Vessel Detection in Sentinel-1 Imagery

Favyen Bastani, Piper Wolters, Rose Hendrix, Joseph Ferdinando, Ani Kembhavi
Allen Institute for Artificial Intelligence, Seattle, USA

I. Introduction

In this document, we detail the approach in our xView3 submission. The xView3 dataset presents the challenge of detecting vessels and other maritime objects in synthetic aperture radar (SAR) images captured by the ESA's Sentinel-1 satellite. The dataset spans over 600 Sentinel-1 scenes, and tasks include detecting maritime objects, classifying whether objects are vessels and whether vessels are fishing vessels, and estimating the lengths of vessels.

Although we experimented with several object detection approaches tailored for the vessel detection problem, such as comparing images of the same location captured at different times to distinguish mobile ships from static islands and platforms, we found that a standard object detection pipeline performed best on this dataset.

Thus, as in the reference code, we train a Faster R-CNN model with a ResNet-50 backbone. We use only the vh, vv, and bathymetry channels as input to the model. To make the best use of the provided data, we employ several data augmentations; to account for the partially labeled training set, we adopt a pseudo-labeling-like approach to add missing labels in the training set.

II. Data Pre-processing

We first describe how we re-normalized the provided data, and how we split the data into training and validation splits.

A. Data Normalization

We re-normalize the provided SAR vh and vv intensities, along with the bathymetry/elevation data, to values between 0 and 1. Note that the provided SAR intensities have already been pre-processed with logarithm and other operations, while the bathymetry is provided in meters. Additionally, bathymetry and elevation are combined into one channel, which is negative over water bodies to provide depth information, and positive over terrain to provide elevation information.

The reference code normalizes all inputs by computing the minimum and maximum values in each channel for every 800×800 window, and linearly mapping that range to $[0, 1]$. However, since this normalization is conducted on a per-window basis, it does not preserve absolute intensity and bathymetry values: for example, if a given window is fully on land, the original bathymetry values would all be positive, but it would not be immediately clear that it is land from the normalized values.

Thus, instead, we adopt a fixed global normalization for each channel, as follows:

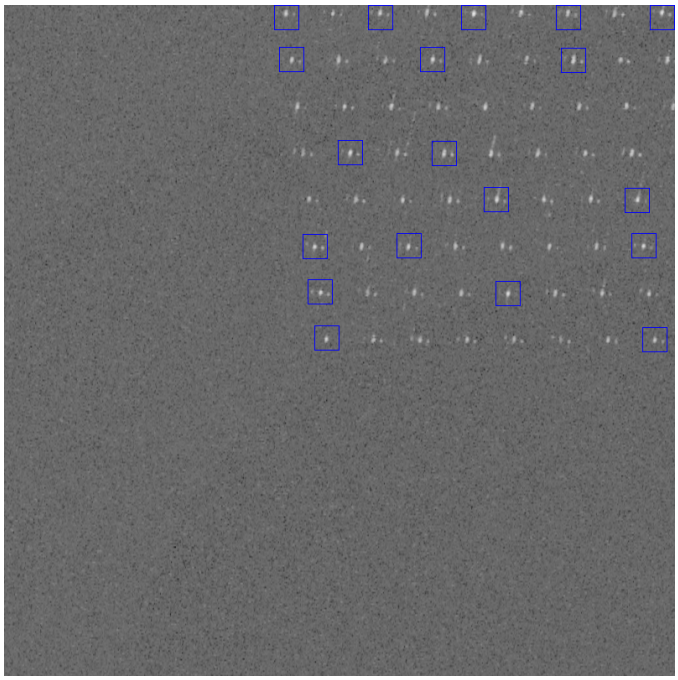

Fig. 1: Since scenes in xView3-Train are automatically labeled using AIS data, in some cases there are missing labels. Here, only a subset of the wind turbines in an off-shore wind farm are annotated (blue boxes).

- vh, vv: we clip values to $[-50, 20]$, and re-scale linearly to $[0, 1]$.
- Bathymetry: we clip values to $[-6000, 2000]$, and re-scale linearly to $[0, 1]$.

B. Training and Validation Splits

The xView3 dataset consists of two subsets:

- xView3-Train: 554 Sentinel-1 scenes labeled automatically using AIS data.
- xView3-Validation: 50 Sentinel-1 scenes labeled largely by hand.

xView3-Validation is labeled more comprehensively: in xView3-Train, since AIS data is the primary source of ground truth, if a maritime object is not broadcasting AIS then it will not be labeled. We show an example of this issue in Figure 1.

Furthermore, xView3-Train includes 315 scenes in the Bay of Biscay that appear in neither xView3-Validation nor the public test set.

Although the subsets are named to suggest one subset be used for training and one for validation, the competition

organizers stated that participants can leverage the provided data in different ways.

