Ship Detection and Segmentation using Mask-RCNN

Introduction

Shipping traffic is growing fast. More ships increase the chances of infractions at sea like environmentally devastating ship accidents, piracy, illegal fishing, drug trafficking, and illegal cargo movement. This has compelled many organizations, from environmental protection agencies to insurance companies and national government authorities, to have a closer watch over the open seas. A lot of work has been done over the last 10 years to automatically extract objects from satellite images with significant results but no effective operational effects.

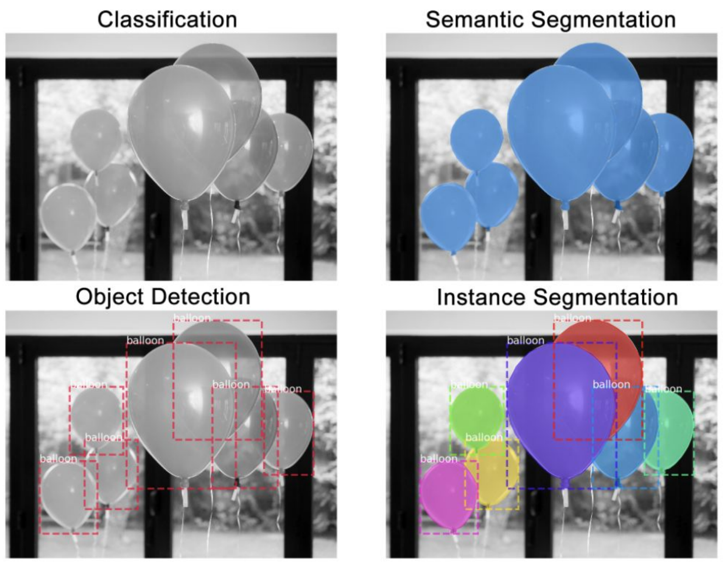
Problem Statement and Dataset

I used "**The Kaggle's Airbus Detection Challenge Dataset**" as the dataset. In this project, the objective is to automatically identify whether a remotely sensed target is a ship or not. To do so, I was required to locate ships in images, and put an aligned bounding box segment around the ships located. There are many images that do not contain ships, and those that contain multiple ships. Ships within and across images differ in size (sometimes significantly) and are located in open sea, at docks, marinas, etc.

Environment and Tools Used

Jupyter Notebook with scikit-learn, scikit-image, numpy, pandas, matplotlib.

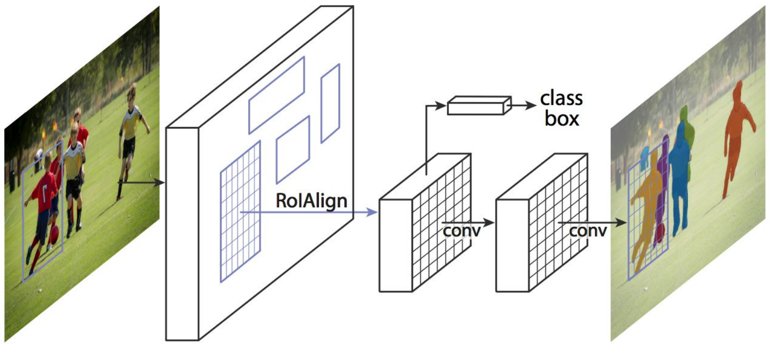
Segmentation and Classification - An Overview



There are four types of segmentation and classification as seen above. For this project, I have tried to incorporate "**Instance Segmentation**." However, the other three methods can be used for other types of projects based on their corresponding requirements.

Mask-RCNN Overview

Mask R-CNN is an instance segmentation technique which locates each pixel of every object in the image instead of the bounding boxes. It has two stages: region proposals and then classifying the proposals and generating bounding boxes and masks. It does so by using an additional fully convolutional network on top of a CNN based feature map with input as feature map and gives a matrix with 1 on all locations where the pixel belongs to the object and 0 elsewhere as the output.

