Using reinforcement learning to detect ships in satellite imagery

Initial thoughts:

I want to initially train a model (undecided on which type of model) on detecting whether there are ships or no ships in a provided satellite image. I then want to visualise where in the image the model looks to come to the conclusion that there are/aren't ships in the given image (this can be done using LIME, Saliency maps etc). I want to use reinforcement learning to narrow down where in the image the model looks to the ship area. One way of doing this could be creating a ground reality bounding box from pretrained models such as YOLO, Faster R-CNN etc and creating bounding box around the areas detected by LIME and essentially using a reward policy to maximise overlap between the LIME bounding box and ground reality bounding box using a loss metric called Intersection over Union.

Initial steps:

1) Need dataset where planes are in bounding boxes

OR code to automatically put planes into bounding box using OpenCV

OR using pretrained models such as YOLO, SSD or Faster R-CNN to detect and draw bounding box (this does mean that the ground reality bounding box generator will have to be finetuned to just draw around ships and ignore buildings, ports etc.)

2) Train model normally on usual dataset - to be used as reference

3) use LIME, Saliency Maps and Grad-CAM to see which part of image is used by model and put bounding box around it [this might need reconsideration]

4) Use Reinforcement learning to direct the model to look at the correct part of the image i.e. change models 'bounding box' until it matches the ground reality bounding box:

a) define the RL environment - include image, bounding box and reward mechanism

b) define RL agent - agent will focus on the region within bounding box by receiving positive

rewards when it focuses on correct region and negative rewards otherwise

c)Use algo such as Q-learning, DQN or PPO to train the model

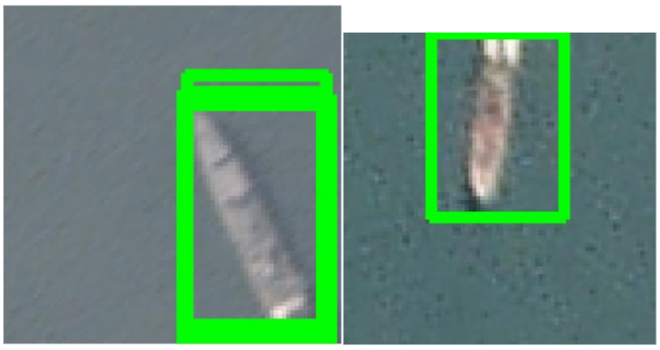
Dataset:

Images sourced from https://www.kaggle.com/datasets/rhammell/ships-in-satellite-imagery however it has been changed so that:

- Every image has ships in it
- Images have only ships and nothing else

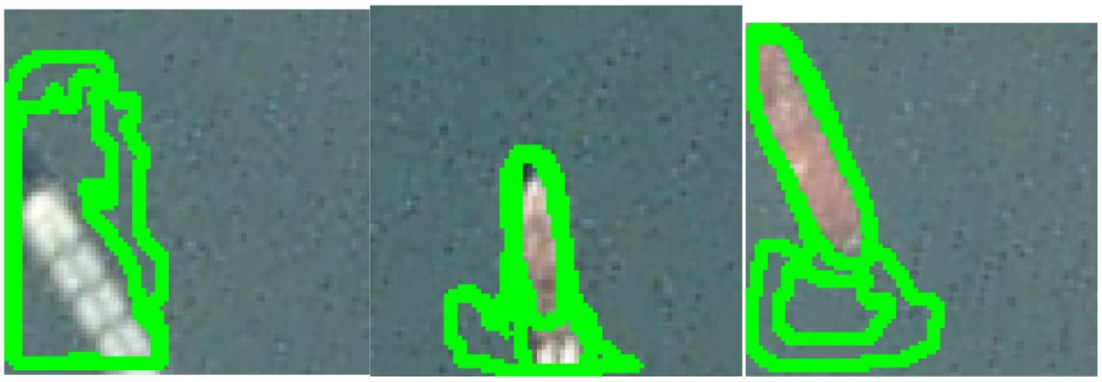
We initially want a successful bounding box around just ships before we train the model to ignore other objects such as bridges

Faster RCNN



- Quite a lot of the training set are square boxes containing only ships, so a bounding box includes the entire image which is not helpful
- Ships are at an angle, so the bounding boxes contain a lot of empty space – need bounding areas which outline exact shape of the ship

Mask RCNN: drawing_bounding_boxes.py



- Does a much better job of outlining ships but also outlines some unnecessary spaces
- Need to finetune this so that it outlines only the ships
- Need to figure out how to make sure the model only highlights ships

Change in problem statement:

Have made dataset to finetune the boundary box generator but trying to finetune it might take me down a different path so I'm going to use the Mask RCNN as is (i.e. treat the images above as ground reality even though they're not great) and if the reinforcement learning is a success, I can attempt to finetune Mask R-CNN.

