Maritime Ship Detection Using Synthetic Aperture Radar Satellite Imagery

Alexander Makhratchev and Sean Ng

University of California: San Diego

March 14, 2023

Abstract

Satellites are being launched into space at an exponential rate and are able to produce high quality images in relatively short intervals of time on any part of Earth. The amount of data and types of it are also increasing significantly and in this paper we specifically use Synthetic Aperture Radar (SAR) satellite imagery in order to detect ships traveling through bodies of water. We created a ship counting tool that intakes a start date, end date, and an area of interest and returns the number of ships for each day between the two dates. We propose a new method where the images are first classified as either offshore or inshore and a separate object detection algorithm counts the number of ships per image. The classifier and object detection networks are trained using the Large-Scale SAR Ship Detection Dataset-v1.0 (LS-SSDD-v1.0) and deployed on Google Earth Engine.

1 Introduction

With recent advances in algorithms and technology within the realm of computer vision, satellite imagery has become an applicable field of study. Roughly 1700 commercial satellites were launched in 2021, bringing the total to 5,000 satellites orbiting Earth by the end of 2021. These satellites can complete an orbit around Earth in under 2 hours and take 5m resolution pictures from over 500 km away providing vast amounts of image data. Using computer vision techniques, we are able to obtain necessary information from the images. Specifically, analyzing satellite imagery for ship detection has been subject to a lot of research where different satellites and detection models have been created and improved on for better speed and accuracy. One of the most widely used image data within this sphere has been the Synthetic Aperture Radar (SAR) images. Previously, optical satellites have been known to produce some of the highest resolution images, however with the drawback that it can only produce high resolution images under certain time of day and weather conditions. SAR's strengths lie in the fact that it use radio signals to produce high quality images even if it is cloudy or at night time.

Ship detection has been a popular subject for research within the fields of object detection. And with the use of SAR satellite imagery, machine learning and deep learning models are able to more accurately detect vessels across the globe. With uses in various economic tasks, search and rescue operations, aiding in determining military decisions, managing ship traffic, and monitoring possible illegal activity, not only do models have to be accurate, but they also need to be quick as well. Previously, other researchers were only to do classification or extract the region of interest from an image. Now, with the emergence of deep learning, we are able to automatically detect vessels without the need of handcrafted features or large amounts of computational resources. First we use classical machine learning to classify an image and then utilize deep learning models like RetinaNet and Faster R-CNN to produce accurate and rapid results for the task of detecting ships in SAR images.

2 Literature Review

A research group in China [15] released their paper on ship detection and the accompanying dataset: LS-SSDD-v1.0. Their goal was to create the most accurate dataset for cargo ship detection from SAR

images. In order to provide the most accurate ground truth annotations, the team used Automatic Identification System (AIS) to locate ships in a specific image and label them. Additionally, the images were overlaid on top of an optical Google Earth engine map to make sure no small islands were annotated as ships. The images were collected in large swaths up to 250 km in size from 15 different locations around the globe, mimicking the data the algorithm will be deployed on. Ships of various sizes were annotated including large cargo container ships and smaller fishing vessels. The large images were divided into smaller ones so they can be easily fit into a training network. None of the images were discarded even with no ships. Overall, this dataset attempts to improve every aspect of previously released datasets for SAR ship detection and that is why we will be using it in this paper.

There are a few other datasets for this task, but we did not choose them due to a variety of reasons. The first one being SSDD [6] dataset, Figure 1 Row 1. It contains 1160 SAR images that are 500×500 pixels and obtained from a mix of satellites with varying resolutions. Two major drawbacks are that annotations are not double checked with Google Earth or AIS and there are numerous repeated scenes, which does not benefit our model. Another dataset we considered was SAR-Ship-Dataset, [14] which contains 43k SAR images of size 256×256 pixels and contains 60k ships, Figure 1 Row 2. Once again the images were collected from many different satellites so their resolutions differ. One change from the previous dataset was that all images without ships were discarded, which increases false positive rate. The labeling on this dataset was done by SAR experts, which still leaves room for error. To combat some of these issues, AIR-SARShip-1.0 [13] a dataset was proposed, which contained 31 SAR images of size 3000×3000 pixels, Figure 1 Row 3. However, these images contained too much noise which reduced the effectiveness of the ship detector. Also, the ground truth was determined by experts solely. The last dataset we considered was the High-Resolution SAR Images Dataset [12] which contained 5604 SAR images of size 800x800 pixels of various resolutions, Figure 1 Row 4. In this dataset, the ground truth was determined using Google Earth, but images with pure backgrounds were abandoned. Thererfore, we chose the LS-SSD-v1.0 dataset to use for our experimentand images from the dataset we used can be found in the data section.

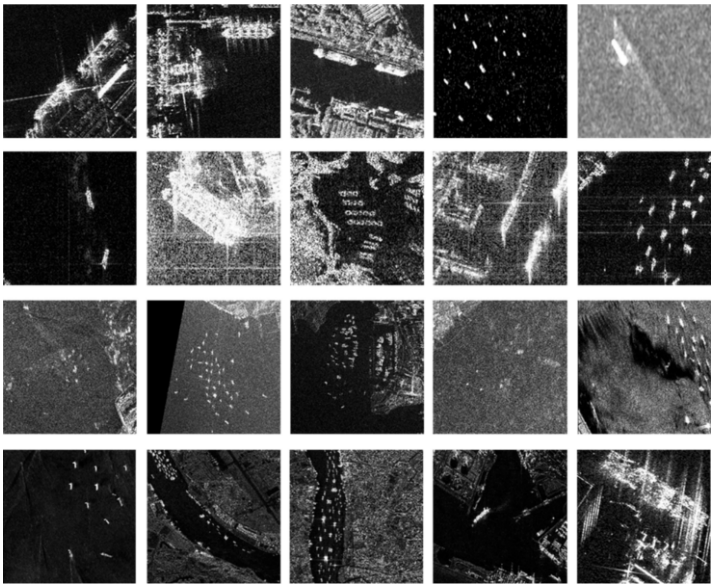

Figure 1: Each row contains sample images from each of the datasets in the order: SSDD, SAR-Ship-Dataset, AIR-SARShip-1.0, High-Resolution SAR Images Dataset. Image Source: [15]

Kanjir et al. released a paper reviewing 119 papers on ship detection using optical satellite images. In addition, they also compared and contrasted the different imaging systems used for vessel detection such as optical and reflected infrared, hyperspectral, thermal infrared, and radar [5].