Thus, because xView3-Validation consists of a different data distribution, which in theory is closer to annotations in the test set, we opt to establish modified training and validation splits. Our training set consists of 239 scenes in xView3-Train (we exclude scenes in the Bay of Biscay) and 25 of the 50 scenes in xView3-Validation. Our validation set consists of the remaining 25 scenes in xView3-Validation.

C. Bounding Boxes

While fine-grained bounding box labels are provided in xView3-Validation, xView3-Train only includes point labels specifying the center position of maritime objects. We opt to only use the point labels across the entire dataset. To train the Faster R-CNN detector, which expects bounding box labels, we construct boxes of a fixed 40×40 size around the point labels.

III. Methodology

A. Data Augmentation

We employ several data augmentations to make the best use of the training and validation data.

Random Cropping. We extract random 800×800 windows from the scenes, which are around 30,000 $\times$ 20,000 pixels in size. In contrast to the reference code, the selected windows need not be aligned with a grid: we select the top-left corner of the window uniformly from the scene, while excluding windows that do not contain any valid data.

Random Flipping. We randomly flip the cropped windows horizontally and vertically with 0.5 probability each.

We also experimented with random jitter, random resizing, random rotation, and Gaussian noise, but these augmentations did not provide an improvement in performance.

B. Object Detection Model

We use Faster R-CNN with a backbone consisting of ResNet-50 followed by a feature pyramid network. The backbone is pre-trained on ImageNet while the RPN and ROI heads are randomly initialized.

We extend this model in three ways:

- We apply the backbone to also predict other provided attributes of object labels.
- We extend the backbone to support pseudo-labeling that adds labels with non-binary ground truth probability distributions.
- We do not back-propagate the loss in regions of the image intersecting low-confidence labels.

We also use a sliding window inference scheme that ensures predictions are made on window edges only when needed. After describing the training procedure, we detail each of these extensions below.

Training. We use SGD optimizer with 0.9 momentum and 0.0005 weight decay. We initialize with a 0.001 learning rate, but decay the learning rate two times by 1/10 down to 0.00001; the decay is scheduled based on when the training loss plateaus.

The training data is sparse, consisting of many more background windows with no objects than foreground windows containing object labels. Thus, as in the reference code, we opt to balance the frequency at which we present the model with background and foreground windows, and arrange for training batches to consist of 2/3 foreground and 1/3 background on average. In contrast to the reference code, though, we make full use of all background windows, instead of selecting a subset of background windows upon initialization: specifically, we sample each training batch from the set of all windows, and use weighted sampling (where background windows are assigned a lower weight than foreground windows) to achieve the 2/3 to 1/3 ratio.

Attribute Prediction. In addition to specifying an object's center position, the ground truth data includes several other attributes, including a confidence estimate (low, medium, or high), whether the object is a vessel, whether the object is a fishing vessel, and the vessel length. We extend our model to predict these attributes as a joint training task, in the hope that learning to predict attributes will help the model become better at detecting objects.

To do so, during training, we apply the ResNet-50 backbone on 128×128 crops around each ground truth label. Note that this is done separately from the Faster R-CNN pipeline: we do not use the predicted proposals or ROI pooling here, but do share the backbone parameters. We then apply a three-layer head for each attribute. We predict the label confidence, vessel class, and fishing vessel class using softmax heads trained with cross-entropy loss, and predict the vessel length using a regression head trained with L1 loss. Losses are masked if the attribute is not available for a particular object label. We add these losses to the Faster R-CNN losses with a 1/20 re-weighting.

Since we predict the vessel and fishing classes using this auxiliary attribute prediction model, we do not require the Faster R-CNN ROI head to perform classification. Thus, we train a single-class Faster R-CNN model, leaving classification up to the attribute predictor.

Pseudo-labeling. We employ one round of a pseudo-labeling-like approach to fill in missing labels in xView3-Train. Specifically, we train our model on all 50 scenes in xView3-Validation, and apply it on each scene in xView3-Train. We identify high-confidence predictions (confidence score exceeding 0.8) that are at least 20 pixels away from the closest pre-existing ground truth label, and add those points as additional labels. We annotate each of these new points with the confidence score that the model produced. We also eliminate ground truth labels where the model did not make any prediction with at least 0.1 confidence score.

We then train the final model on our training set with these new labels included. However, when computing the binary cross entropy objectness loss for these labels, we use the confidence score from the original model as the ground truth

positive class probability rather than a binary ground truth distribution. This way, the added labels are treated as "softer" targets than the provided ground truth labels, reflecting that they may not all be correct.

Low-confidence Labels. Labels in the xView3 dataset include a confidence attribute, which may be low, medium, or high. During evaluation, only predictions matching with medium- and high-confidence labels are rewarded when computing recall, while predictions matching with low confidence labels are ignored.

Thus, we opt to use only medium- and high-confidence labels as positive examples when training the detector. Nevertheless, intuitively we should not penalize the detector for predicting points corresponding to low-confidence labels. To accomplish this, we modify the RPN and ROI heads to discard proposals that intersect with low-confidence labels with at least 0.5 IoU.