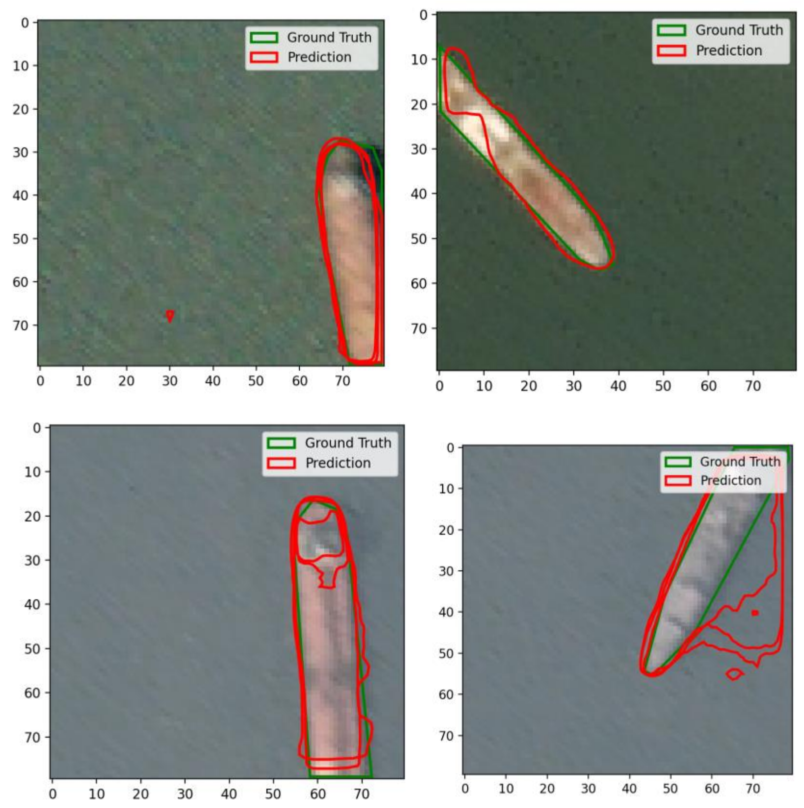
Change in problem statement again:

If the entire task is to display reinforcement learning then instead of using RL to train a plane or no plane detection model, I can just finetune the bounding box generator Mask R-CNN using reinforcement learning.

Steps:

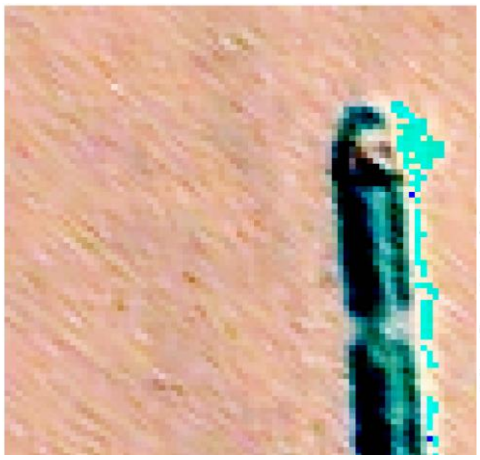
- Use mask R-CNN to initially generate bounding boxes.
- Will use the dataset of perfect bounding boxes as a basis for training the model.
- Reward policy will give rewards on basis of how well the bounding box matches the manually outlined data (ground reality).

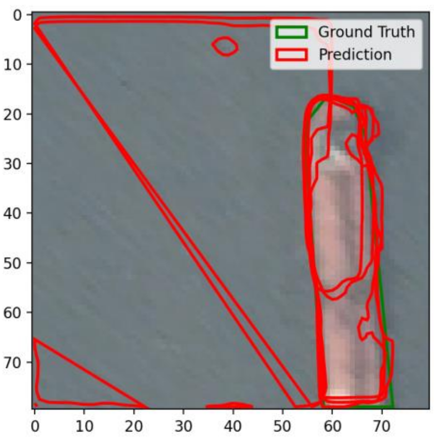
Visualisation of Mask R-CNN and ground reality:

