the Kaggle competition.

The main area of interest is the file `TRAIN_SHIP_SEGMENTATIONS_V2.CSV` which maps images to ship segments. This file contains 231,723 entries, each corresponding to a particular ship segment.

Index	ImageId	EncodedPixels
212424	eadb974dd.jpg	1698 2 2466 4 3233 7 4001 10 4768 13 5536 15 6...
65337	484d10506.jpg	539373 2 540141 6 540908 10 541676 14 542444 1...
107138	75ed4fcfd.jpg	NaN
201931	df36a66aa.jpg	255647 10 256415 10 257183 10 257951 10 258718...
51178	38dabe454.jpg	NaN

Table: A sample from `train_ship_segmentations_v2.csv`

Each row has

- the field `index`, which is a continuously increasing index assigned by pandas,
- the field `ImageId`, which the filename of the corresponding image, and
- the field `EncodedPixels`, which contains ship segments in run-length encoding format. This format is used to encode segmentation boxes of ships; for example, `1 3` implies starting at pixel 1 and running a total of 3 pixels (1, 2, 3).

The dataframe's cardinality is higher than the number of images, which indicates that some of the images (around 16.9%) have more than one ship. Additionally, all filenames can be 1-to-1 mapped to the images in the folder, thus the provided data is complete. By further grouping the dataframe by image, we can calculate the distribution of the number of ships per image. The distribution follows a long tail: there are around 150,000 (or 77.8%) images without ships, while the maximum number of ships per image is 15 in 66 cases.

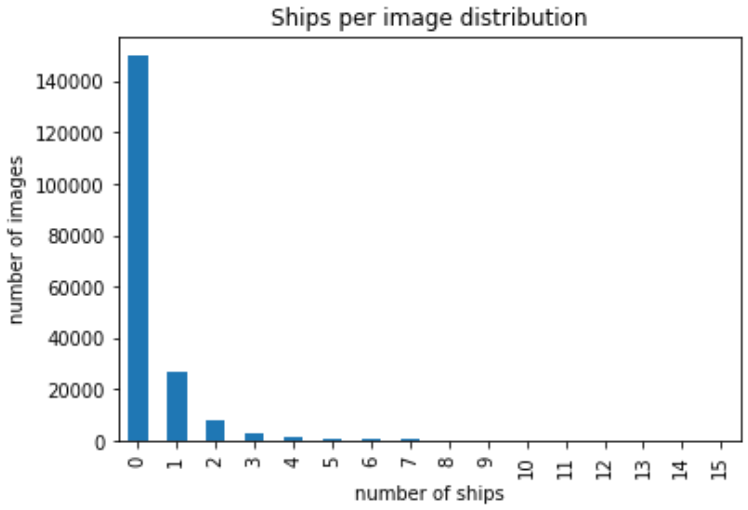

Figure: Ships per image distribution

Another interesting metric is the file size per image in kB. The distribution follows a normal skew distribution, with the minimum of 10 KB, the median of 147 kB, and the maximum of 512 kB. The differences in image size are the result of compression and indicate that most images have a simple, monotonous background (water) that can be easily compressed, while heavier images have more complex visual structures such as land cover or high waves.