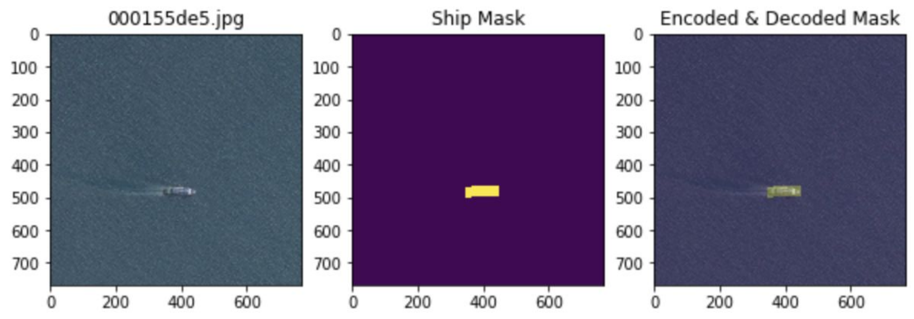


Step 1 : Initial Setup

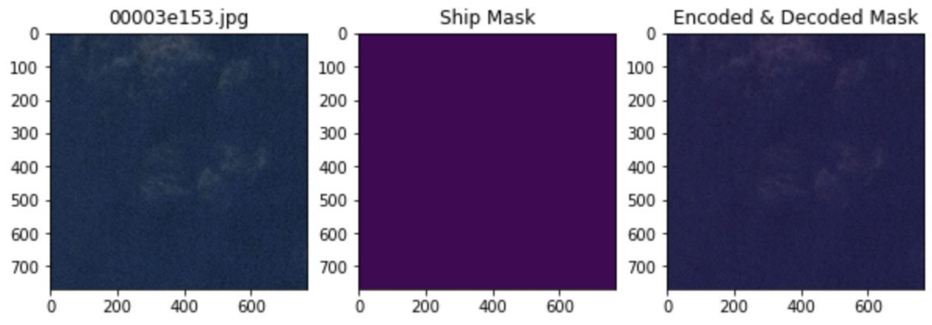
I installed the dataset, required dependencies and directory files, all using a Python script.

Step 2 : Playing with Masks

I continued with defining functions for showing some sample images with a ship mask and an encoded and decoded mask. Here masks are used as ground truth images.



```
000155de5.jpg  Check Decoding->Encoding RLE_0: 1 -> RLE_1: 1
```



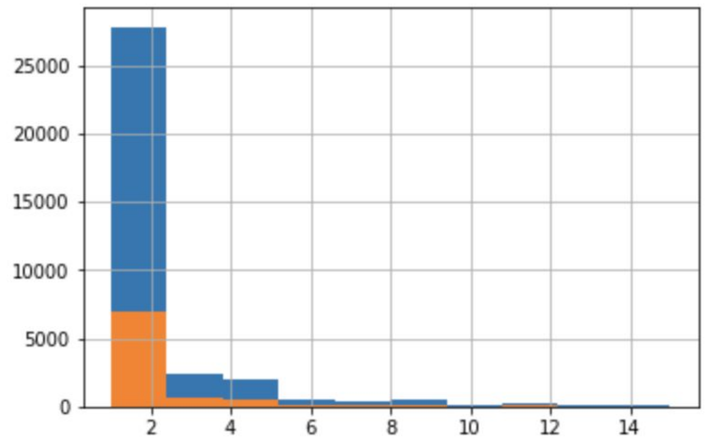
```
00003e153.jpg  Check Decoding->Encoding RLE_0: 1 -> RLE_1: 0
It could be different when there is no mask.
```

So only if the ship is present in a picture, we get to see it after encoding and decoding the ship picture as seen in the picture above.