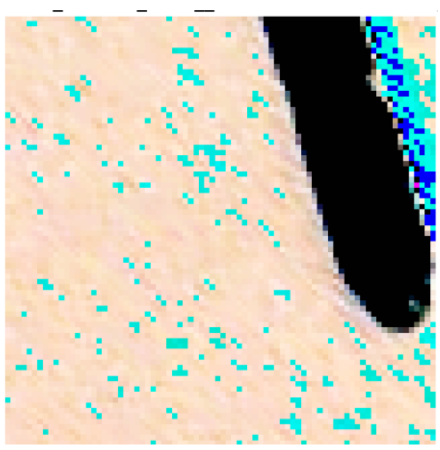


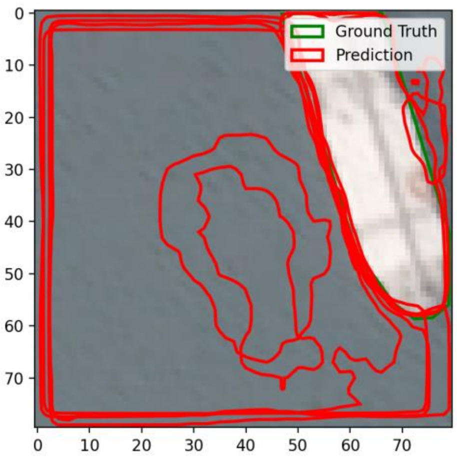
This is prior to any reinforcement learning, just to display performance of an untrained Mask R-CNN model against manually created ground reality. There are issues of the model not predicting polygons for some images which is causing issues for the reinforcement learning. Will attempt to increase contrast in images and lower threshold for confidence in model – I want to have predictions, no matter how bad they are displayed alongside ground reality so I we can see progress once reinforcement learning is applied.

Managed to get a prediction on every single image: display_multiple_polygon_predictions_as_is.py