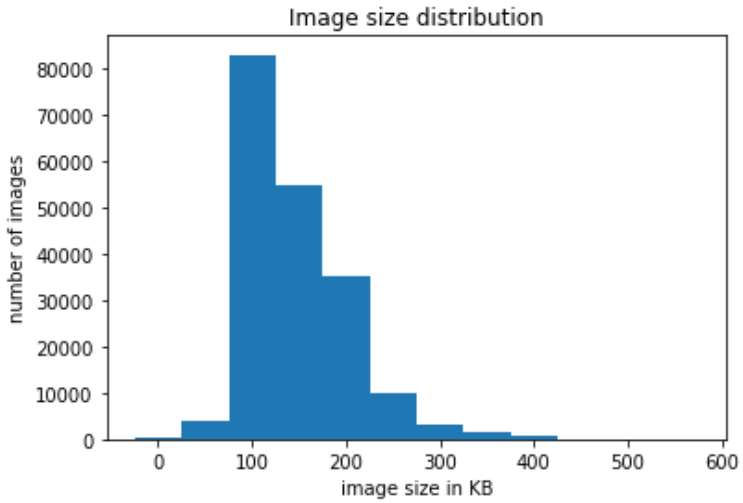

Figure: Image size distribution

Algorithms and Techniques

Image segmentation

The solution for the given image segmentation task is to use U-Net. U-Net is a CNN which takes as input an image and outputs a label for each pixel. It follows a classical autoencoder architecture and contains two building blocks: (1) an encoder structure to capture the context in the image, and (2) a decoder structure to enable precise localization of image objects (segments). The encoder structure (in our case pre-trained ResNet-34) follows the traditional stack of CNN layers to reduce the receptive field, while the decoder structure utilizes deconvolution layers for upsampling such that input and output dimensions match. U-Net initially was developed to detect cell boundaries in biomedical images (SOURCE). The U-Net model takes a three-dimensional image (width, height, RGB) as input and produces a two-dimensional tensor (width, height), with each value being a probability of the respective input pixel being a ship. Thus, it requires both input and ground truth to be of the same shape.

An alternative to U-Net is Mask-RCNN. While Mask-RCNN produces masks for each recognized object, U-Net produces only one mask, hence it is used to predict the union of all masks, followed by post processing to split the predicted mask into one mask per object. U-Net is way simpler than Mask-RCNN and consumes less resources (which is a driving factor), but also requires more complex post processing such as splitting the output image into ship segments. Since most public kernels on the competition website use U-Nets, this seems to be a solid model for this task and also easily comparable to benchmarks.

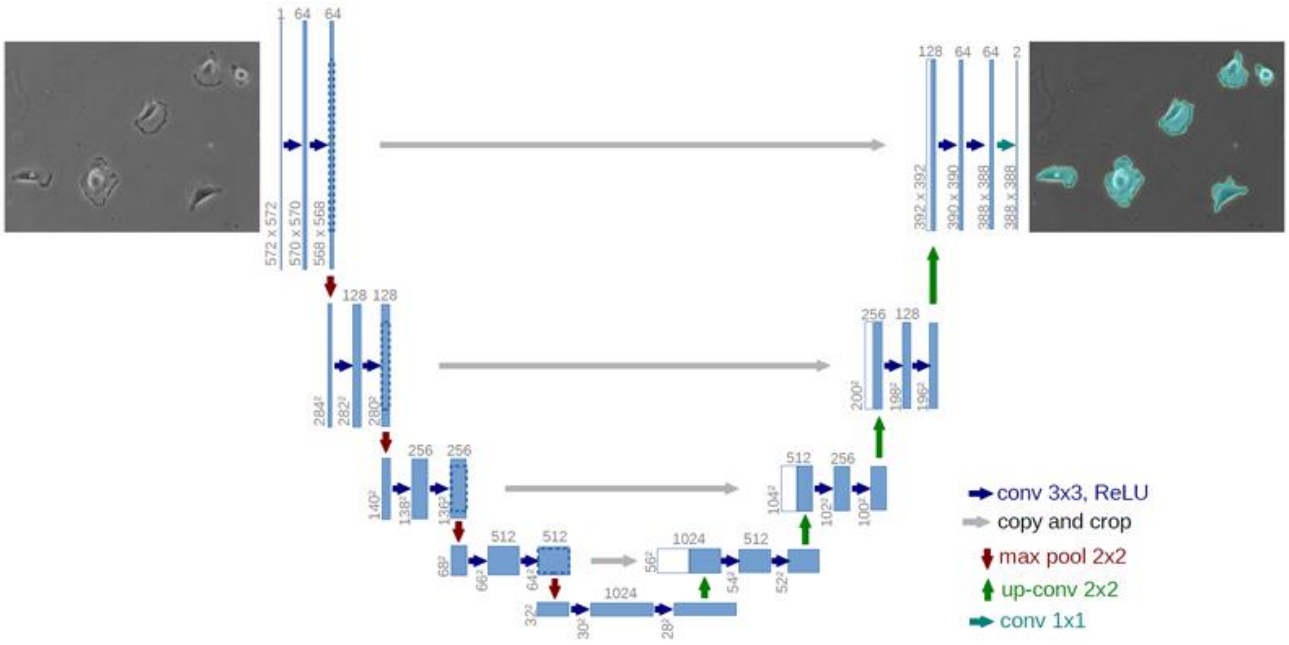

Figure: Architecture of U-Net (SOURCE)

Image classification

To further minimize the False Positives rate and boost the accuracy of the predictions, a binary ResNet-34 classifier was trained to detect whether an image contains ships or not (a similar approach was showed HERE). This way, the images are effectively discarded where the U-Net identified ships but the classifier did not. Furthermore, the training process of such classifier is cheaper and faster (HERE is one of the projects that discusses such approach). Similarly to the U-Net model, the ResNet-34 classifier takes a three-dimensional image (width, height, RGB) as input and calculates the probability of that input containing a ship. Thus, it requires labels to be either 0 (contains no ships) or 1 (contains ships).

