Airbus Ship Detection

Xinyuan Lu, xlu25@ucsc.edu; Linyin Yang, lyang34@ucsc.edu; Haonan Xu, hxu24@ucsc.edu

Abstract

Machine learning has always been a game of trading off. Which aspect of the model do we want the most, training time, accuracy, Inference speed, or transfer learning abilities? In our project, we aim for accuracy and inference speed and came up with a two levels network tree that trades a little bit accuracy for a boost in inference speed. The first neural network is a convolution neural network that only performs simple classification tasks and feeds all the positive instances to the second neural network. The second neural network will do some more computationally intensive work based on the given input. By doing so, we were able to increase the inference speed by ~28% without losing too much accuracy.

Introduction

Our task, Airbus Ship Detection, is one of Kaggle's prediction challenges. We aim to find ships on satellite images as accurate as possible and as quick as possible. This project builds a model of automatic ship detection, in order to meet the requirements for both accuracy and inference speed. The general idea is to classify whether there are ships or no ships in the images first and if there are ships, then to locate the then.

Dataset

Our dataset contains approximately 190k 768x768 satellite images. Some images have ships, either one or multiple. Others have no ships but oceans, shores, and clouds. One csv file provides ground truth data of encoded pixels for the train set. The following figures are some example images in our dataset.