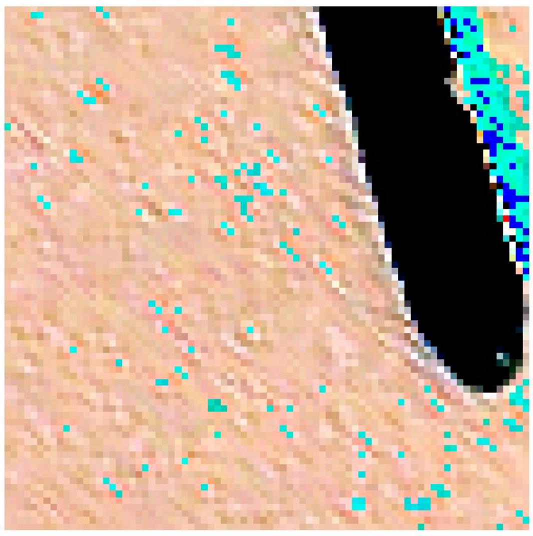


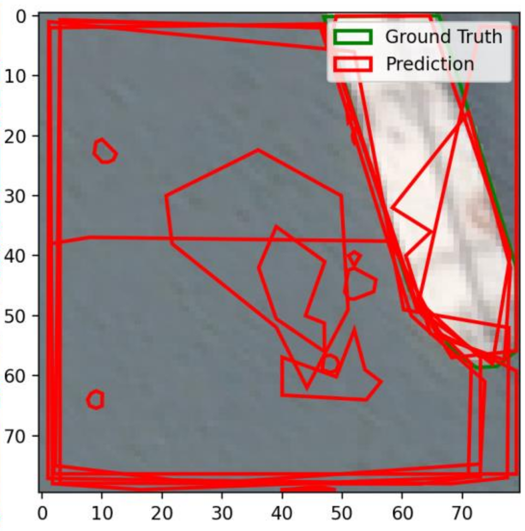


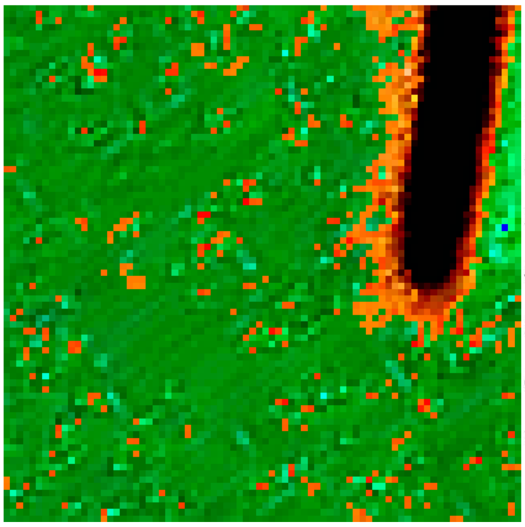


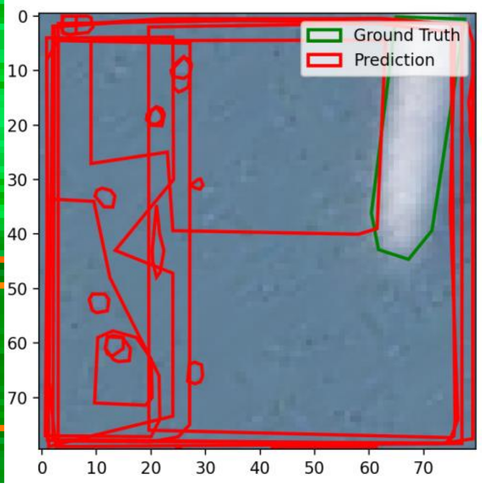
Since the RL code will require specific number of vertices for predicted polygon (so that it can be trained by moving each vertex until overlap between polygon being changed (the predicted polygon) and the ground reality polygon is above a threshold), I will attempt to make each predicted polygon only have 8 vertices:

