Task

We want to optimise the bounding boxes/outlines generated by a model. The specific application is outlining ships in satellite imagery.

The initial problem statement was to use reinforcement learning to improve ship classification in satellite imagery. The plan of action was to train a model to detect ships, then use techniques such as Grad-Cam to visualise which parts of the image the model focuses on when making its decision. This area would then be outlined to give a bounding box/outline which would then be changed using reinforcement learning to match a ground reality polygon outlining the actual ship. Thus, the model would look at more focused sections of the image closer to the actual ship in order to make its decision, this improving accuracy of predictions.

However, this plan of action is not quite realistic as just by reading through it, I myself struggle to imagine how I would go about quantifying parts of the image used by model to come to an answer into a bounding box. Hence, I came up with a slightly more realistic plan.

The ground reality polygons outlining the ships will be generated manually. Outlining models such as Mask RCNN would be used to initially predict an outline of the ship. Then reinforcement learning would be used whereby each vertex of the predicted polygon would be moved until the predicted polygon better matches the ground truth polygon. The aim of this task is to show that there is incremental improvement in performance when the model is trained via reinforcement learning. This can be as simple as comparing the reinforcement learning model to the MASK RCNN model used to generate the initially predicted polygon.

The purpose of this report is to explain reinforcement learning, why it is suitable for this task, how it was applied and how this differs from typical application of reinforcement learning. The bounding_analysis_working_doc.pdf outlines the development of the problem solution to get to the script that generates the initially predicted and ground reality polygons which were imported into the RL script.

Overview of Reinforcement Learning

Reinforcement Learning (RL) allows for the agent to learn by interacting with the environment by making decisions. The agent aims to maximise cumulative reward over time by taking actions based on its observation of the environment.

The environment is the ‘world’ in which the agent operates – essentially any scenario where decisions are made. An example could be a maze which a rover needs to navigate.

The agent is the learner/ decision-maker. It observes the environment, takes actions and learns from the outcomes.

It might be easier to explain methods in the following format:

Step ('step' method):

- This is the code interaction between the agent and the environment. The agent takes an action, and the environment responds by returning:
    - Next state: the new situation the agent finds itself in after the action
    - Reward: A numerical value that tells the agent how good or bad the action was
    - Done: A Boolean flag indicating whether the episode (a complete sequence of actions) has ended
    - Info: Additional information that can be usedful for debugging or further analysis

Reset ('reset' method):

- This method is used to start a new episode. It resets the environment to an initial state, providing the agent with the first observation. This is typically done after an episode ends, to begin learning again from a fresh state.

The agent uses rewards received after each step to learn which actions are good (should be repeated) and bad (should be avoided). Oer time, the agent improves its policy, i.e. the strategy used to pick actions given a certain state.

Why Reinforcement Learning is suited for this task

It allows the model to explore different bounding box configurations and learn from its mistakes. By continuously refining its bounding boxes and receiving feedback on how well they fit the objects, the RL model can develop a robust policy that performs well across different satellite imagery.

It frames the bounding box adjustment as a sequential decision-making process, where each adjustment is an action taken by the agent in response to the current state of the bounding box. The model is trained to maximise the cumulative reward which ensures that the predicted polygon is as close to the ground truth as possible whilst still being a valid polygon.

Simply put, by focusing on fine-tuning each vertex's position based on the satellite imagery, RL provides precise control, making it highly effective for achieving accurate and reliable object detection in complex visual environments.

Best way to explain the application of RL is by describing the flow of code

1. Imports and Dependencies
   - Imports necessary libraries such as Gymnasium (for RL environments), Shapely (for geometric operations) and Stable Baselines 3 (for RL algorithms – PPO is used in current script). Also imports custom utility functions for processing images and generating polygons.

2. 'PolygonEnv' Class

   Initialisation:

   - This class is a custom Gym environment, which is initialised with an initial [predicted] polygon, ground truth polygon and the image in which these polygons aim to outline the ship
   - The action space is a continuous, defined as a 16-dimensional vector representing potential movements for the polygon's vertices
   - The observation space is a dictionary containing the image data and the vertices of the polygon

   'reset' method:

   - Initialises the environment at the start of each episode, resetting the polygon to its initial state and returning the initial observation
   - The observation is a combined 1D array of the image and flattened vertices of the polygon

   'step' method:

   - Where the agents' actions are applied. It takes an action, adjusts the vertices of the polygon accordingly, and computes the new state.
   - The reward is calculated according to the reward policy. This involves maximising IoU value, in other words, maximising overlap between the predicted polygon and the ground truth polygon. In order to avoid the predicted polygon being changed so that it takes over the entire image to maximise overlap, a second factor is taken into consideration – penalisation of non-overlapping areas between the predicted and ground truth polygons. Finally, to ensure that the polygon is valid, i.e. no intersecting edges, a penalty was put into place every time an action causes the edges to intersect with each other. This overall reward policy should ensure that the predicted polygon matches the shape of the ground truth as much as possible whilst remaining a valid polygon.

   Utility methods ('calculate_iou', 'calculate_non_overlap_penalty', 'is_done'):

- Help compute IoU, penalties and check whether the episode should terminate based on predefined conditions. In this script, the conditions are that IoU value i.e. overlap between the two polygons should be above 95% and the non-overlapping areas should be less than 5% of the initial polygon.

3. Wrapper classes

   'GymToStableBaselines3Wrapper':

   - This wrapper converts observations from dictionary format used by the custom environment into a numpy array format expected by SB3. It also converts step method as the custom gym environment generates step method with 5 outputs whereas the SB3 library expects it with 4 outputs ('info' and 'truncated' combined into a singular 'info' key)

   'CustomDummyVecEnv':

   - Extends 'DummyVecEnv' class from SB3, managing multiple environments for parallel processing – essential for efficient training.

   'CustomVecTransposeImage':

   - Processes and transposes observations across multiple environments, ensuring that they are formatted correctly. [might be possible that the current implementation of this in the code needs to be corrected as the RL model mentions type of input information being a tuple (it cannot handle tuples - attempting to change this involves rewriting parts of the SB3 library in a new custom wrapper) and this wrapper is responsible for ensuring compatibility between the input features and the RL model. According to debugging, this wrapper seems to be working correctly however the RL model still reports being given features in the form of a tuple so this needs to be investigated].

   'CustomCNNFeatureExtractor':

   - Custom feature extractor that uses a pre-trained ResNet18 model to extract features from the image, which are then used by the RL model to make decisions.

4. Main function
   Setting up Environment and Data:
   - Main function processes a directory of images, imports initial and ground truth polygons from the gen_predicted_grndreality_polygons.py script.
   - Iterates over these images, creating an instance of the 'PolygonEnv' environment for each image

Training the RL model:

- Environment is wrapped in custom wrappers to ensure compatibility with SB3
- PPO model is created with a custom policy
- Model is trained in steps, where both the model and rewards are saved at intervals

How this script differs from typical reinforcement learning scripts

In usual RL tasks, observations are simple (e.g. position and velocity in Catpole) and actions are discrete or low-dimensional. However, in this script, the observations include high-dimensional image data and vertices of the polygon. The action space is continuous and involves directly manipulating the vertices of a polygon, which is more complex than typical RL scenarios.

General RL implementations directly interact with the environment whereas the current scrip involves multiple custom wrappers for the environment and data to account for incompatibility between SB3 and gym libraries.

Typical RL environments have straightforward reward (e.g. +1 for reaching a goal and -1 for failure) whereas this implementation takes into account IoU between polygons, penalties for non-overlapping areas as well as checking for polygon validity – this also makes the termination condition more complex.