Step 3 : Splitting my dataset and setting up the Mask-RCNN model

I divided my dataset into training and validation sets using a python script to count the number of ships in them and divide in a random manner.

```
34044 training masks
8512 validation masks
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c8345cba8>
```

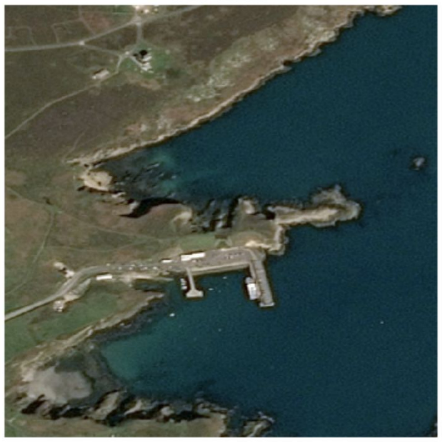
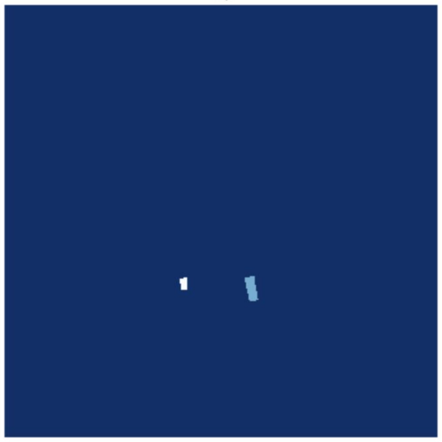
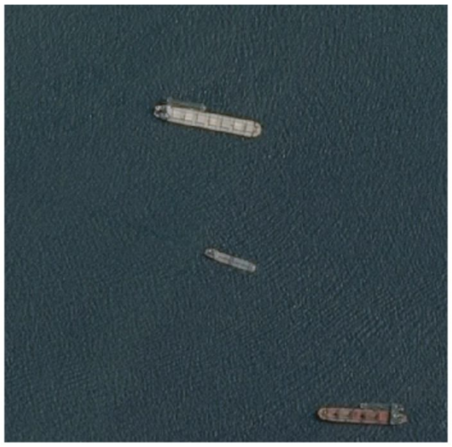
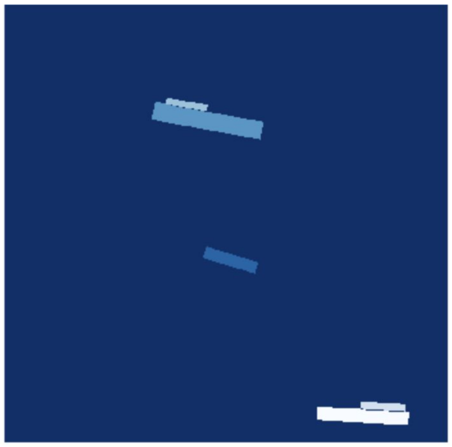
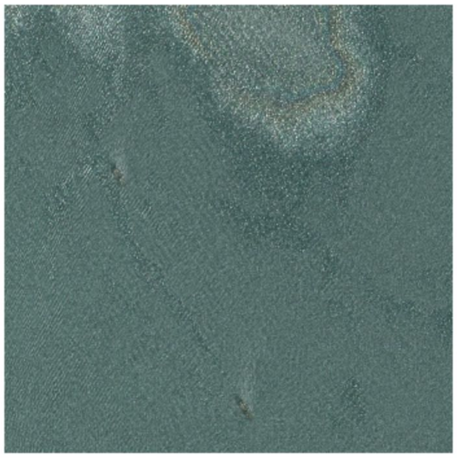
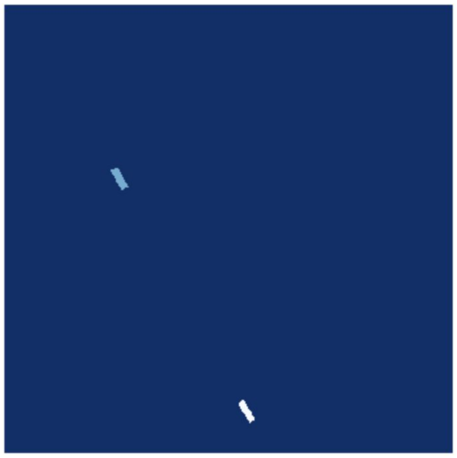


Blue denotes the training set and orange denotes the validation set. I generated 34,044 training images and 8,512 validation images. The X-axis denotes the number of ship masks in each image.