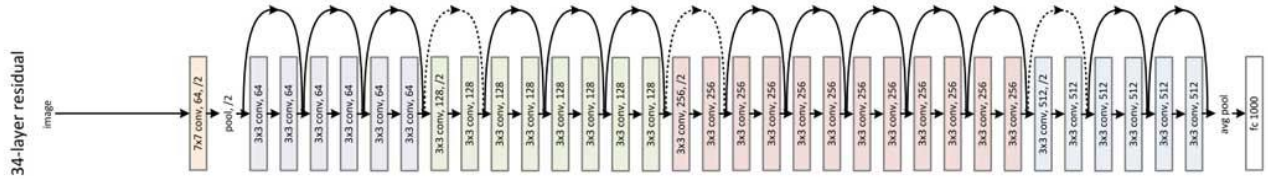

Figure: Architecture of ResNet-34 (SOURCE)

Transfer learning

This project leverages transfer learning to minimize the time and cost of training CNNs. This is done by downloading an already pre-trained on IMAGENET DATASET ResNet-34 classifier and training them in two stages:

1. Feature extraction: Freeze the encoder and train only the final classifier layer.
2. Fine tuning: Unfreeze and train the whole model. Set the learning rate to be smaller for low-level layers.

This way, we can accurately re-train both networks to fit to the satellite imagery.

Benchmark

Since the competition is over, the model's performance will be evaluated by using Late Submission: after a competition ends, one can still receive a score and see where the submission would have ended up on the public leaderboard of that competition. But since this competition recently changed its data from v1 to v2 and some users encountered problems with the way the predictions were evaluated, we will focus on high-rated public kernels instead.

The benchmark for this project was produced by THIS PUBLIC KERNEL with dice of **`0.895`** and IoU of **`0.763`**. This notebook is particularly interesting because the author published his model artifacts and uses similar algorithms and techniques. For example, the author uses the same data, ResNet-34 base model and loss functions. Nevertheless, the author enriched his work by using a model already pre-trained on a binary classification task from another kernel, fine-tuned the model on higher resolution images (384x384 and 768x768) and used data augmentation. My approach is rather to stick to the 256x256 resolution, but to train for longer and with a higher batch size, and use upsampling at test time.

The second benchmark is related to binary classification and was produced by THIS PUBLIC KERNEL with accuracy of **`0.972`**. The author used the whole imbalanced dataset to make predictions, while I'm using a dataset balanced manually. Both kernels are good benchmarks since they give us the exact idea of how the submission was produced instead of just looking at the public LB scores.

III. Methodology

Data Preprocessing

As part of the data cleaning process, 30.459 outliers smaller than 100 kB and larger than 300 kB where removed, because monotonous images bring no new information (low variance) while complex images are distracting for the model and slow down the training process. Additionally, all images and encoded pixels were checked for encoding errors such that we have no surprises during training. As a result, 162,097 images were selected for training and validation.

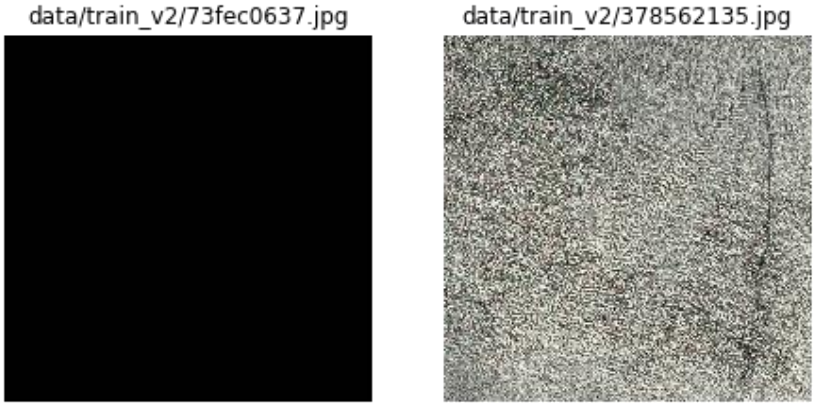

Figure: The smallest and the largest image

In order to prepare the data for the use in image segmentation, the string formatted encodings must be decoded with **RUN LENGTH DECODING** and converted into binary masks where 1 is ship and 0 is background. This is done with a set of encoding and decoding methods that were tested thoroughly to produce accurate results.