Fig.1 Image with multiple ships

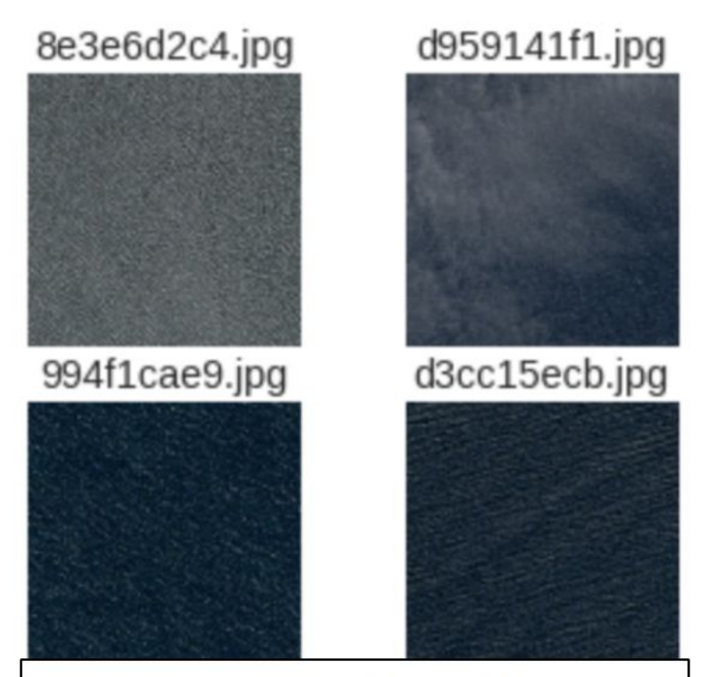


Fig.2 Images with no ships

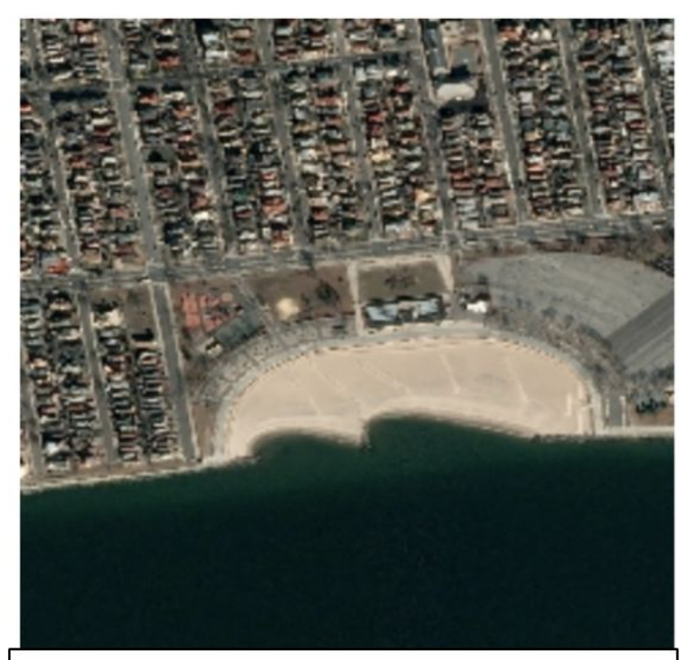


Fig.3 With shore but no ships

Fig.4 Ship covered with clouds

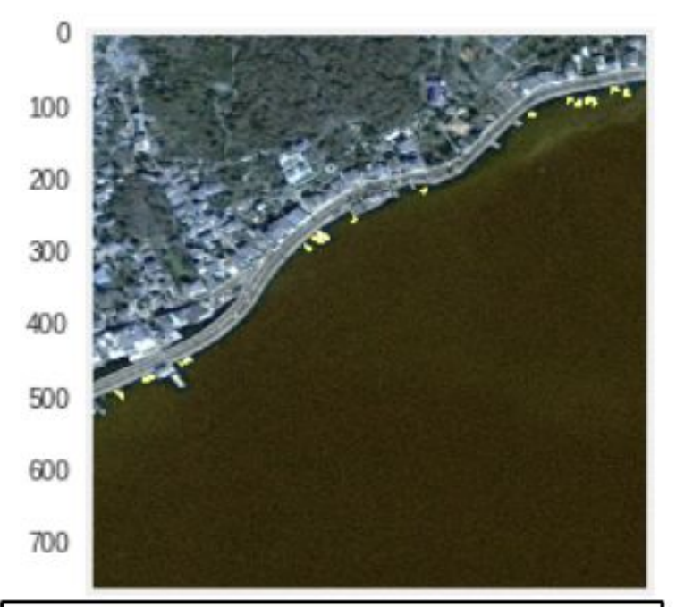
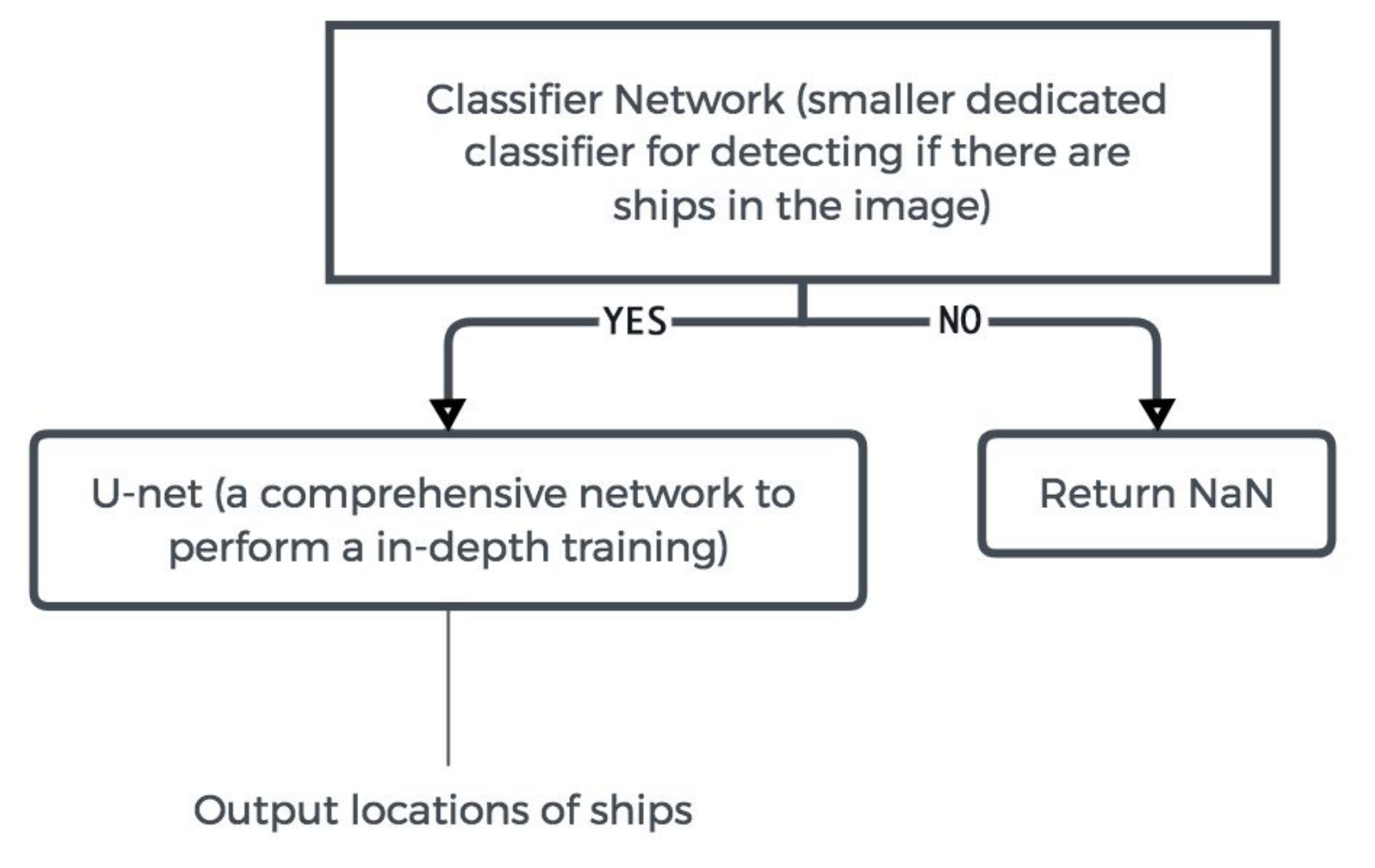