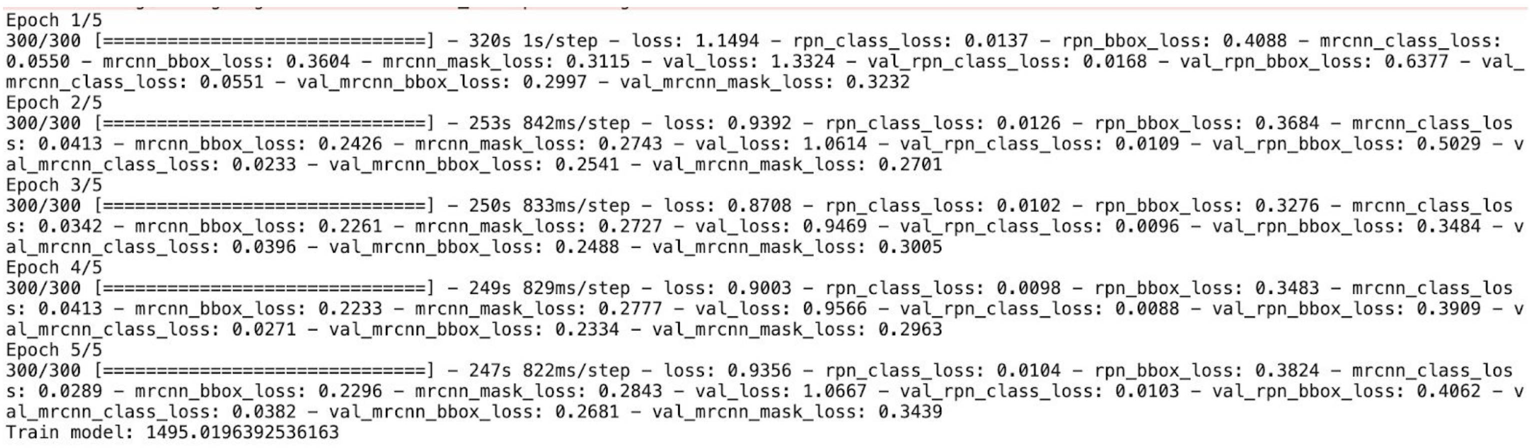
Next, I forked the original MASK-RCNN repo from here - https://github.com/matterport/Mask_RCNN

Step 4 : Applying Masks to all images

A few outputs I got are -

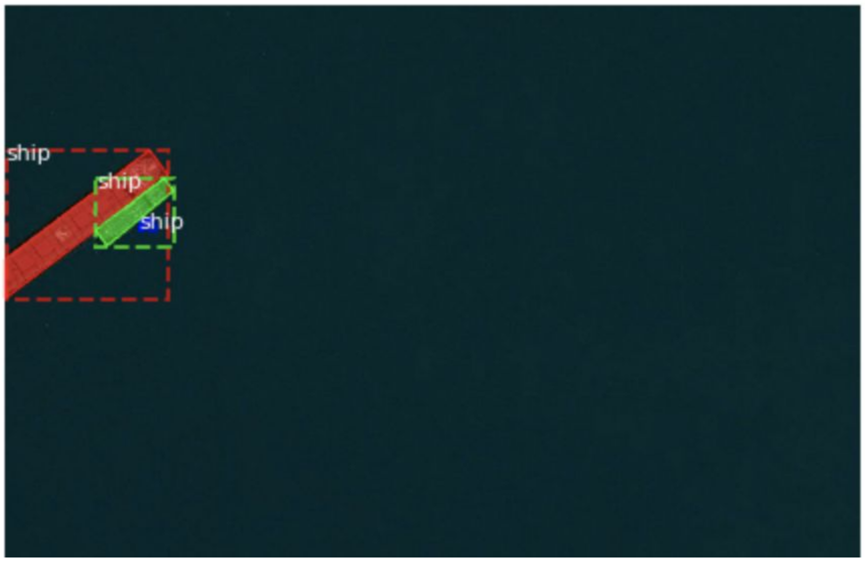
Step 4 : Downloading the pre-trained COCO weights and Training my Mask-RCNN model using them