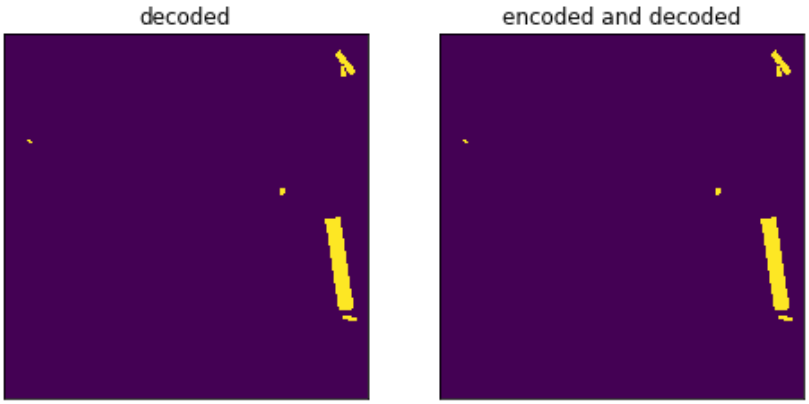

Figure: Visual proof that encoding and decoding methods work properly

Since the size of ship masks is much smaller than the size of images, that leads to imbalanced training with 1 positive pixel per 1000 negative ones. Moreover, training time is longer since we need to process more images in each epoch. So, it is reasonable to drop empty images and focus only on ones with ships. These masks are then fed as labels into the model. The dataset will not be further augmented because of the huge number of diverse examples it already contains. Moreover, for the binary classification task, the data was perfectly balanced by downsampling the images without ships to the number of images with ships.

Implementation

Image segmentation

One of the challenges of this competition is strong data imbalance: the ratio of ship pixels to the total number of pixels is around 1:1000 for images with one or more ships. For images without ships this number is dramatically higher, thus those images are not considered in the training process. Simple loss functions such as binary cross-entropy loss work poorly. To boost predictions, soft dice loss is used. But since it often leads to unstable training, it is usually combined with binary cross-entropy loss (**SOURCE**). The other loss function that is used for strongly imbalanced datasets is focal loss (**SOURCE**). This loss function demonstrates amazing results on datasets with class imbalance of levels 1:10-1000 (**SOURCE**). To bring both losses to similar scale, focal loss is multiplied by some factor.

The U-Net model was automatically constructed using a dynamic U-Net model from fastai, where the left (encoder) part of the model is a ResNet-34 backbone pre-trained on ImageNet, while the right (decoder) part is automatically built by inferring the intermediate sizes. Skip connections are added between both to facilitate the information flow at different abstraction levels. Using a ResNet34 model on pre-trained on ImageNet dataset allows us to have a powerful encoder without a risk of overfitting and necessity of training a big model from scratch. The total capacity of the

model is around 21M parameters. The model's constructor requires passing the data loaders, the encoder architecture, the metrics, and the loss function.

Due to limited computational resources, smaller image sizes (256x256) and ResNet-34 as encoder structure were used, as opposed to the ResNet-50 or higher. ResNet-34 is commonly used as an encoder for U-Net and SSD, boosting the model performance and training time.

After the data preprocessing steps, the model was trained in two stages: (1) freeze the encoder and train the decoder (feature extraction), and (2) unfreeze the encoder and train the whole model (fine tuning). In the first stage, the head has been trained with the batch size of 28 and learning rate of 0.001 for 5 epochs. In the second stage, the whole model has been unfrozen, the batch size has been decreased to 24 to fit into GPU, and the model has been trained for 5 more epochs with a variable learning rate (the lowest layer had the LR of 0.000001, while the highest had 0.0001, since lower layers do not vary much from one image dataset to another). The training has been run with learning rate annealing: a periodic LR increase followed by a slow decrease drives the system out of steep minima towards broader ones that enhances the ability of the model to generalize.