Each of the imaging systems has their pros and cons, but radar and specifically Synthetic Aperture Radar (SAR) satellites are the best sensor for ship detection. Hyperspectral sensors and thermal infrared both have too low of a spatial resolution to be used for ship detection. While optical satellites have a high spatial resolution, bad weather conditions may make it hard for an optical satellite to get clear images. For example, Figure 2 below showcases an example of an optical image that is covered

mainly by clouds and can make classification quite difficult. In addition, it is also dependent on the reflection of sunlight off the Earth's surface, so if it is an overcast day or even at nighttime, the optical satellite cannot get clear images. SAR on the other hand is able to consistently collect high resolution images over a wide area due to its use of radar signals that are bounced off the Earth's surface. In addition, radar signals allow the SAR satellite to be able to collect data regardless of time of day as well as most weather conditions. Not only that, but larger ships often appear as bright objects in SAR images because many are made of metal and contain sharp edges that reflect the radar signals very strongly and allow for better ship detection. There are some drawbacks with SAR images being noisy, having a tough time detecting small ships, being sensitive to high winds and certain sea conditions, and classification of vessels is difficult [5]. Despite these drawbacks SAR satellite imagery is the best system we have for collecting consistent high quality images.

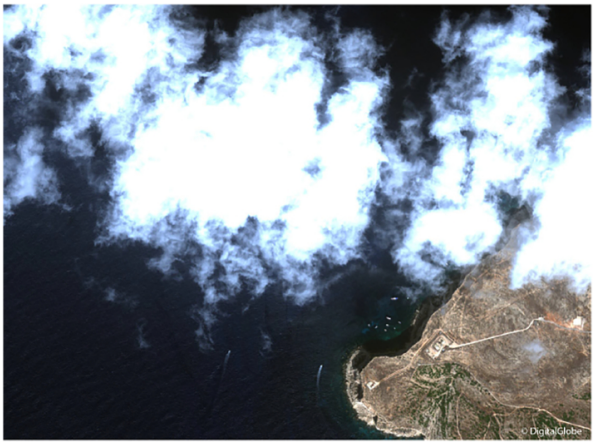

Figure 2: Clouds are a major nuance when dealing with optical data. Image Source: [5]

Chang [4] firstly explores older object detection methods and models to give background, but more importantly compares and contrasts the more recent sophisticated deep learning models that have been popular. They compare the R-CNN, Fast R-CNN, Faster R-CNN and finally the YOLOv2 models in the order of how each model improves on the other. R-CNN is the slowest model out of all of them which led to the development of the Fast R-CNN model that improves runtime by utilizing a much faster softmax function instead of support vector machines [4]. Both the R-CNN and Fast R-CNN models are two stage detector models, and use a selective search algorithm to identify a small subset of regions that might contain a ship, and then use a region based CNN to classify it. The Faster R-CNN model significantly improves on this by allowing a neural network with attention to predict the proposed areas that might contain a ship, and use the same convolution network to perform the classification [10]. The Faster R-CNN model also uses anchor boxes in order to predetermine areas of interest for each single proposed area allowing it to detect objects at different scales and aspect ratios. In essence, the Region Proposed Network guides the region-based CNN where to look for the object, which overall reduces the inference time compared to smaller models such as YOLOv2.

The You Only Look Once (YOLO) model performs faster than Faster R-CNN due to the fact that the whole model is contained in a single network. This allows for easier optimization during training. In addition, since the Faster R-CNN model only looks at regions in isolation, it often misidentifies parts of the background as objects. On the other hand, the YOLO model sees the whole image during training and is able to learn contextual information which leads to better classification. In their results,[4] show that YOLOv2 had around 20% better accuracy on two different datasets than Faster R-CNN and also performed around 5.8 times faster than Faster R-CNN [4]. These performance results made it quite clear to us that the YOLO models would be beneficial for our task.

The last family of object detection algorithms that we looked at were Single Shot Detector (SSD). They are similar in speed to the YOLO model but offer the accuracy close to that of the Fast R-CNN models. RetinaNet is one of the most common models in the SSD family and consists of two main parts. The first part is the pretrained image classification model that is used for feature extraction from each region. It also utilizes anchor boxes in order to find the initial shape of the object. The second part of the model is the SSD head, which consists of one or more convolutional layers and

produces multi scale feature maps that are later converted to bounding boxes and class predictions. This differs from YOLO in the way that overlapping boxes are dealt with: YOLO uses non-maxima suppression, while RetinaNet treats it like a regression problem. Using techniques from both types of models, SSD are able to achieve two stage detector model accuracy while maintaining relatively low inference times [8].

3 Data

In this paper we will be using the LS-SSDD-v1.0 dataset which is obtained from the C-band from the Sentinel-1 satellite. Because vessels on water experience higher backscattering values, only the co-polarization channels (VV) are kept from these images. There are 15 large scale images in this dataset of size 24,000 x 16,000 pixels. The first ten images are the training set and the last five images are the test set. In order to be able to load the data onto a GPU during training, each image is cut into 600 equal sized pieces of 800×800 pixels. This data we will use for training most resembles the images we intend to deploy our pipeline due to the large size of the images as well as the quality and resolution of them. Figure 3, Figure 4, and Figure 5 are some different types of images found in the dataset.