Fig.5 Masked ships along shore

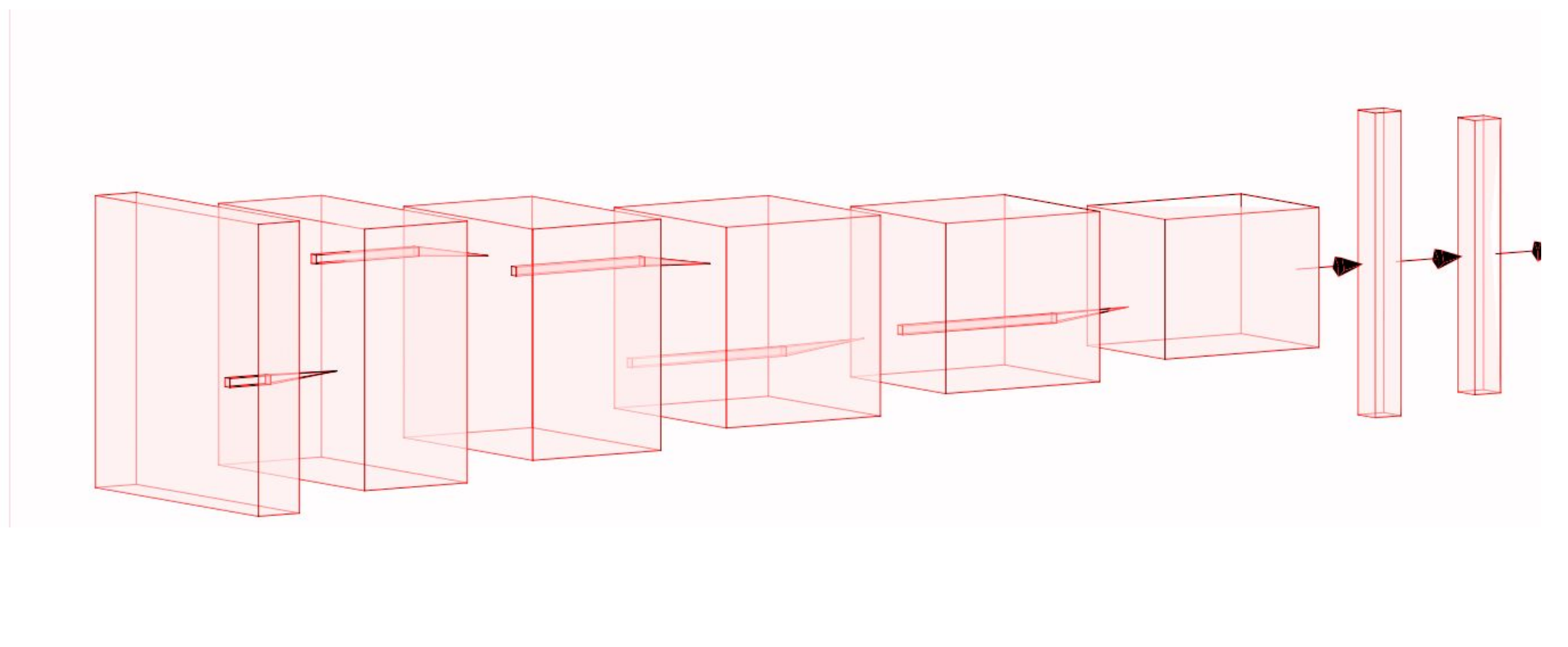
Fig.6 Very tiny ship

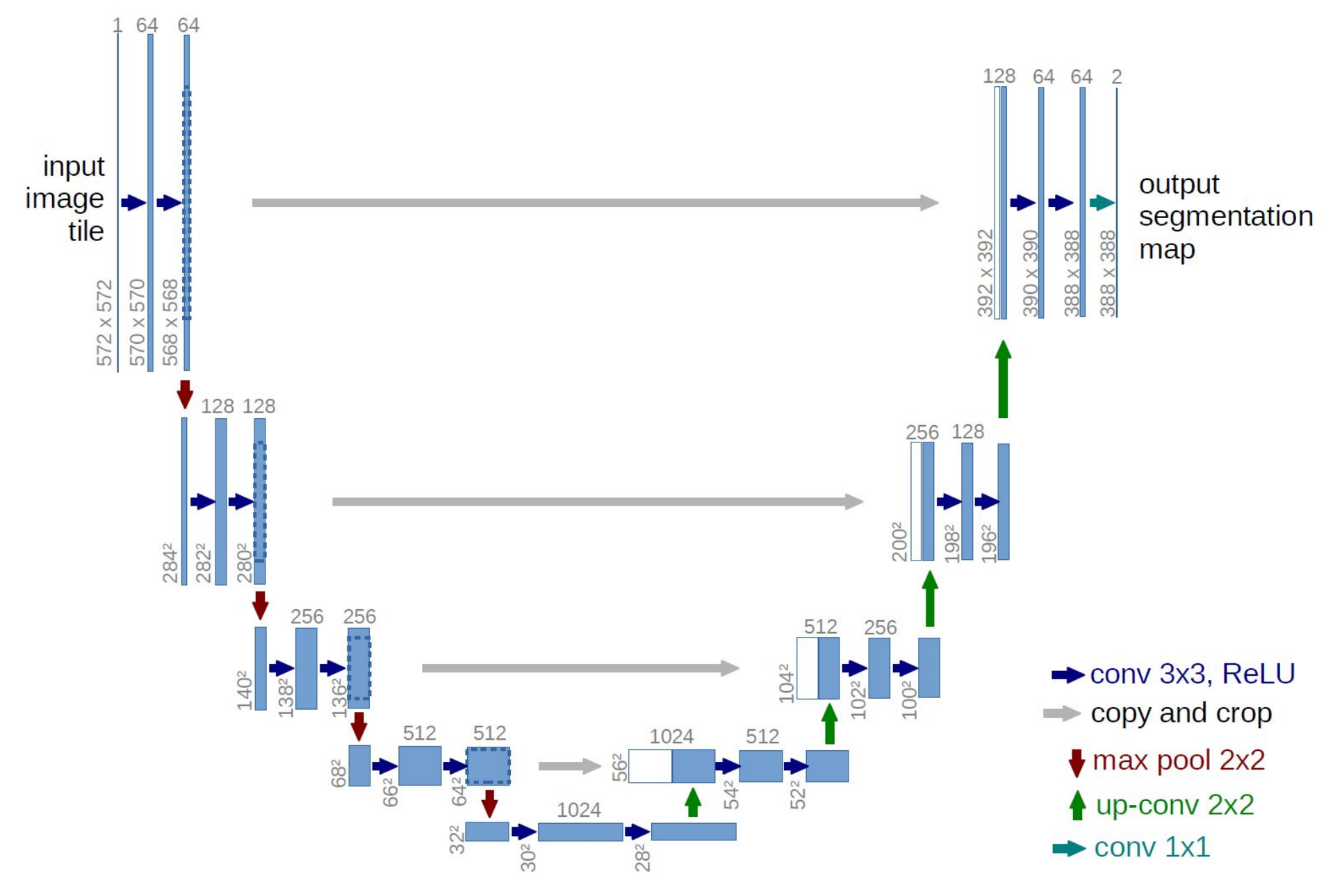
Models


Classifier Network (smaller dedicated classifier for detecting if there are ships in the image)
YES
NO
U-net (a comprehensive network to perform a in-depth training)
Return NaN
Output locations of ships
Overall structure of our network

In order to reduce the size of our input and potentially improve our detection speed, we build a two stage reference structure which comprises a cnn classifier network and a deep U-Net network. If an image with ships is detected in the classifier network, it will pass into the dataset for U-Net; Othersize, the images will be simply ignored.