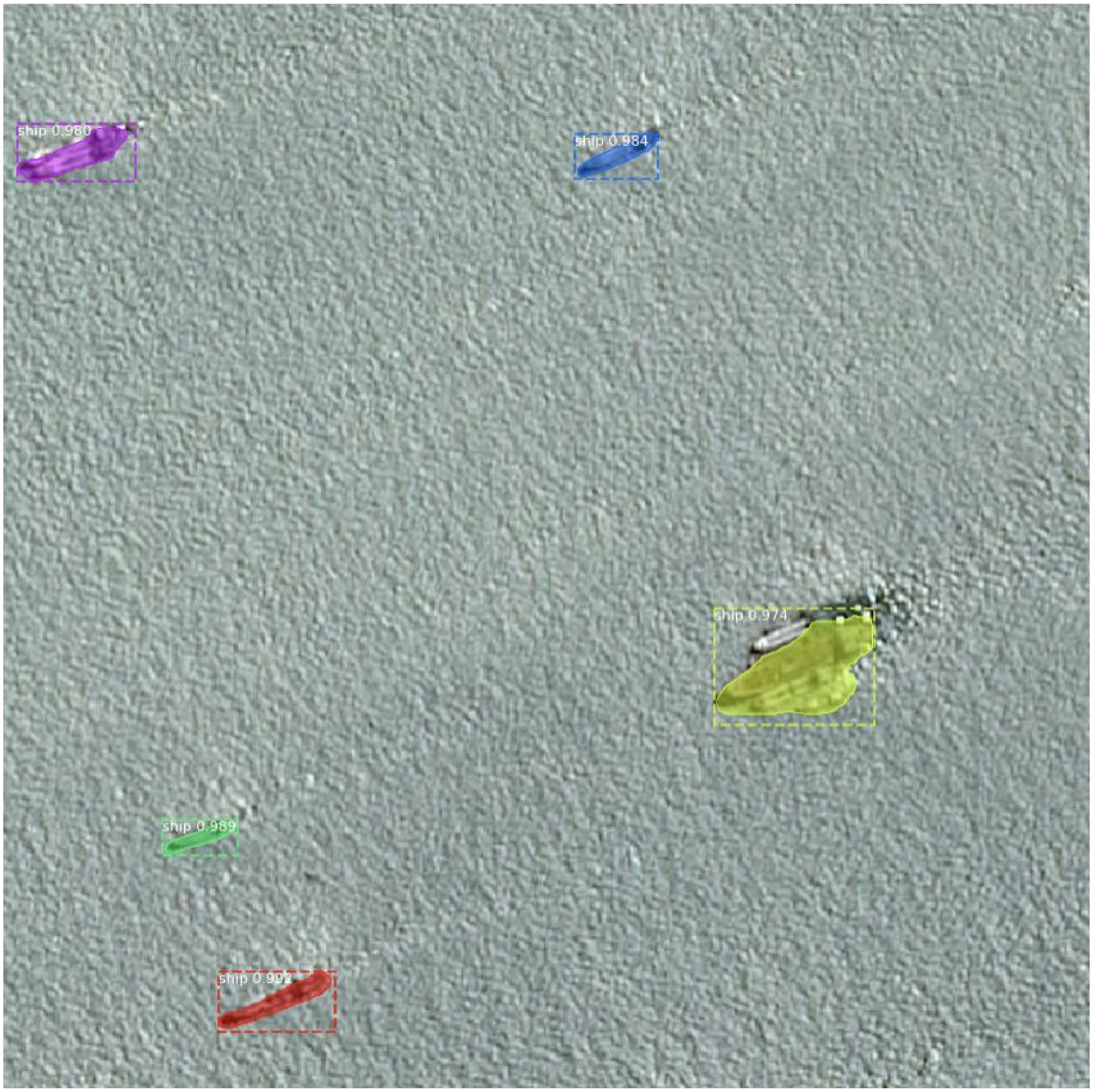
```
Epoch 1/5
300/300 [==============================] - 320s 1s/step - loss: 1.1494 - rpn_class_loss: 0.0137 - rpn_bbox_loss: 0.4088 - mrcnn_class_loss:
0.0550 - mrcnn_bbox_loss: 0.3604 - mrcnn_mask_loss: 0.3115 - val_loss: 1.3324 - val_rpn_class_loss: 0.0168 - val_rpn_bbox_loss: 0.6377 - val_
mrcnn_class_loss: 0.0551 - val_mrcnn_bbox_loss: 0.2997 - val_mrcnn_mask_loss: 0.3232
Epoch 2/5
300/300 [==============================] - 253s 842ms/step - loss: 0.9392 - rpn_class_loss: 0.0126 - rpn_bbox_loss: 0.3684 - mrcnn_class_los
s: 0.0413 - mrcnn_bbox_loss: 0.2426 - mrcnn_mask_loss: 0.2743 - val_loss: 1.0614 - val_rpn_class_loss: 0.0109 - val_rpn_bbox_loss: 0.5029 - v
al_mrcnn_class_loss: 0.0233 - val_mrcnn_bbox_loss: 0.2541 - val_mrcnn_mask_loss: 0.2701
Epoch 3/5
300/300 [==============================] - 250s 833ms/step - loss: 0.8708 - rpn_class_loss: 0.0102 - rpn_bbox_loss: 0.3276 - mrcnn_class_los