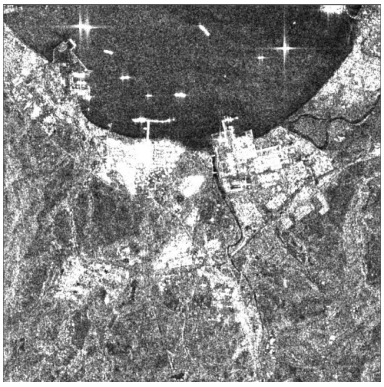

Figure 3: Inshore image with ships

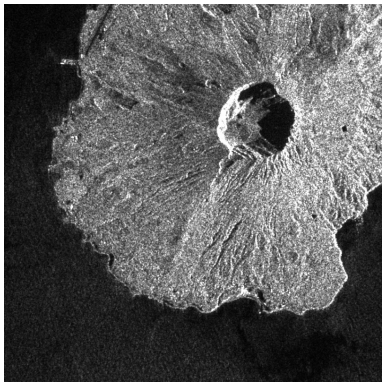

Figure 4: Inshore image without a ship

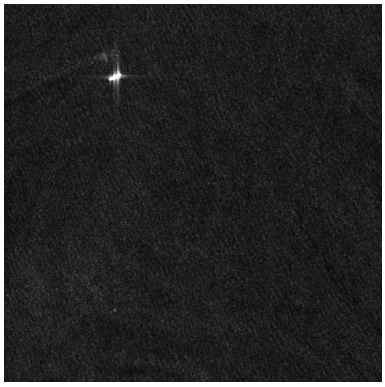

Figure 5: Offshore image with a ship

Below in Table 1, we can see the distribution of the image data for inshore and offshore as well as ships or no ships. The large difference in the number of offshore and inshore images motivated us to split the ship detection problem into two parts. However, we did not train each model on the subset of inshore and offshore data separately, because we did not have enough data for each one. Only a third of the images were labeled inshore and offshore so we created a classifier trained on that data to classify the rest of the images to be inshore or offshore. For the object detection model training, we only used images with ships in them.

Table 1: Inshore and offshore ship image percentages

	Inshore	Offshore	Totals
Ships	4%	16%	20%
No Ships	24%	56%	80%
Totals	28%	72%	100%

4 Methods

4.1 Inshore-Offshore Classifier

To build our inshore-offshore classifier, we utilized the the portion of the Large-Scale SAR Ship Detection Dataset-v1.0 (LS-SSDD-v1.0) [15] dataset that contains inshore and offshore labels due to the fact

that we need our images to be labeled when training our classification models. For training, we used a 70/30 train to test split ratio to prevent overfitting. For each image, we extracted a set of features to be used within the model. We took the 30th, 50th, 80th, and 90th percentiles of pixel values. We decided to use the 80th and 90th percentiles because when looking at images from the training dataset, we found that images with any sign of land even if the majority of the picture is ocean, is considered to be inshore. For example, if looking at Figure 6, the majority of the image is blank ocean, however there is a little bit of land present and therefore the image is labeled inshore. Due to this, we wanted to use higher percentiles as features to be more robust towards these kinds of cases. In addition, the 30th and 50th percentiles we also used to give a better representation of the images and looking at the lower percentiles do still prove to be useful. To find the best model for classification, we gathered a few different machine learning models from scikit-learn (K-Nearest Neighbors, SVM, Decision Tree Classifier, Random Forest Classifier, AdaBoost, and Gaussian NB) into a list and looped through each model to test its accuracy. For each individual model, it was fitted and tested on the training and test sets 10 times and the scores for those 10 runs were averaged to give the score for that model. From this, we found that the K-Nearest Neighbors worked the best. Then to find the best parameters for our model, we used 5-fold grid search cross validation. We tested values 1, 2, 5, 10, 15, 20 for the number of neighbors, uniform and distance for the weights parameter, and different algorithms such as ball_tree, kd_tree, and brute. From our cross validation, we found that the best parameters for our K-NN model were 5 nearest neighbors, kd_tree for the algorithm, and uniform for the weights.

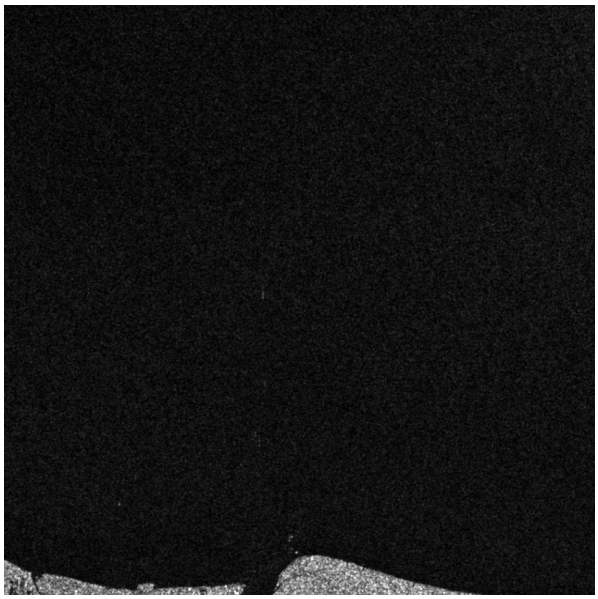

Figure 6: Image with small amount of land labeled as inshore. Image Source: [15]

Previously, when building the inshore-offshore classifier, we used a handcrafted approach utilizing thresholds to preprocess and classify images. Essentially what this entailed, was first binarizing an image where all pixel values below the threshold were set to zero, and values greater than or equal to the threshold value were set to the max value of 255. Then after binarizing the image, the sum of pixel values was taken as a feature and a classification threshold was used where if the sum of pixel values was above the threshold, it would be classified as inshore and below the threshold would be classified as offshore. As it can be seen, this method uses different features compared to the machine learning models where instead of deriving percentiles of pixel values, the sum of pixel values was used as the main feature. Conceptually this made sense where offshore images would essentially be mostly black and have low sum of pixel values, while inshore images would have large pixel values and in practice it worked pretty well with about a 92% accuracy. However, with this method, a few problems arose which will be discussed further in the discussion section.

4.2 Object Detection Models

The first class of object detection models we decided to train was YOLOv7. The model we trained was a YOLOv7-D6, with 154.7 million parameters and it is able to do real time inference at 44 frames per second. This is a relatively light-weight model that trades accuracy for speed. During training, we used the entire data set and train/test split as specified by the author of the dataset. Additionally, for the model we downsized the images to 640 by 640 pixels, used a batch size of 8, and trained for 200 epochs. For the hyper parameters, we used the default ones given in the training script for momentum, learning rate, and optimization function. The YOLOv7-D6 models took and 17 hours to train on a GeForce RTX 2080TI. After training the model, we experimented with a variety of thresholding values and found that 0.25 worked the best.

The next two models we used were trained in Pytorch and we only used images with ships for them due to how the optimizer was set up. This brought the number of images down from the original 9,000 to about 1,900 images and we kept them in the original 800 by 800 pixel size without downsizing. We attempted to solve this issue by introducing image augmentations such as rotating and noise, but were unable to implement such features given our time frame. To solve this issue, we trained our algorithms for more iterations on the training set.