The first network is classifier network, contains 5 convolution layers and 2 fully connected layers. This network is simple enough that it took a comparably little amount of time to do the prediction task, yet powerful enough to give us a good performance.


Structure of classifier network

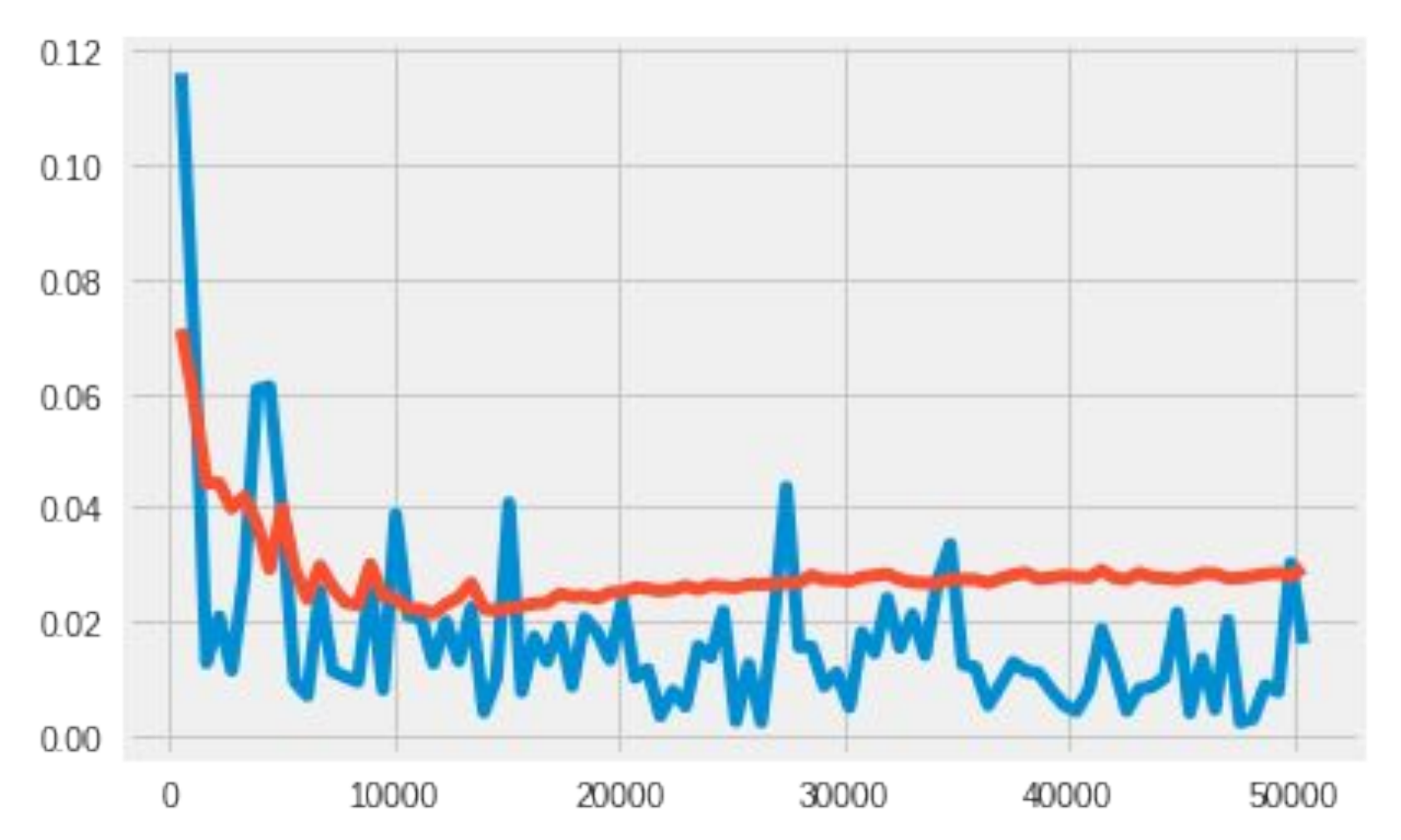


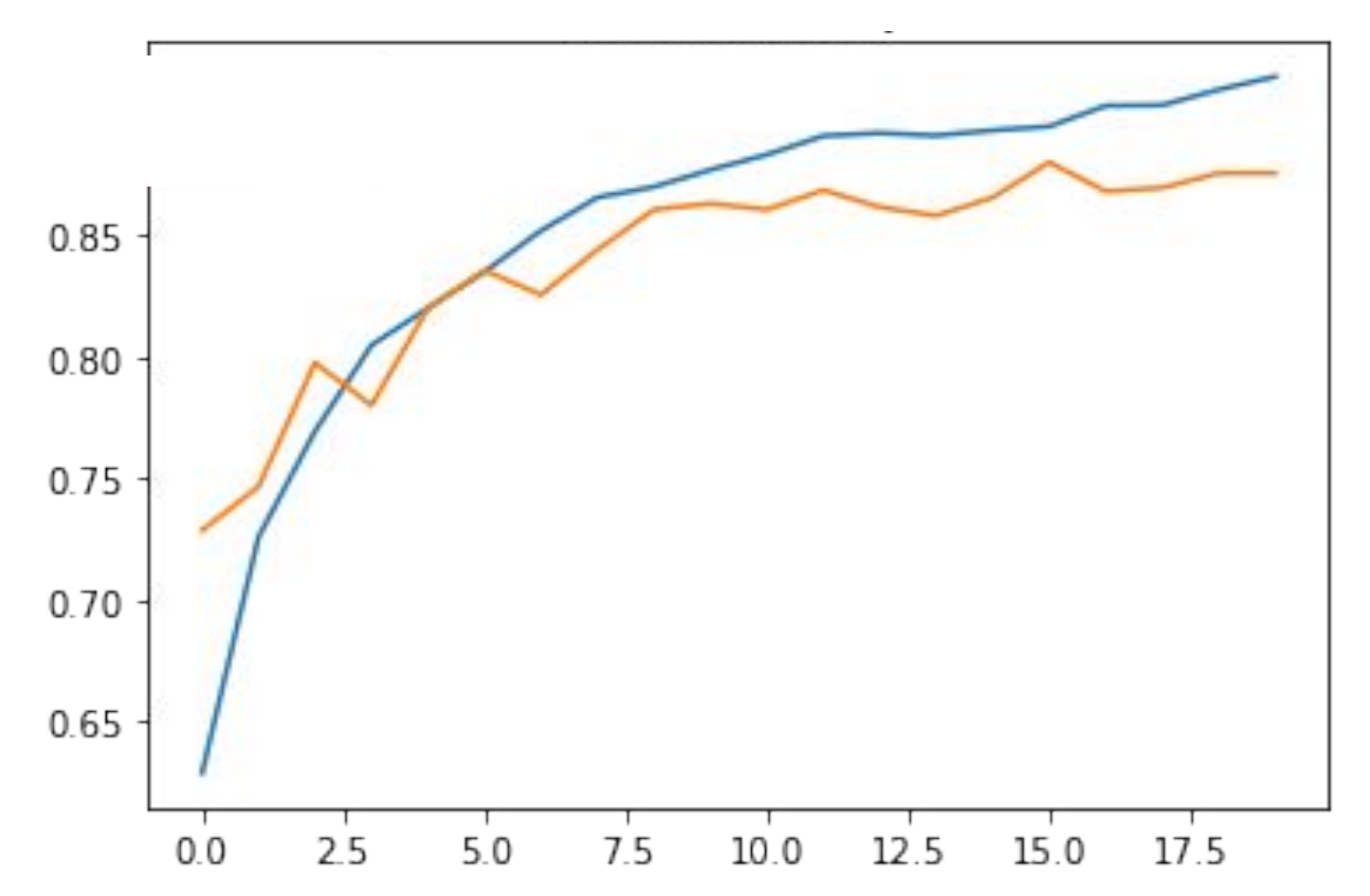
Example U-shape architecture of U-net

U-net is a deep convolutional neural network designed for image segmentation. It contains an usual contracting network and modifies a large number of feature channels in the successive layers (replacing max pooling with upsampling), which allow the network to propagate context information to higher resolution layers.

Experiments

We calculated our loss using cross-entropy function.
For U-net, we trained our model on full image resolution for 5 epoches, and for classifier network, we trained on full image resolution for 18 epoches. Here is the result.