Image classification

Binary classification is simpler in the way that it operates on binary labels and requires as little as no preprocessing. Similar to the segmentation task, first the head of the classifier was trained (for 5 epochs with LR of 0.01), and then the whole classifier (for 10 epochs with variable learning rate from 0.00001 to 0.0001). For evaluation, a simple accuracy score was used since the dataset has been perfectly balanced beforehand.

Refinement

The work had gradual improvements over time:

The first iteration included a single U-Net on top of a basic ResNet-18 model that failed to overfit to the data despite efforts to reduce regularization: the training loss was always higher than the validation loss. The second iteration included the final U-Net described earlier. Various settings have been tried out, such as weight decay, a different number of epochs (the total number was limited by GPU limit), as well as variable learning rates. Additionally, by using differential learning rates, the validation loss decreased from 0.164680 to 0.120643. After the U-Net finally produced satisfactory results, the third and final iteration included a ResNet-34 classifier to account for false positives. The model had the same "knobs" as the U-Net, thus hyper-parameters were selected similarly (and manually, since neural networks are less fit for automated hyper-parameter search).

Modern convolutional nets support input images of arbitrary resolution. To decrease the training time and increase generalization, one can start training the model on low resolution images first and continue training on higher resolution images. Thus, both models were also trained on 384x384 and 768x768 resolutions. But to my surprise, the model's performance on 256x256 images was the best. An increased image size requires to decrease the batch size to fit the parameters into GPU. If then the batch size becomes too small (as for 768x768 resolution), it provides a further regularization to the model (less averaging over noisy data) and thus makes the training process less efficient. The resolution of 256x256 is optimal, since you can always upsample the predicted mask to 768x768 by

either resizing arrays with `SCIPY`, upscaling images with `PIL`, or upsampling tensors with `TORCH.NN.UPSAMPLE`.

IV. Results

Model Evaluation and Validation

To test that the U-Net model produces accurate results, the predicted masks were compared with true masks visually, and also the distribution of predicted ships was compared to the true distribution of ships. Both distributions are very similar, which hints on the great predictive power of the model. The only deviation are the images that are predicted to contain no ships (false negatives) while the validation set contains only images with ships. But this behavior is expected, since many ships are so small that they are hard to capture, thus produced masks are blank.

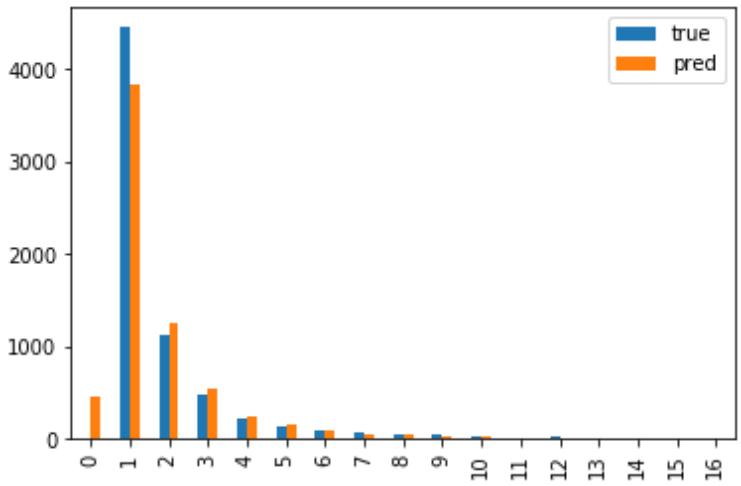

Figure: True and predicted ship count distribution for the U-Net model

Both models were tested visually on images from the test set. For example, the U-Net model was able to capture fine-grained details that it was not trained for, such as antennas and the shape of ships, even if training set is composed of straight bounding boxes. Visual testing also validated the robustness of the model and exposed some weak points (such as small islands identified as ships, or oil platforms). The results align with the expectations and this can be also proven by using the deployed web application.