One model we used was RetinaNet [8] and we used it with a ResNet-50-FPN backbone that was pre-trained on the COCO dataset [7]. We also changed the output layer to only two classes, which were ship and background. After a few hyperparameter tests, we found the best ones for training using stochastic gradient descent were batch size of 10, 300 epochs, learning rate of 0.001, and a momentum value of 0.9. It took about 12 hours to train on a GTX 1080TI.

The next model we used was the Faster R-CNN [10] and it also had a ResNet-50-FPN backbone. We replaced the pre-trained head for the classifier with our two output classes of ship and background. For the hyper parameters we found the best values to be batch size of 10, 300 epochs, learning rate of 0.0001 and momentum of 0.7. It took 14 hours to train on a GTX 1080TI.

4.3 Downloading Google Earth Engine Images

The data we intend to deploy our model on is the Sentinel-1 SAR image collection on Google Earth Engine. To access this data, we built a helper function that takes in the coordinates of the desired area as well as the start and end dates that the user wants to pull from. The coordinates for this function can be found on the Google Earth Engine Code editor where you create a rectangular bounding box over the desired area of the map and within the console, it will show the coordinates of the bounding box. The helper function will download the set of images locally as .tif files. The bands we chose when downloading our images was the VV single co-polarization band. With this band, ships provide better reflections of the radar signals compared to some of the other polarization bands such as VH and make it ideal for ship detection [15]. In addition, we have to do some additional cleaning of the image values because as it can be seen in Figure 7, the image appears to have a gray tint to it. This is because the values are usually between 0 and 1 and must stretch the original image by $10\log_{10}(I)$. Fixing this requires clipping images values from -20 to 0 where values less than -20 are set to -20 and values greater than 0 are set to 0 [9]. As it can be seen in Figure 8, it creates a much more contrasted and clearer image. From there, we use another helper function to pad and then split the image into sub-images. The helper function first pads the full images with black pixels to the next biggest multiple of 800×800 pixels so that each image will cleanly be split up into sub images of size 800×800 pixels. The reason why we split the images into sub-images of 800×800 pixels is because our inshore-offshore classifier and object detection models predict on images of that size. We pad the images with the value -20 because after we clipped the values in the original image, -20 represents a black pixel. In addition, we chose to pad our images rather than resize them as to reduce possible distortions when resizing to the next biggest multiple of 800×800 pixels. The helper function finally then splits the image into an array of dimensions $m \times n \times 800 \times 800$, where $m \times n$ are the grid of sub images generated with each sub image being 800×800. We then flatten this array into a $m \cdot n \times 800 \times 800$ so that we don't have to use a nested for loop when iterating over the set of sub-images and reduces the overall run time of the final ship counting script.

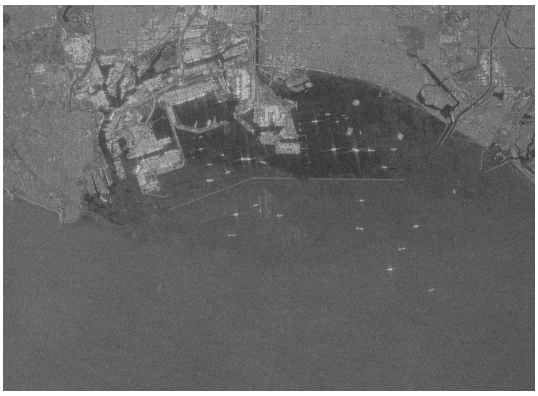

Figure 7: Raw Google Earth Engine Image

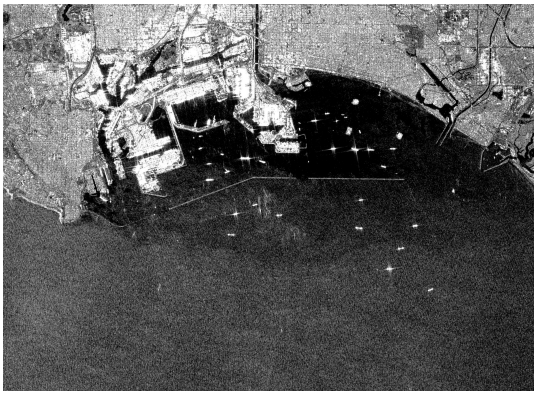

Figure 8: Transformed and Clipped Google Earth Engine Image

5 Results

5.1 Inshore-Offshore Classifier

Table 2 shows a comparison of the different classification methods for the inshore-offshore classifier. As it can be seen, the handcrafted thresholding method is the fastest running time of about 13 seconds, while the slowest being AdaBoost. K-Nearest Neighbors had the best average accuracy of .972 with Random Forest, AdaBoost, and Decision Tree showing very similar average accuracies of 0.967, 0.961, and 0.59 respectively. The worst performing classifier in terms of average accuracy was the Naive Bayes Gaussian classifier with a score of 0.88.

Model	Average Accuracy	Runtime
Handcrafted Thresholding	0.926	13.63
Decision Tree	0.959	104.10
Random Forest	0.967	107.06
Adaboost	0.961	123.36
Gaussian NB	0.88	100.30
K-Nearest Neighbors	0.972	111.45

Table 2: Average Accuracies and Runtimes of Classification Methods

5.2 Object Detection Models

For selecting the best object detection model, we first needed to find the best thresholding value for each model. We ran the model on the test set using 20 different thresholding values from 0 to 1 and picked the one with the highest F1 score. Other metrics we looked at were presion, recall, and mAP.

After running the thresholding experiment, we found the optimal values for each mode, and their metrics on the full test set which can be summarized in Table 3 and Figure 9. The YOLOv7-D6 had a confidence threshold of 0.25, which led it to have precision of 0.93, a recall of 0.5, an mAP of 0.49, and an F1 score of 0.65. It only took 32.44 seconds for the model to run on the full test set of images with ships. The next model was RetinaNet and it had a confidence threshold of 0.8, precision of 0.71, recall of 0.75, mAP of 0.66, an F1 score of 0.73, and a run time of 50.21 seconds on the full test set. Lastly, the Faster R-CNN model had a confidence threshold of 0.7, precision of 0.94, recall of 0.92, mAP of 0.91, F1 score of 0.93 and took 53.01 seconds to run on the test set.