U-Net Losses over iterations


Classifier Network accuracy for train and validation set over epoches

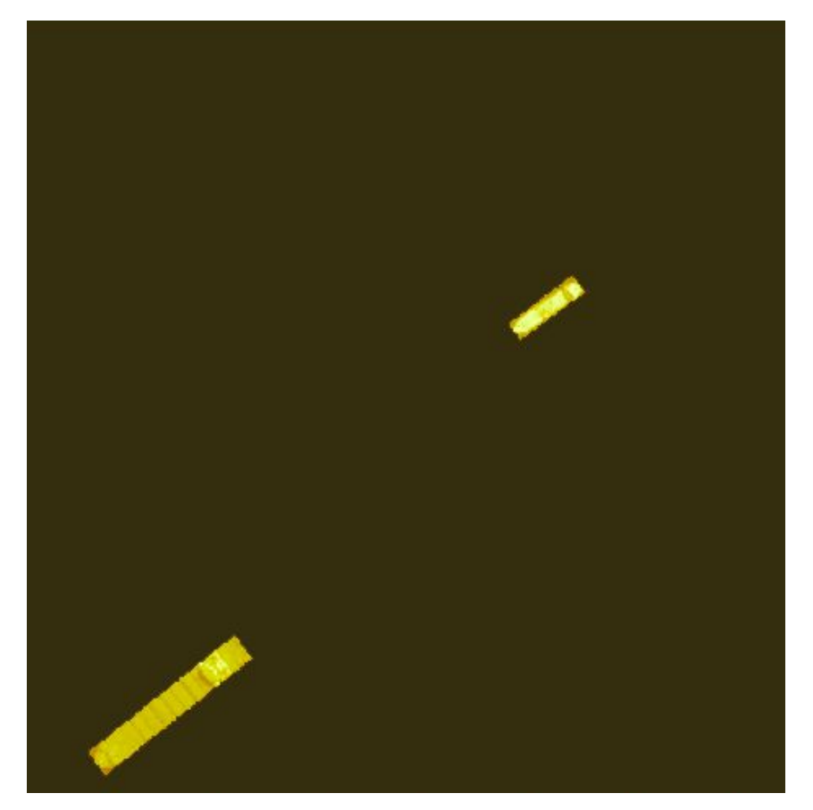


Fig. 1 Ground truth mask over image