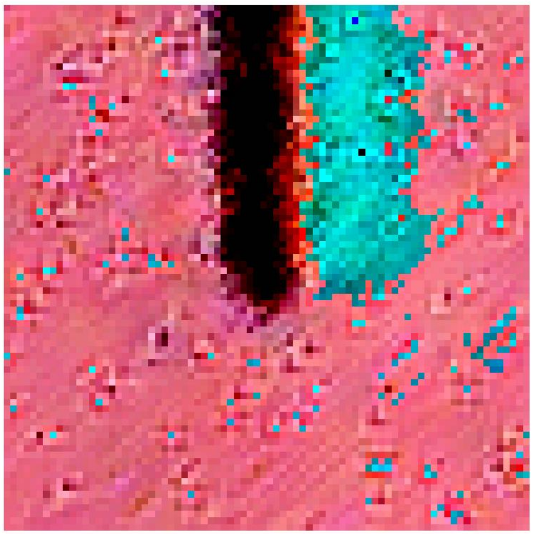
multiple_verticed_polygons_displayed.py

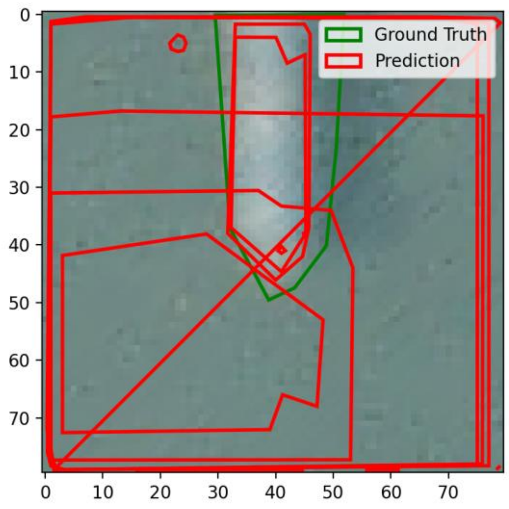




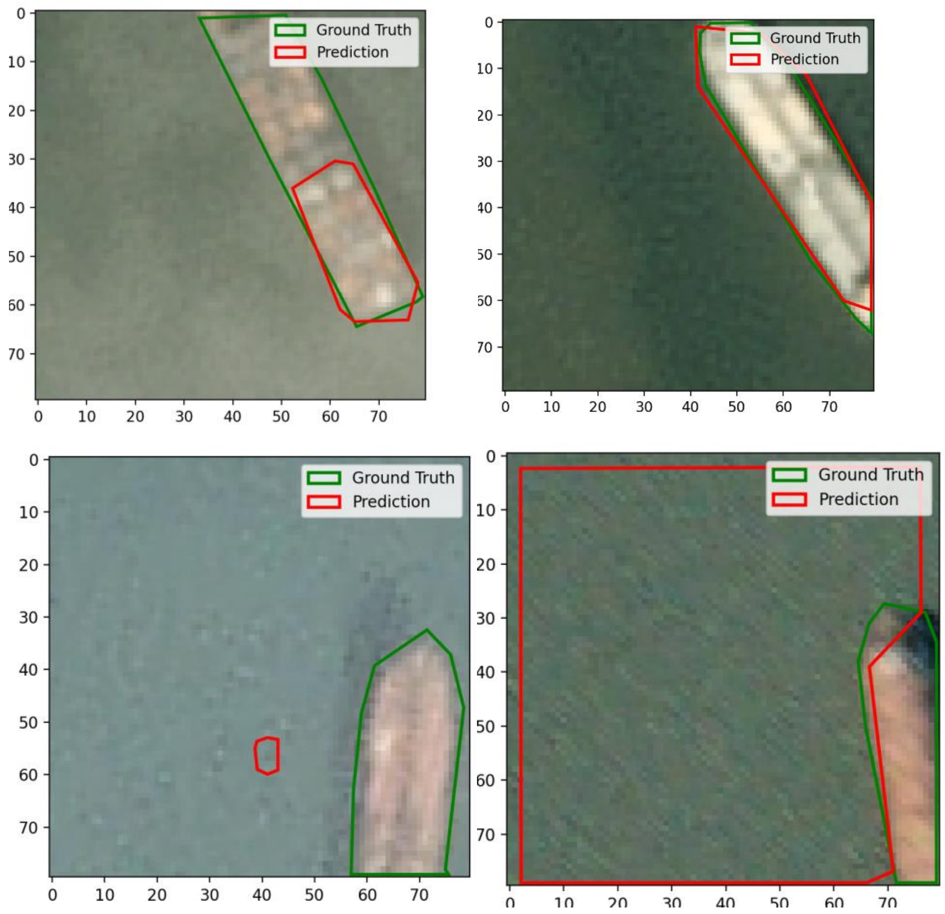








This is giving better results than before. In most cases, out of the many 8 vertices polygons predicted, at least 1 polygon was on the actual ship itself. Next steps will be to figure out a way to narrow down to 1 polygon: both_polygons_displayed.py



I could have technically picked the singular prediction that best matched the ground reality but since we are already using ground reality in the reinforcement learning section, I wanted to keep that separate. Since the Mask R-CNN model gives a confidence rating for each predicted polygon, going with polygons of the highest rating will give us a fairly good polygon for the most part – we don't need it to be perfect, we just need a prediction that comes close to the ship so changing it using reinforcement learning will take fewer computational resources and lower training time for us to see noticeable improvements. The images shown above now need to be implemented in the reinforcement learning framework. I previously started off with the RL code and I found that polygons weren't being generated in that example. Now that we have separate scripts that successfully display both the best predicted polygon (to be changed by the RL code) and the ground reality, the final step will be getting the RL code to recognise the best predicted polygon and change its vertices accordingly during training. Once this is successful, future steps are to keep increasing dataset with ground reality, train and show difference between different trained models.

File which allows predicted and initial polygons to be imported into the reinforcement learning script: gen_predicted_grndreality_polygons.py

Initially the reward policy was to maximise IoU overlap between the predicted polygon and the ground reality polygon, but I realised that this is just incentive for the predicted polygon to get bigger and bigger, so it encompasses the ground reality polygon without specifically narrowing down on the ground reality polygon. For this reason, in addition to the IoU overlap positive reward, the reward policy also includes negative rewards for non-overlapping areas covered by the predicted policy. This will encourage the model to cover all the ground reality polygon whilst also reducing its shape purely just to match the ground reality.

The script for reinforcement learning: environment_and_run.py

I am currently working on a separate document which explains reinforcement learning from ground up and how RL has been applied in this context. For now, I have given some points below explaining issues I ran into so far.

Progress and issues with RL script:

- Ground truths were not recognised as valid – fix them using a zero-width buffer around polygon to clean up invalid geometries
- Also changed RL script to make sure that when vertices are being moved, the sides don't intersect themselves by using negative reward for every time intersection occurs.
- Further cleaning of dataset – removed pictures for which the predictions weren't possible – bit redundant if I just opt for default polygon.
- Instead of trying to recreate the predicted polygon in the RL script, I instead imported them
- For the longest time I was trying to get an initial prediction for every single image which worked in a standalone script but did not work in the RL script – instead just having a default shape to start off with (slight change in problem statement)
- Initially I thought id give RL just the initial and ground truth polygons as inputs but then I realised this would be pointless when testing so I changed the code so that it observes both the image and vertices
- Used OpenAI Gym (environment interface and a collection of predefined environments in which RL algos can be developed and run) and Stable Baselines 3 (set of reliable implementations of RL algorithms – uses model initialisation, training and feature extraction). The step function determines what happens at each step until the episode ends. This step function has a certain number of outputs – issue is OpenAI Gym expects 5 outputs whereas Stable Baselines 3 expects 4 outputs from step function – needed a wrapper class to convert from one to another. The output in question is a flag (truncated) that tells us if the episode has ended due to reaching an external limit.