We also wanted to look at metrics grouped by inshore or offshore. There were only 619 offshore images and 117 inshore images in the test set, so some of the metrics might be skewed due to the small sample size. Figure 10 and Table 4 summarize the metrics of the inshore images of the object detection algorithms. We can see that relative to the overall values, these metrics were much worse

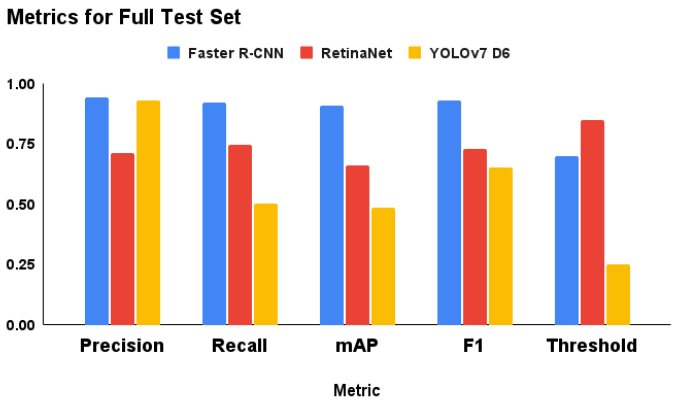

Figure 9: Metrics on entire test set with ships

Model Name	Precision	Recall	mAP	F1	Threshold	Time (seconds)
Faster R-CNN	0.94	0.92	0.91	0.93	0.7	53.01
RetinaNet	0.71	0.75	0.66	0.73	0.8	50.21
YOLOv7 D6	0.93	0.50	0.49	0.65	0.25	32.44

Table 3: Metrics for Full Test Set

for all of the models. Figure 11 and Table 5 summarizes all the metrics for the offshore test images. The offshore metrics are much better when compared to the inshore test.

Model Name	Precision	Recall	mAP	F1
Faster R-CNN	0.89	0.83	0.80	0.86
RetinaNet	0.43	0.50	0.31	0.46
YOLOv7 D6	0.84	0.14	0.13	0.24

Table 4: Metrics for inshore test set with ships

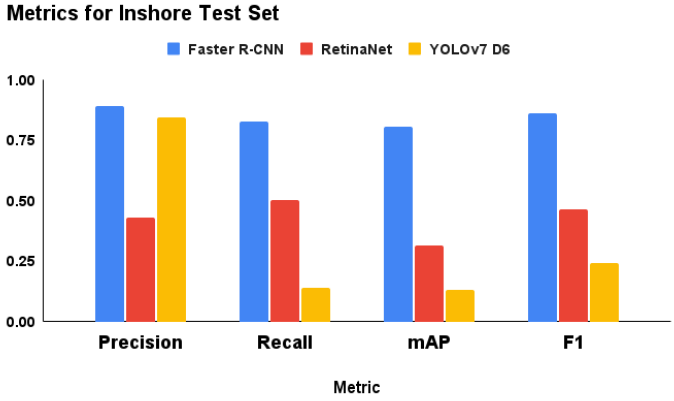

Figure 10: Metrics for inshore test set with ships

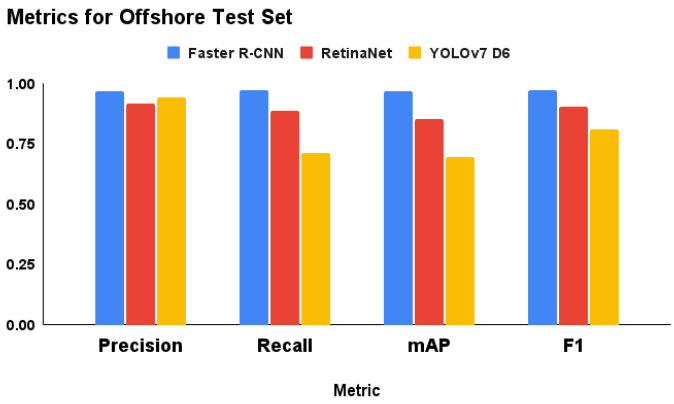

Figure 11: Metrics for offshore test set with ships

Model Name	Precision	Recall	mAP	F1
Faster R-CNN	0.96	0.97	0.97	0.97
RetinaNet	0.92	0.89	0.85	0.90
YOLOv7 D6	0.94	0.71	0.70	0.81

Table 5: Metrics for offshore test set with ships

6 Discussion

6.1 Inshore-Offshore Classifier

As mentioned before, the handcrafted method proved to do well on the training dataset especially in terms of runtime, however, a few problems arose when using this method:

1. The LS-SSDD-v1.0 authors' definition of inshore was an image with any bit of land in it, so images with even just a little bit of land were classified as inshore. This proved to be a problem with using the sum of pixel values as a feature, because the sum of pixel values for these types of inshore images would be low due to the large portion of the image being dark and would therefore be misclassified.

2. Also for some images that have a more gray tint, there would still be a lot of white pixels after binarization and would then make the total sum of pixel values of the image pretty high. See Figure 12 and Figure 13

3. Another problem would be with generalization of our model because these threshold values were based on these images we already had, but the data we would get from Google Earth Engine might not be the same.

Due to these problems, training a machine learning model proved more beneficial because it is more generalizable for a new set of images and is more robust towards some of these edge cases.

When training and testing the different models, many of the models performed quite similarly in terms of both runtime and average accuracy. Table 2 shows that the Decision Tree, Random Forest, and K-Nearest Neighbors classifiers all had quite high accuracies as well as fast runtimes relative to each other. The other models like Adaboost and Naive Bayes Gaussian models either were much slower than its counterparts or had a much lower average accuracy. However, we chose to go with the K-Nearest Neighbors classifier due to the fact that it consistently performs well and the slightly slower runtime can be compromised with better accuracy. In our application, we want to ensure more accurate predictions so that we can correctly feed inshore and offshore images into their respective ship detection models. In cases of misclassification, feeding an offshore image into our inshore classifier may end up slowing down the overall pipeline and feeding an inshore image into the offshore classifier may produce less accurate ship counts. Due to these reasons, the K-Nearest Neighbors classifier best suited our use case.