s: 0.0342 - mrcnn_bbox_loss: 0.2261 - mrcnn_mask_loss: 0.2727 - val_loss: 0.9469 - val_rpn_class_loss: 0.0096 - val_rpn_bbox_loss: 0.3484 - v
al_mrcnn_class_loss: 0.0396 - val_mrcnn_bbox_loss: 0.2488 - val_mrcnn_mask_loss: 0.3005
Epoch 4/5
300/300 [==============================] - 249s 829ms/step - loss: 0.9003 - rpn_class_loss: 0.0098 - rpn_bbox_loss: 0.3483 - mrcnn_class_los
s: 0.0413 - mrcnn_bbox_loss: 0.2233 - mrcnn_mask_loss: 0.2777 - val_loss: 0.9566 - val_rpn_class_loss: 0.0088 - val_rpn_bbox_loss: 0.3909 - v
al_mrcnn_class_loss: 0.0271 - val_mrcnn_bbox_loss: 0.2334 - val_mrcnn_mask_loss: 0.2963
Epoch 5/5
300/300 [==============================] - 247s 822ms/step - loss: 0.9356 - rpn_class_loss: 0.0104 - rpn_bbox_loss: 0.3824 - mrcnn_class_los
s: 0.0289 - mrcnn_bbox_loss: 0.2296 - mrcnn_mask_loss: 0.2843 - val_loss: 1.0667 - val_rpn_class_loss: 0.0103 - val_rpn_bbox_loss: 0.4062 - v
al_mrcnn_class_loss: 0.0382 - val_mrcnn_bbox_loss: 0.2681 - val_mrcnn_mask_loss: 0.3439
Train model: 1495.0196392536163
```