Sliding Window Inference. Due to the large size of the Sentinel-1 scenes (approximately 30K $\times$ 20K pixels), GPU memory limitations make it infeasible to provide the entire scene as input to the model. Instead, we adopt a sliding window inference strategy, where we divide the scene into 3072×3072 windows and process each window independently. We can then concatenate the predictions across windows to derive the final output.

However, we find that predictions along the borders of these windows are less accurate than predictions well in the center of the windows. For example, a vessel that is cut in two across two windows may be missed or even predicted twice. To mitigate issues along window borders, we employ an overlapping window inference strategy.

Specifically, we use 3072×3072 windows during inference, but arrange for windows to share an 800-pixel overlap with each adjacent window above, below, to the left, and to the right. We only retain predictions in the center 2272×2272 of the window, with the exception of windows on the border of the Sentinel-1 scene, in which case we retain predictions up to that border. Finally, after concatenating predictions across windows, we perform a duplicate pruning step to remove predicted points that are within 10 pixels of a higher-scoring point. This strategy ensures that the model has enough context to make each prediction.

Exponential Moving Average. We find that using the exponential moving average (EMA) of parameters from training provides more stable performance during inference than using the best-performing or most recent parameters. Thus, we save the parameters using EMA with a decay of 0.995 on each training step.

C. Attribute Prediction Model

Although we extend our object detector to predict attributes such as vessel length, we find that its accuracy is limited and highly variable. Thus, while the auxiliary attribute prediction heads were useful for joint training and improving detection performance, a separate attribute prediction model provided better accuracy at actually recovering the attribute values.

Our attribute prediction model architecture is similar to the auxiliary attribute prediction heads detailed in the previous sub-section. The model inputs 128×128 crops around ground truth points, and predicts the confidence annotation, binary vessel class, binary fishing class, and vessel length. To train the model, in addition to random vertical and horizontal flipping, we translate the crop by up to 8 pixels in either direction along both axes.

During inference, we apply the model on a crop around each point predicted by the object detector. We label the point as a vessel if the vessel class probability exceeds 0.5, and label it as a fishing vessel if the fishing class probability also exceeds 0.5. We directly use the predicted vessel length in the output as well.

D. Hyperparameter Tuning

We tune the confidence threshold to obtain the maximum score on our validation set. We also evaluate the model parameters (after EMA) periodically during training, and save the parameters that provide best performance on our validation set.

E. Test-Time Augmentation and Ensembling

We do not use an ensemble of models. However, we do employ test-time augmentation to derive a small improvement in performance. Specifically, during inference, we apply the model four times: once on the original image, once with the image flipped vertically, once with it flipped horizontally, and once with it flipped both ways. We then average the scores of predictions produced across the four augmentations.

IV. Approaches Tailored for Vessel Detection

We experimented with several approaches tailored specifically for the vessel detection problem. However, we did not attain any performance improvement with these approaches. Nevertheless, we detail these attempts, which were not ultimately incorporated into our final approach, in this section.

A. Comparing Overlapping Images

Intuitively, comparing the current window to windows extracted from spatially overlapping Sentinel-1 scenes captured at different times should improve performance, by enabling the model to avoid predicting static islands and rocks as maritime objects. For example, we might find a feature in the current image that looks like a vessel, but appears in every overlapping image as well; in this case, we may conclude that the feature is actually a small island and not a vessel. This approach was very feasible to adopt for xView3 since the vast majority of scenes in xView3 overlap with at least one other scene in xView3-Train.

We experimented with several approaches that implement this idea. In one variation, we employ a shared encoder that inputs a two-channel image consisting of the SAR vh and vv channels, and outputs a feature map at a reduced resolution. We first compute features through the encoder in the current window w. We then identify up to four windows from other

Sentinel-1 scenes that overlap w, apply the encoder independently on each of these windows, and average the extracted features at each pixel across the windows. We concatenate the features computed in the current window with the averaged features computed in overlapping windows, and feed the concatenated result to the feature pyramid network and the RPN and ROI heads. In this way, downstream networks can compare the feature representation of pixels in the current window against averaged features of the same pixels in windows from Sentinel-1 scenes captured at different times.

However, none of the variations we tested provided a performance improvement.

B. Static Infrastructure

We found that many maritime objects in xView3 correspond to static infrastructure such as wind turbines. Intuitively, localizing these objects a priori and reproducing them in the appropriate positions at inference time should improve performance. We localize static infrastructure by identifying longitude-latitude points that are covered by a point within 10 pixels across 5 different spatially overlapping Sentinel-1 scenes: if a maritime object is annotated in the same location across so many images, then it is likely a static object. However, we did not observe a performance improvement from this approach.

V. Conclusion and Acknowledgements

Our approach achieves 0.576 aggregate score on the public test set, and 0.578 aggregate score on the hidden test set. We would like to thank Global Fishing Watch and Defense Innovation Unit for releasing the xView3 dataset and organizing the competition.