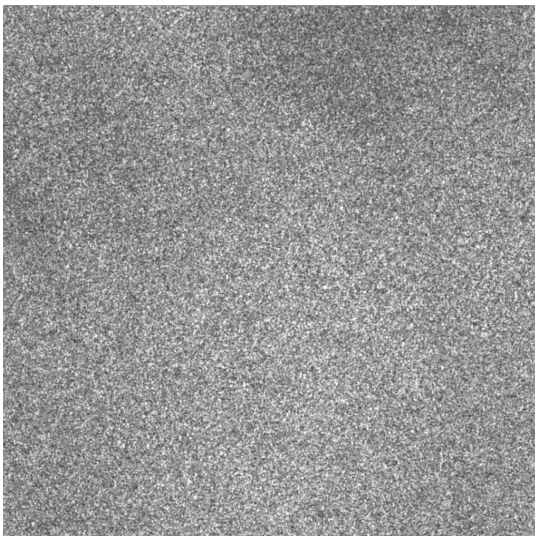

Figure 12: Gray offshore image

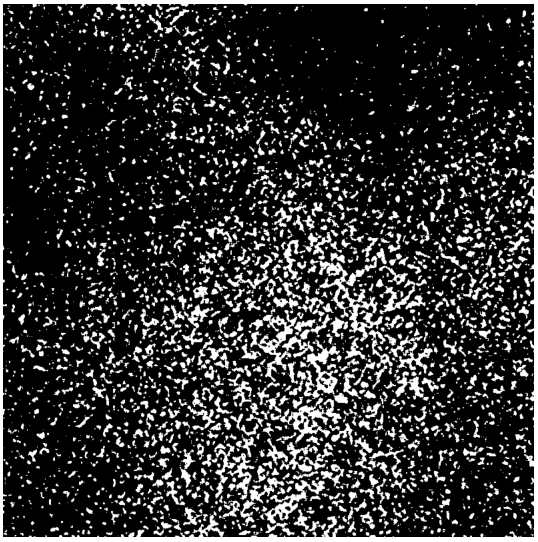

Figure 13: Binarized gray offshore image

6.2 Object Detection Models

From the above metrics, we can see that the Faster R-CNN had the highest accuracy scores, but was slowest on inference. Figure 14 is an inshore image with no ships in it and to the right of it is the Faster R-CNN predictions. We can see that the algorithm has made some false predictions. When we look closer we can see that the areas it predicts are bright spots with darkness around them, similar to a ship on the open sea. Figure 15 contains two images, with the left image being the ground truth and the right image contains the predictions of the model. The majority of the ships are detected correctly, with one bright spot being misclassified. The model might mistake the noise for some ship. Overall, the model does well enough for the given task, but there is always room for improvement.

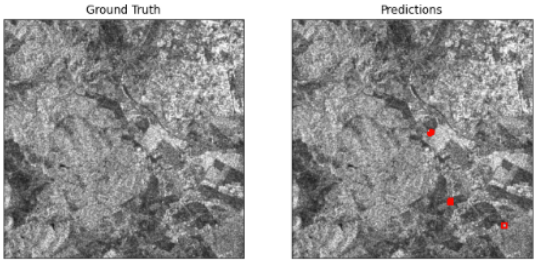

Figure 14: Ground Truth and Faster R-CNN prediction on an inshore image

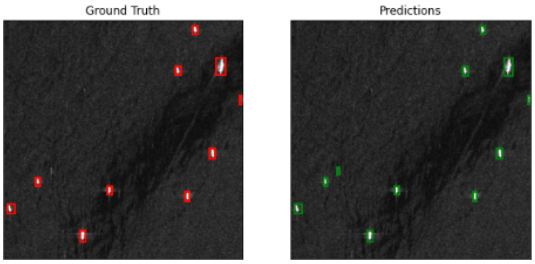

Figure 15: Ground Truth and Faster R-CNN prediction on an offshore image

For the tool, a separate object detection algorithm for the inshore and offshore images is needed. Faster R-CNN will do best for the inshore images, but RetinaNet is almost as good as Faster R-CNN on the offshore images. The F1 score of the two images only differs by 0.07, so there will only be a negligible tradeoff in accuracy. The speed up is also relatively small, of only about 3 seconds between the two models. However, we believe the small sacrifice in accuracy will be worth the potential speed up of the tool, because the majority of the images are going to be offshore ones.

7 Applications

7.1 Tool Overview

All of the pieces for the tool are ready and it is time to combine them. The process can be visualized in Figure 16. The first step is getting all of the raw SAR images using the Google Earth Engine API. Then, each image is preprocessed by clipping all the pixel values between -20 and 0 to remove the gray tint, and zero padded to be sharded into 800 by 800 pixel sub images. After, it is passed into a inshore/offshore classifier and that determines which object detection model counts the number of ships in it. The details of running the tool locally can be found on our Github [3].