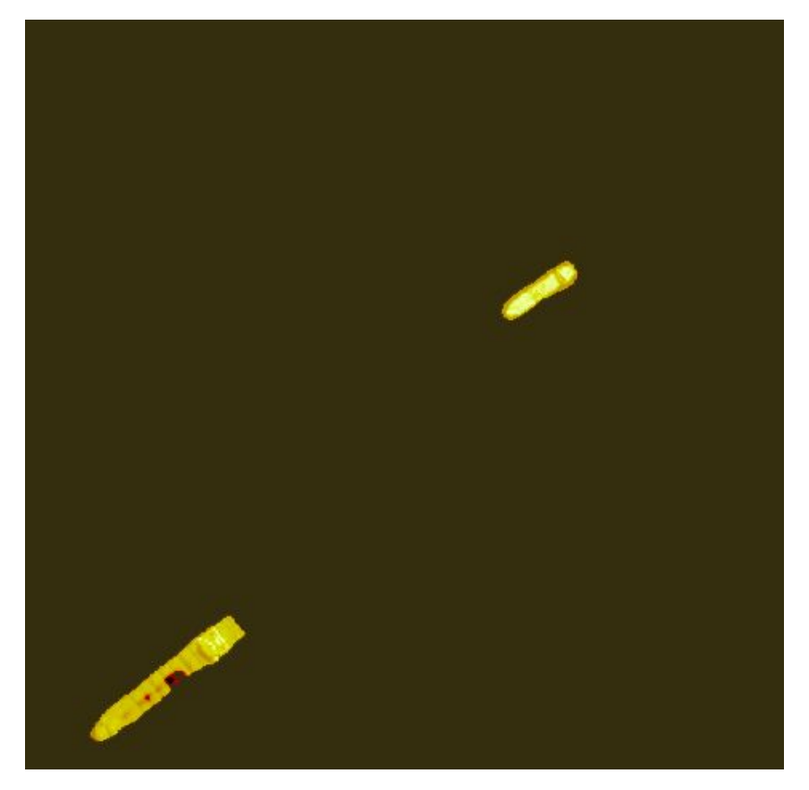
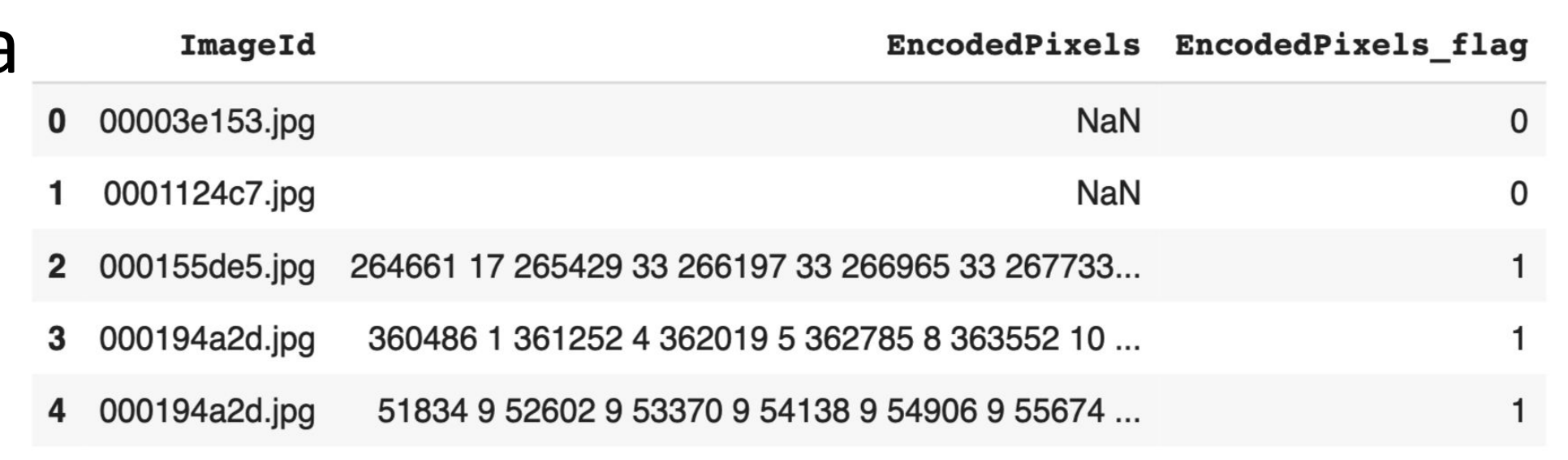


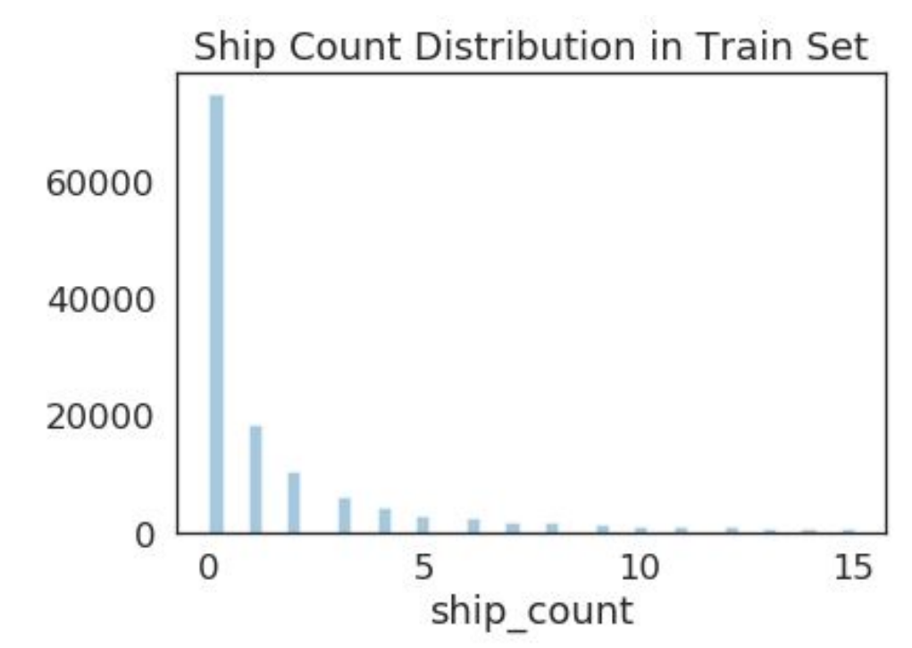
Fig. 2 Ground truth mask over the same image(One of the best results!!)

