Capstone Proposal

Domain Background

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. One of the leading organizations that offers such services is Airbus, which aims to automatically extract objects from satellite images with significative results but no effective operational effects. To increase the accuracy and speed of automatic ship detection, Airbus turned to Kagglers and created the AIRBUS SHIP DETECTION CHALLENGE.

Problem Statement

The objective is to build a model that locates all ships in satellite images and puts an aligned bounding box segment around the detected ships. 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. The problem is well defined by Airbus and there exist many potential solutions, including traditional and deep learning methods; a look at the public kernels of this competition reveals the variety of solutions and their baselines. My goal is to beat those baselines by using state-of-the-art methods in image segmentation.

Datasets and Inputs

The data will be obtained on the AIRBUS SHIP DETECTION CHALLENGE WEBSITE. The dataset for training comprises of around 200k satellite images of resolution 768x768, located in the folder `TRAIN_V2` (with the total size of about 27GB zipped). Many images do not contain ships (about 22%), and those that do may contain multiple ships. 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. This format is used to encode segmentation boxes of ships. There is also a CSV file `SAMPLE_SUBMISSION_V2.CSV` that contains 15k 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.

Solution Statement

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. To tackle the segmentation task, a SOTA deep learning architecture such as U-Net or R-CNN will be used. After training such classifier, the labels for each image in the test set will be produced. The predicted labels will then be parsed into segments and their location and boundaries will be encoded for submission.

Example: Ship segments (SOURCE)

Benchmark Model

The benchmark for a Kaggle competition is its public and private leaderboards. 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. Furthermore, the model's validation scores will be compared to that of public kernels with similar approaches, such as THIS and THIS.

Evaluation Metrics

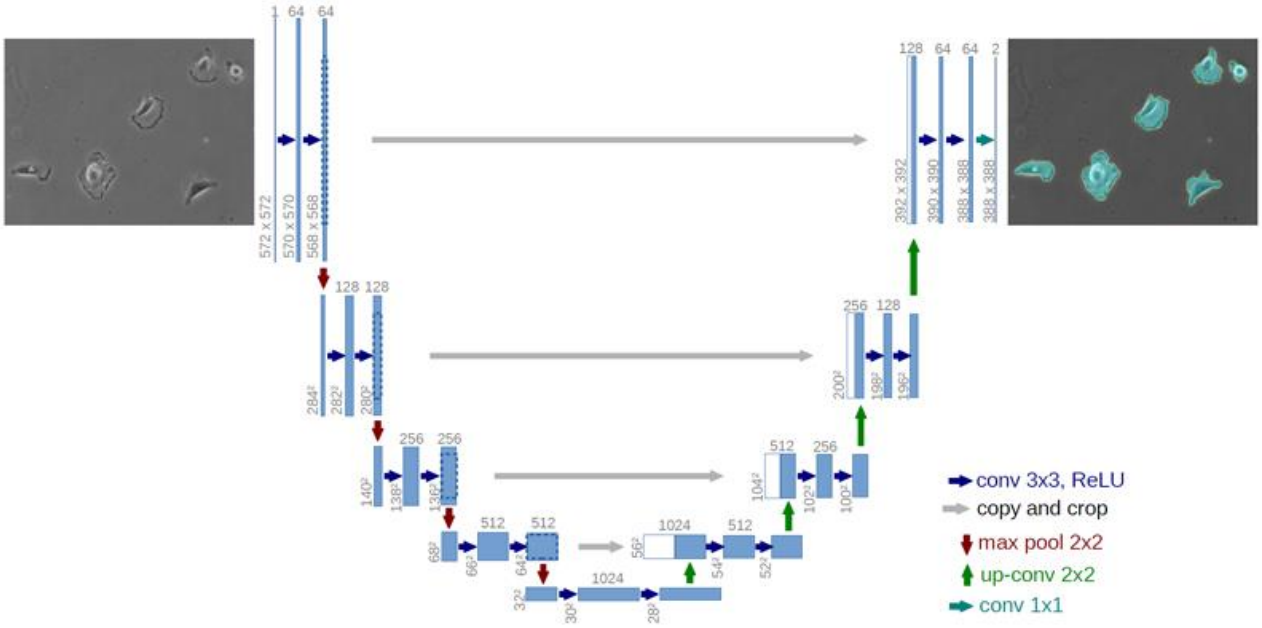
The evaluation metrics are well defined by the competition HERE. Since this is a segmentation problem, the predicted segmentation and the ground truth must be compared pixel-wise. The model's performance will be evaluated based on two metrics: (1) the intersection over union (IoU), also known as the Jaccard Index, which is the area of overlap divided by the area of union between the predicted segmentation and the ground truth pixels; and (2) the Dice score (F score), which is the

area of overlap divided by the total number of pixels in both images. For more details, see THIS article. 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. The final score in this competition is then produced by calculating the mean F2 Score at different IoU thresholds.

Project Design

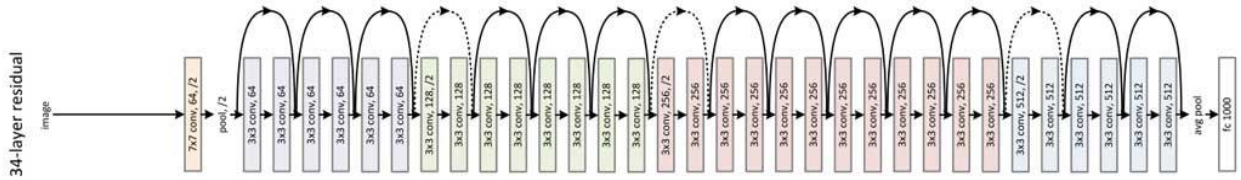
In order to prepare the data for the use in image segmentation, the string formatted encodings will be decoded with RUN LENGTH DECODING and converted into binary masks where 1 is ship and 0 is background. Since the size of ship masks is much smaller than the size of images, that leads to imbalanced training 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 will be fed as labels into the model. The dataset will not be further augmented because of the huge number of diverse examples it already contains.

The proposed model 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 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). 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 (a driving factor), but also requires more complex post processing.



Example: Architecture of U-Net (SOURCE)

The U-Net model will be constructed using fastai. Specifically, the dynamic U-Net model will be used, where the left (encoder) part of the model is any backbone pretrained on ImageNet (in this case ResNet34), while the right (decoder) part is automatically built by inferring the intermediate sizes. Due to limited computational resources, smaller image sizes (256x256) and ResNet-34 as encoder structure will be used, as opposed to ResNet-50 and higher. ResNet-34 is commonly used as an encoder for U-net and SSD, boosting the model performance and training time. After data preprocessing, the model will be 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).



Example: Architecture of ResNet-34 (SOURCE)

To further minimize the False Positives rate and boost the accuracy of the predictions, a binary ResNet-34 classifier will be trained to detect whether an image contains ships or not. This way, the images will be 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). Since classification problems usually suffer from imbalanced data themselves, the data will be manually balanced by downsampling the images without ships to the number of images with ships. At the end, both models will be used to produce the predictions for the Kaggle competition.

After each training stage, the model's artifacts will be saved to the disk. The best model will then be deployed into a web application that will receive an image and return the same image but with all ships highlighted and counted. The application will have (1) a user interface where the user can upload one or multiple images for detection, and (2) an API endpoint that receives an image and returns the one-length encoding of all ships identified on this image. The application will be hosted on Render.