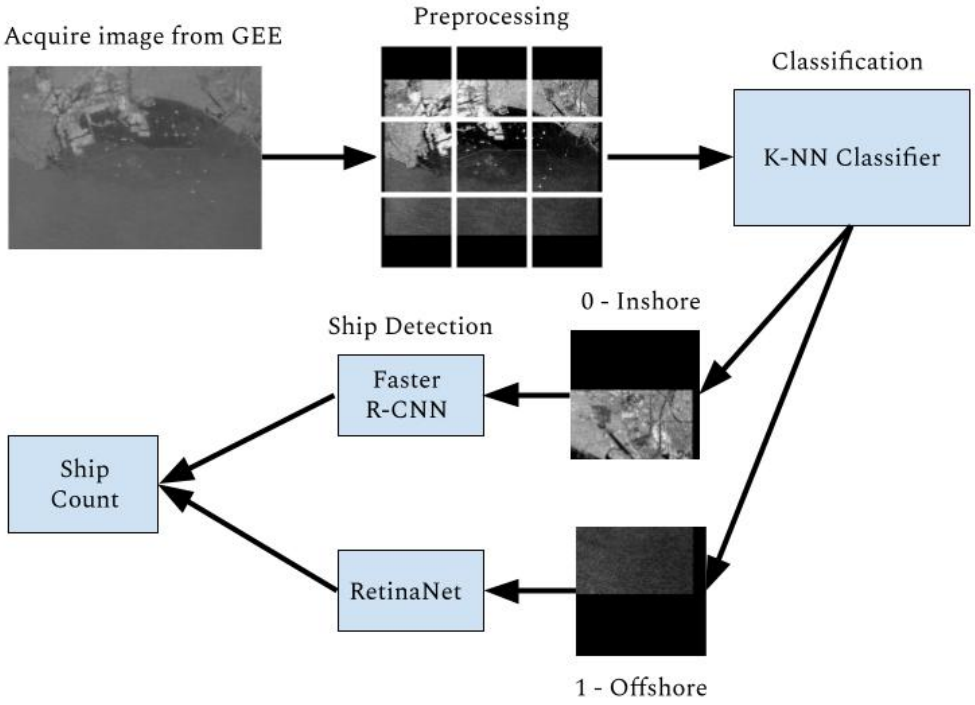

Figure 16: Workflow of our Tool

We tried two different versions of the tool, one as described as above and one where we only used Faster R-CNN for ship counting. To test, we used the full sized test images in the test set and calculated the Root Mean Squared Error of the ship counts as our metric. The results are shown in Figure 17. The regular tool slightly does better on the RMSE metric and a lower runtime than the Faster R-CNN. We hope the difference in speed would translate to scale.

7.2 Ever Given Stuck in the Suez Canal (March 23, 2021 - March 29, 2021)

One application that our tool can be used on is to look at the effect of the Suez Canal blockage in 2021. On March 23, 2021, a massive container ship named the Ever Given got lodged diagonally within the Suez Canal. This blockage cost global trade an estimated $6 billion to $10 billion a day. Then on March 29 that same year, the Ever Given was finally freed [11]. By using our tool, we can view these effects by looking at differences in the number of ships before, during, and after the blockade. As we can see in Figure 18, a few days before the blockade, there are roughly 44 ships going into the canal. But during the blockade (See Figure 19), we can see the number of ships outside the canal jumps up to 95 ships and then after the ship is freed, the number goes back down to around 54 ships (See Figure 20). We can see how drastic the blockage caused, not only through the numbers, but also through our satellite images. One thing to note though is that our object detection model doesn't

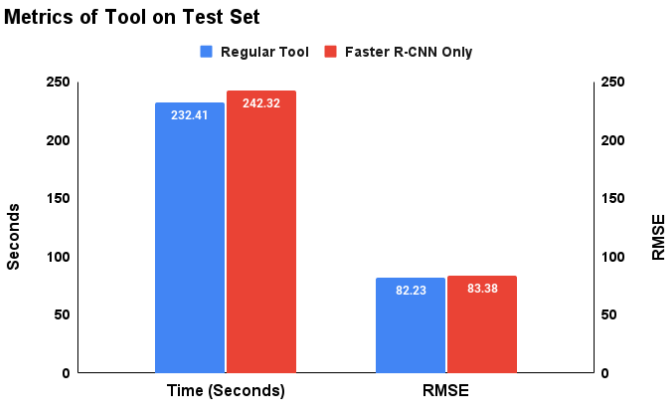

Figure 17: Metrics of the tool on the test set

seem to identify the ships within the canal, which could possibly be due to the fact that it thinks that the canal and ships within the canal are land due to the fact that the canal itself is pretty small and is surrounded by land on all sides.