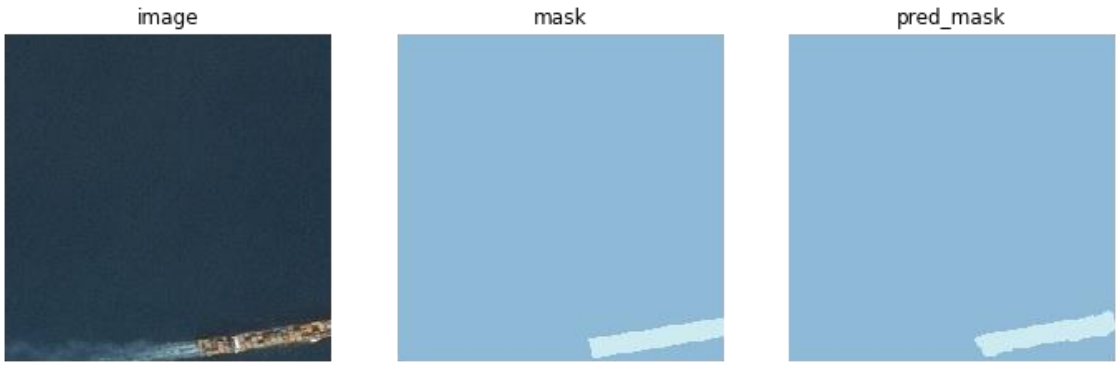

Figure: The U-Net model produces a mask that follows the shape of a ship

Justification

The results of the final models outperform both benchmarks and all public kernels on the newest data (of version 2). The U-Net model scored **`0.917`** dice and **`0.848`** IoU on 6,657 images from the validation set, while the classification model achieved **`0.979`** accuracy on a balanced validation set consisting of 13,314 images. The (late) submission to the competition was made that produced a public LB score of **`0.71969`**, which is the 155th place out of 881. Furthermore, when using the classification model in conjunction with the U-Net model, the submission scored **`0.72019`**, which is the 152th place.

In contrast to the benchmark solutions, the developed solution takes the advantage of low resolution images to train the algorithms for longer and fit more to the data, and then uses test-time upsampling to produce higher resolution images with little quality loss. On the other hand, the benchmark solutions train also on higher resolutions but for shorter periods of time, failing to completely fit to the imagery. Thus, the results of this project are stronger.

V. Conclusion

Free-Form Visualization

Instead of providing any static visualization, the reader can visit the deployed **SHIP DETECTOR** application to play with the discussed models. The user can upload his own image or select an example one, and the application will return the same image but with predicted mask as overlay.

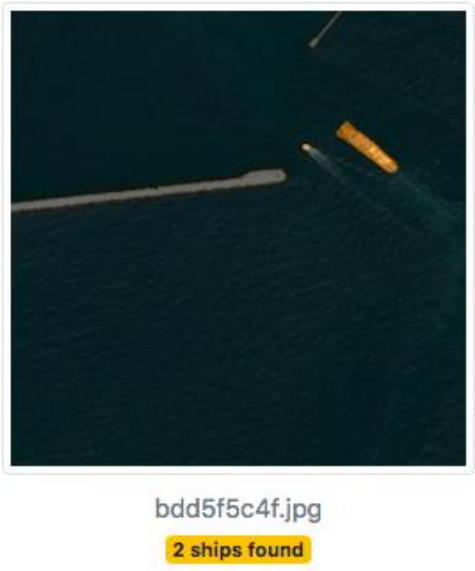

Figure: Result example from the web application

Reflection

In this project, I combined two deep learning models to highlight ships on satellite imagery provided by the **AIRBUS SHIP DETECTION CHALLENGE**. The first model predicts the masks of ships and the second model classifies whether the predictions of the first one are false positives. The interesting part of this project was data pre- and post-processing that consumed over 80% of time to decode

encoded masks into images and encode them back for submission. Another interesting aspect was observing how both models advance over time, although the process was a lengthy one. The difficult aspects included the large dataset, data imbalance problem, as well as limited GPU resources that forced me to work with lower resolutions only. The provided solution combines many best practices for image segmentation and classification problems and can be adapted for the use in other satellite imagery problems such as **UNDERSTANDING CLOUDS FROM SATELLITE IMAGES** competition.

Improvement

The solution could be further improved when reversing the order of training both models and injecting the trained ResNet-34 classifier as encoder into the U-Net, such that the U-Net model could directly benefit from the information on satellite imagery. Furthermore, training on higher resolution, with larger batches and for longer time could result in a better score, although it requires a powerful hardware to begin with. Test-time augmentation is another technique that can improve the score but it would further increase the inference duration. Also since most winners in Kaggle competitions describe their solution in blogs, one has tons of new feature ideas. But the idea of this project was not collecting best practices but implementing something given just the public kernels on the competition website and own ideas.