Preprocessing & Data Loading

To better classify if an image contains a ship or not, we map encoded pixels flag to 1 if there is a ship and to 0 otherwise. By grouping image_ids, we successfully splits images into different categories for future analysis. (Lower right). Also, we used transform module from Pytorch.torchvision to achieve common image transformations on masks as well as on training a

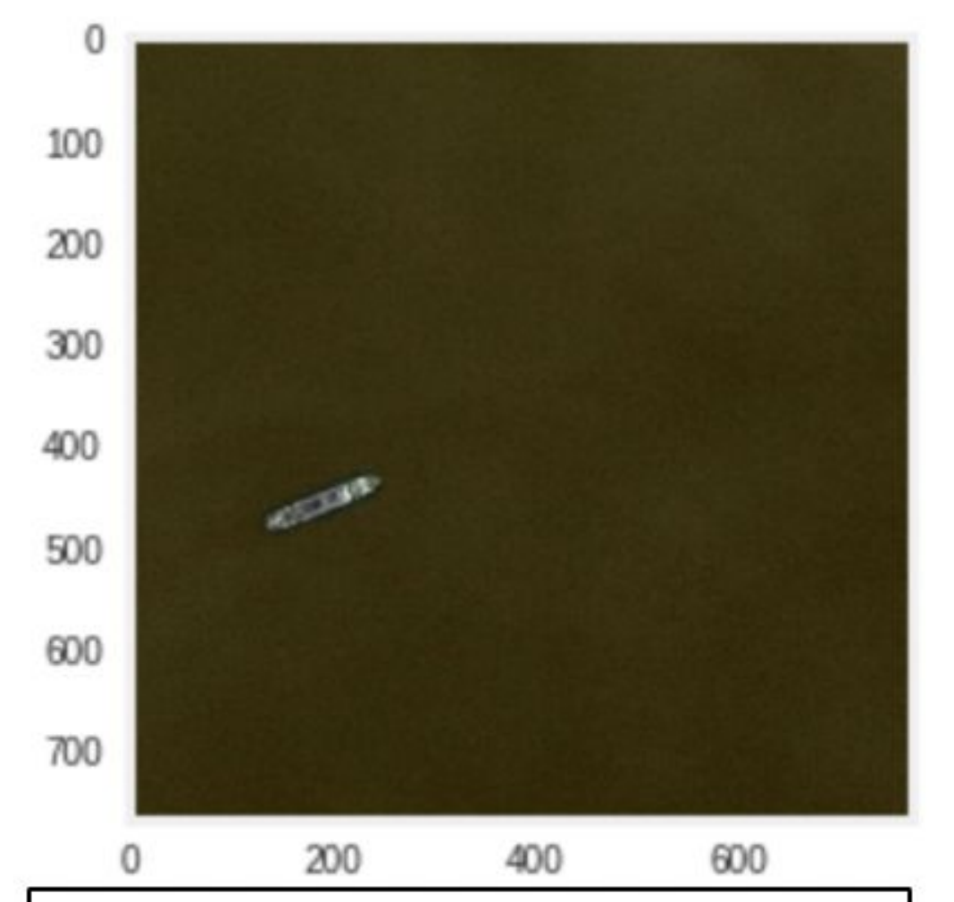


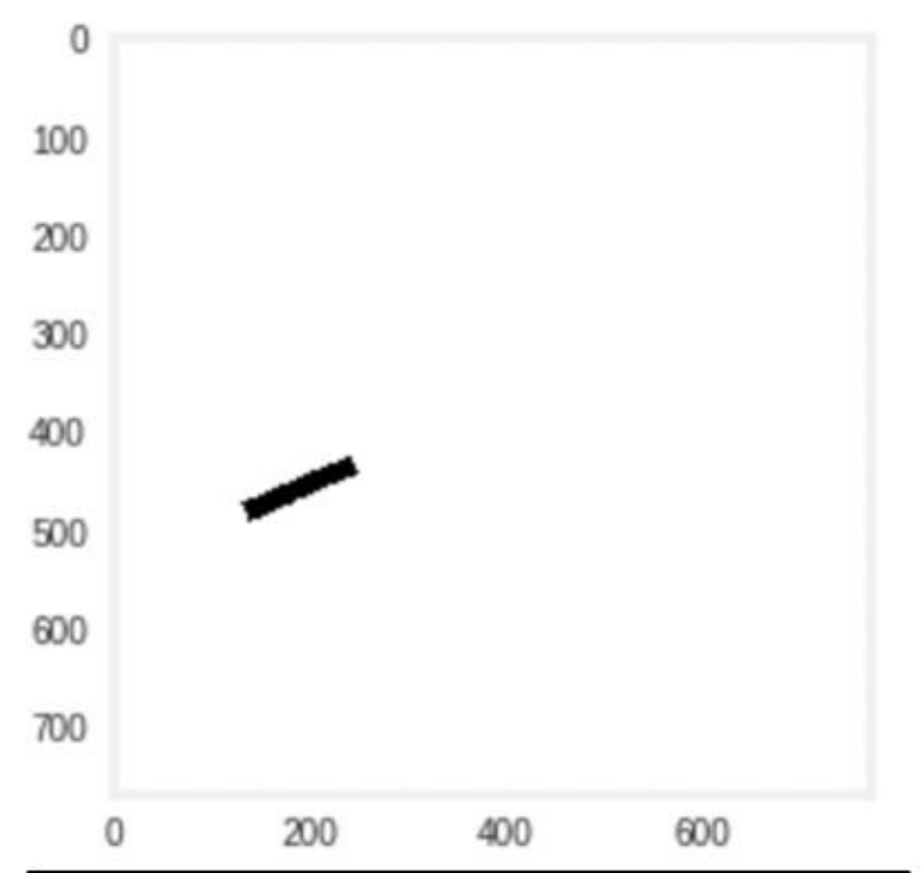
	ImageId	EncodedPixels	EncodedPixels_flag
0	00003e153.jpg	NaN	0
1	0001124c7.jpg	NaN	0
2	000155de5.jpg	264661 17 265429 33 266197 33 266965 33 267733...	1
3	000194a2d.jpg	360486 1 361252 4 362019 5 362785 8 363552 10 ...	1
4	000194a2d.jpg	51834 9 52602 9 53370 9 54138 9 54906 9 55674 ...	1