Step 5 : Output

```
Re-starting from epoch 3
original_image           shape: (768, 768, 3)        min:    0.00000  max:  250.00000  uint8
image_meta               shape: (14,)                min:    0.00000  max: 7698.00000  int64
gt_class_id              shape: (3,)                 min:    1.00000  max:    1.00000  int32
gt_bbox                  shape: (3, 4)               min:    0.00000  max:  263.00000  int32
gt_mask                  shape: (768, 768, 3)        min:    0.00000  max:    1.00000  bool
```



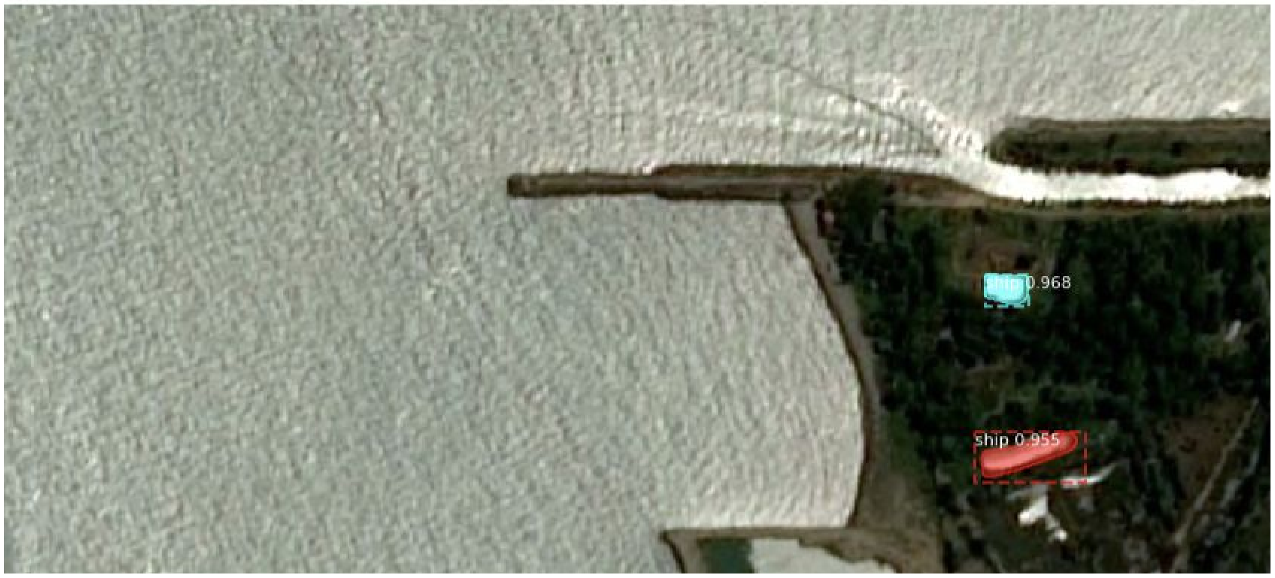
```
Processing 1 images
image                    shape: (768, 768, 3)        min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 768, 768, 3)     min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)              min:    0.00000  max:  768.00000  int64
anchors                  shape: (1, 147312, 4)       min:   -0.47202  max:    1.38858  float32
```



```
Processing 1 images
image                    shape: (768, 768, 3)        min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 768, 768, 3)     min: -107.70000  max:  147.10000  float64
image_metas              shape: (1, 14)              min:    0.00000  max:  768.00000  int64
anchors                  shape: (1, 147312, 4)       min:   -0.47202  max:    1.38858  float32

*** No instances to display ***
```

```
Processing 1 images
image                    shape: (768, 768, 3)        min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 768, 768, 3)     min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)              min:    0.00000  max:  768.00000  int64
anchors                  shape: (1, 147312, 4)       min:   -0.47202  max:    1.38858  float32
```



Conclusion and Issues in the Model

Even though I trained it for 5 epochs, a few problems persist. In a few images, the land in and around the ships has also been getting detected, as seen in the below picture. I need to figure out how to solve this problem.