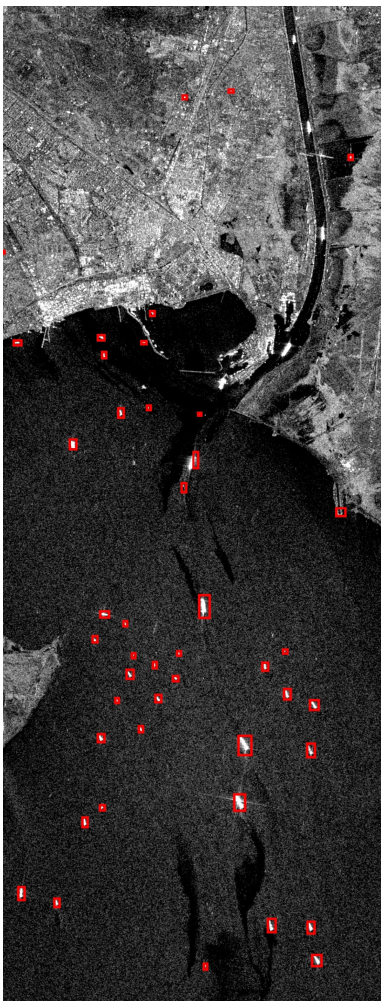

Figure 18: Before Ever Given Stuck (3/20/2021): 44 ships

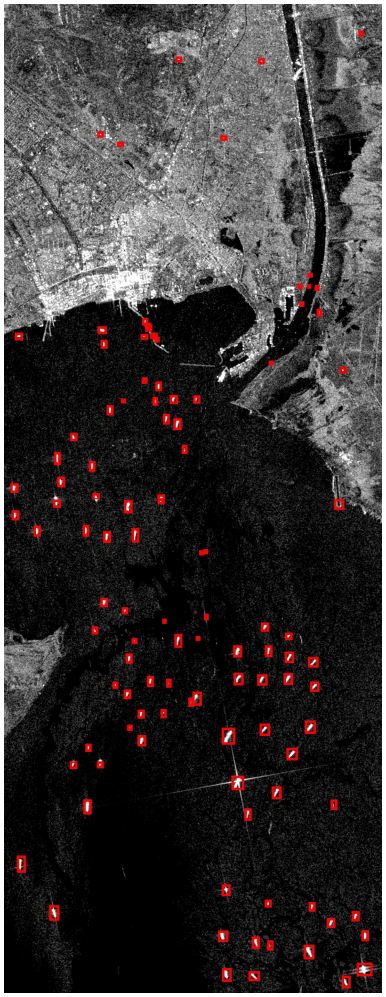

Figure 19: While Ever Given Stuck (3/25/2021): 95 ships

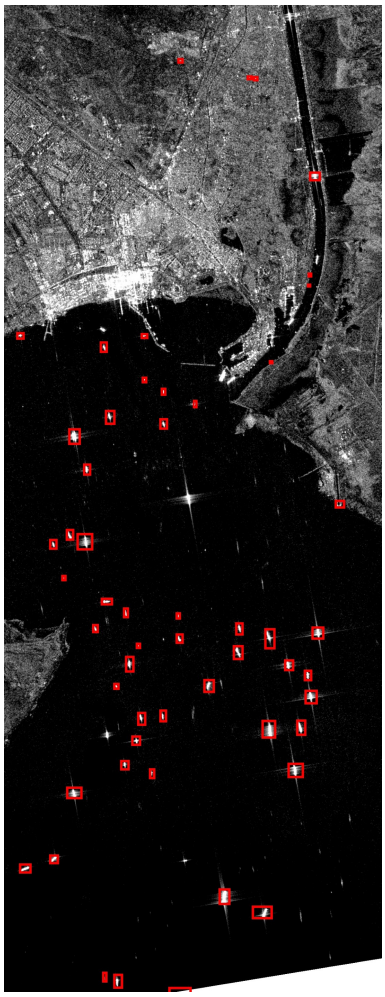

Figure 20: After Ever Given Freed (4/11/2021): 54 ships

7.3 Activity in the Port of Los Angeles during COVID Lockdown (February 2020 - April 2020)

Another use case could be analyzing activity within the Port of Los Angeles before, during, and after the COVID-19 lockdown. The Port of Los Angeles is one of the most active ports in America and is one of the most important ports that connects America to International trade [1]. During the year 2020, the world was hit with a worldwide pandemic and in March of 2020, America went on a lockdown [2]. Due to this worldwide event, we thought it would also be interesting to see how activity within the Port of Los Angeles changed throughout the first few months of lockdown. When at the count of ships in Figure 22 compared to Figure 21 and Figure 23, the number of ships as the lockdown was happening dropped by almost 10-20 compared to both a month before and month after.

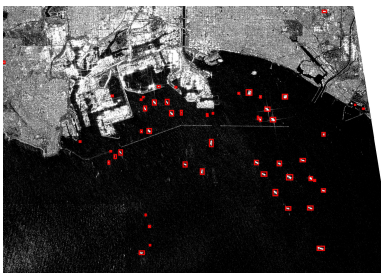

Figure 21: Before COVID Lockdown (2/4/2020): 52 ships

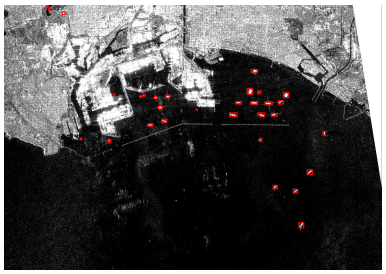

Figure 22: During COVID Lockdown (3/17/2020): 33 ships

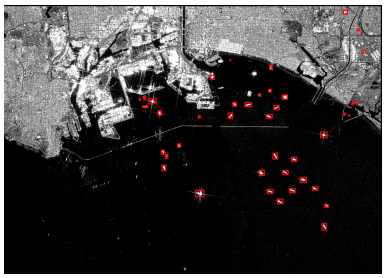

Figure 23: After COVID Lockdown (4/11/2020): 43 ships

8 Conclusion

Our goal is for the ship counting tool we have created to be used for other projects such as political policy monitoring, or economic modeling. The modularity of it allows for different parts of the tool to be easily replaced or reconfigured. For example, if a new object detection model is created it can be easily trained and swapped with the current one in the tool. Another example is if a new source of satellite imagery data becomes open source or purchased it can be inputted into the tool instead of Google Earth Engine. Since our framework is novel, we hope that others continue to build and improve upon this tool or apply it for a beneficial purpose.

References

[1] Port of los angeles. URL https://californiaports.org/ports/port-of-los-angeles/.

[2] Cdc museum covid-19 timeline, Aug 2022. URL https://www.cdc.gov/museum/timeline/covid19.html.

[3] Sean Ng Alexander Makhratchev. SAR-satelite-image-ship-detection. https://github.com/alexmak001/SAR-satelite-image-ship-detection.

[4] Yang-Lang Chang, Amare Anagaw, Lena Chang, Yi Chun Wang, Chih-Yu Hsiao, and Wei-Hong Lee. Ship detection based on YOLOv2 for SAR imagery. *Remote Sensing*, 11(7):786, 2019.

[5] Urška Kanjir, Harm Greidanus, and Krištof Oštir. Vessel detection and classification from spaceborne optical images: A literature survey. *Remote sensing of environment*, 207:1–26, 2018.

[6] Jianwei Li, Changwen Qu, and Jiaqi Shao. Ship detection in SAR images based on an improved faster R-CNN. In *2017 SAR in Big Data Era: Models, Methods and Applications (BIGSARDATA)*, pages 1–6. IEEE, 2017.

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[8] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[9] mortcanty. Detecting Changes in Sentinel-1 Imagery (Part 1). https://developers.google.com/earth-engine/tutorials/community/detecting-changes-in-sentinel-1-imagery-pt-1.

[10] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[11] Michael Safi, Helena Smith, and Martin Farrer. Suez canal: Ever given container ship freed after a week, Mar 2021. URL https://www.theguardian.com/world/2021/mar/29/suez-canal-attempt-re-float-ever-given-delay-salvage-tugboats.

[12] Shunjun Wei, Xiangfeng Zeng, Qizhe Qu, Mou Wang, Hao Su, and Jun Shi. HRSID: A high-resolution SAR images dataset for ship detection and instance segmentation. *Ieee Access*, 8: 120234–120254, 2020.

[13] SUN Xian, WANG Zhirui, SUN Yuanrui, DIAO Wenhui, ZHANG Yue, and FU Kun. AIR-SARShip-1.0: High-resolution SAR ship detection dataset. , 8(6):852–863, 2019.

[14] Rong Yang, Gui Wang, Zhenru Pan, Hongliang Lu, Heng Zhang, and Xiaoxue Jia. A novel false alarm suppression method for CNN-based SAR ship detector. *IEEE Geoscience and Remote Sensing Letters*, 18(8):1401–1405, 2020.

[15] Tianwen Zhang, Xiaoling Zhang, Xiao Ke, Xu Zhan, Jun Shi, Shunjun Wei, Dece Pan, Jianwei Li, Hao Su, Yue Zhou, et al. LS-SSDD-v1. 0: A deep learning dataset dedicated to small ship detection from large-scale Sentinel-1 SAR images. *Remote Sensing*, 12(18):2997, 2020.