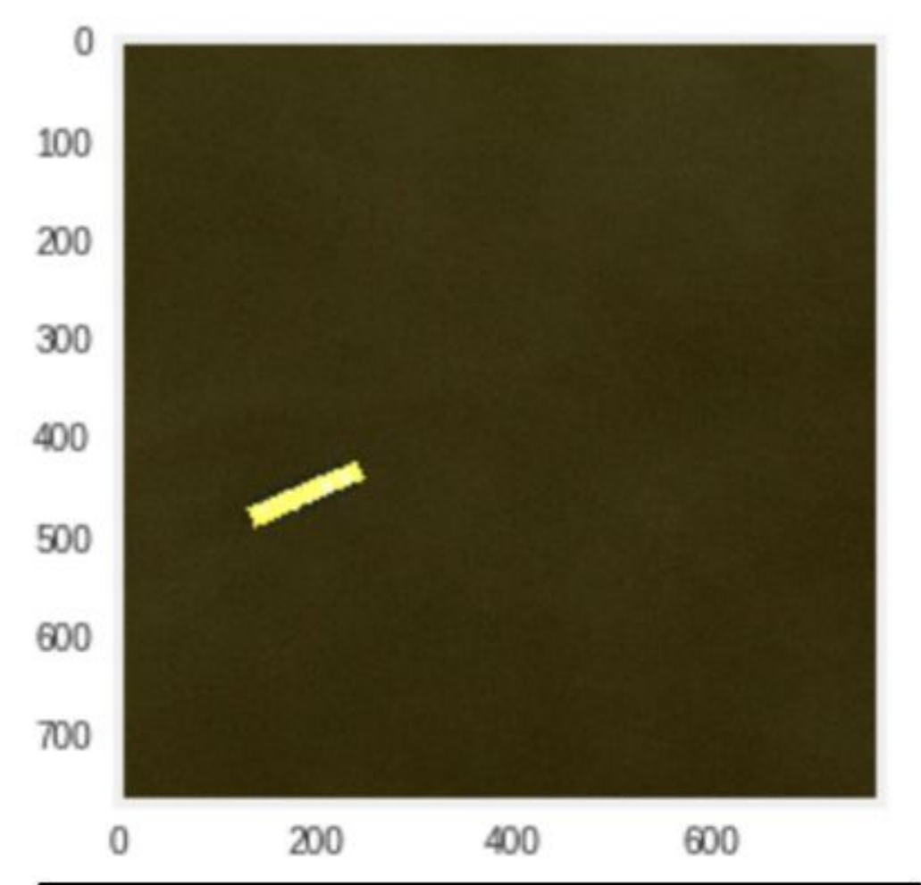


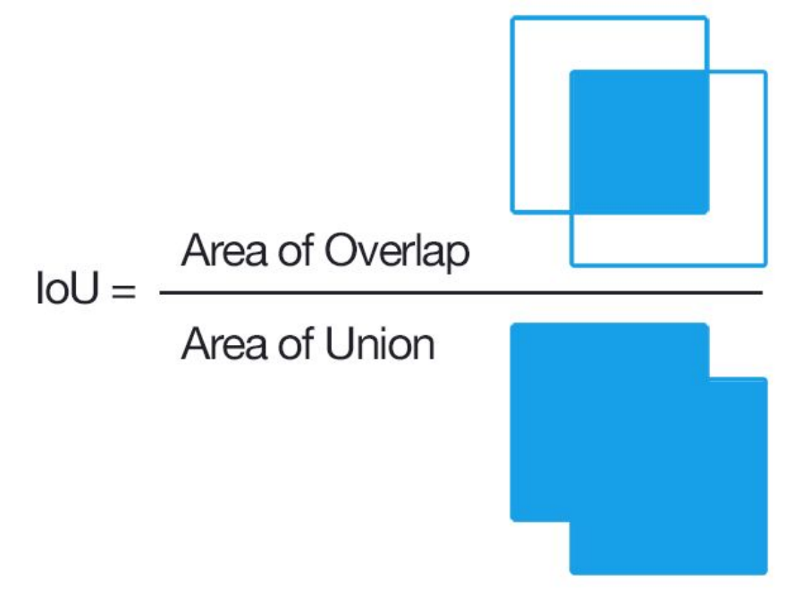
Run-Length Encoder, Mask & IoU

Since the encoded pixels for the training set are given in run-length encoding (RLE) format, we need to first understand what RLE is. RLE is a very simple form of lossless data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. Each string from the original csv file only represents one bounding box of one ship. If an image contains more than one ship, there will be multiple lines in the file that record different bounding boxes of the same image. The encoded pixels can be easily turned into 2d matrices


Original Image with ship


Plotted mask of bounding box


Mask over boat



Intersection of Union, IoU, a statistic used for comparing the similarity and diversity of sample sets. We used this function as our accuracy metric for location detection.

Conclusion

We reached a 87% test accuracy in our classifier network. We also reached 54% IoU accuracy in U-net.
Compared to vanilla U-net, the inference speed increased by ~28%, where the inference accuracy of the network only dropped by ~6%. We considered this as a successful trade off between prediction speed and accuracy.

Reference

1.Kaggle source https://www.kaggle.com/c/airbus-ship-detection
2.IOU metrics https://www.kaggle.com/iezepov/fast-iou-scoring-metric-in-pytorch-and-numpy
3.U-Net documentation https://github.com/jvanvugt/pytorch-unet
4.Rle bounding boxex https://www.kaggle.com/julian3833/2-understanding-and-plotting-rle-bounding-boxes