Airbus Ship Detection

Xinyuan Lu
Jack Baskin School of Engineering
UC Santa Cruz
Santa Cruz, CA, USA
xlu25@ucsc.edu

Haonan Xu
Jack Baskin School of Engineering
UC Santa Cruz
Santa Cruz, CA, USA
hxu24@ucsc.edu

Linyin Yang
Jack Baskin School of Engineering
UC Santa Cruz
Santa Cruz, CA, USA
lyang34@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 to opt for accuracy with inference speed. During the class, we learned important machine learning concepts such as overfitting, residual network, one by one convolution cell, and etc. These knowledges enabled us to convert papers into working solutions. In this project, we will compare and contrast two different solutions in terms of inference speed and inference accuracy. One solution is U-net[1], a deep convolutional neural network that used for image segmentation. Another solution is a two-layer neural network tree that consists of u-net and a classifier network, we will call it fast inference net in the paper later.

KEYWORDS

U-net; Deep Convolutional Neural Network; Hierarchical Inference Structure; Large Dataset; Attention Models

1 Motivation and Objective

Our project is Airbus Ship Detection from Kaggle competition challenges list. Shipping traffic is growing fast. 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. We aim to find ships on satellite images as quickly as possible, to fight against potentially devastating events at sea like ship accidents, piracy or illegal fishing.

In this project, we will build different models for automatic ship detection from satellite images and test their prediction performances. The ultimate goal is to achieve both inference speed and inference accuracy for the prediction task. A lot of work has been done over the last 10 years to automatically extract objects from satellite images with significative results, according to the competition host, but no effective operational effects. Our project achieved a more balanced solution compare to the existing ones.

2 Dataset

2.1 Run-length Encoder (RLE) and Masks

The given dataset contains approximately 190k 768x768 satellite images, 100k for the training set and 90k for the test set. Some images have ships, either one or multiple. Others have no ships but complex terrains such as oceans, shores, and clouds. One .csv file provides ground truth data of encoded pixels for the train set. Figure 1 shows some example images from our dataset.

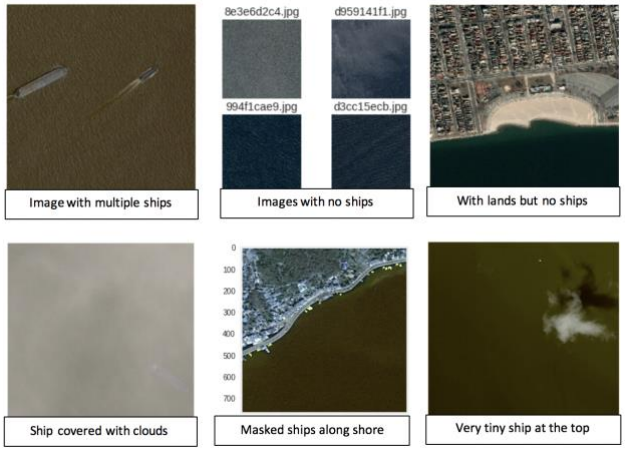

Figure 1: Example of images of the dataset.

Since the encoded pixels for the training set are given in run-length encoding 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

[1] U-net. *Wikipedia*. https://en.wikipedia.org/wiki/U-Net

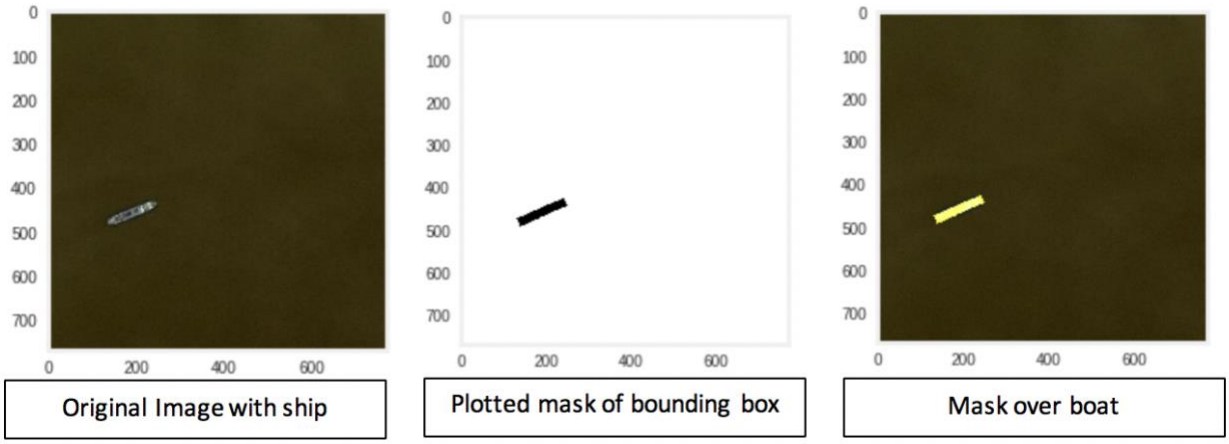

Figure 2: Example of decoding ground truth data: A training instance (Left); A plotted mask after decoding the original string (Middle); Mask over the ship (Right).

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. In this particular case, the strings are in *start* and *length* pairs. Each string from the original csv file only represents one bounding box of one ship. If an image contains more than one ship, there are multiple lines in the file that record different bounding boxes of the same image. The encoded pixels can be easily turned into 2d matrices which we utilized to plot the bounding boxes. We call these outputs *masks*. Figure 2 presents how we plot an encoded string from given ground truth to images/masks after decoding. This procedure is for both labeling and visualization purposes that help in training stage.

2.2 Preprocessing and Data Loader

Given the original image dataset, we have image ids corresponding with encoded pixels of a ship. To better understand and classify if an image contains a ship or not, we map encoded pixels flag 1 to images that have a ship and 0 to images without any ships (Figure 3). We then group them by image id to get the number of ships in each image (Figure 4).

	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

Figure 3: Images with EncodedPixels_flag

By doing this, we could successfully classify images into two groups: with and without ships, which makes sampling and future data preprocessing more efficient. Knowing the size of the dataset and the goal of our task, we sub-sample images without ships in order to have a sufficient training set. Then we concatenate two groups of images and randomly split into train and test set.

	ImageId	ships
0	00003e153.jpg	0
1	0001124c7.jpg	0
2	000155de5.jpg	1
3	000194a2d.jpg	5
4	0001b1832.jpg	0

Figure 4: ImageIds with number of ships

We deploy transform module from Pytorch torchvision to achieve a common image transformation for train, validation, and mask. Its functionality also provides us with fine-grained control over the transformations. Having all the necessary functions defined, we then prepare our data by saving data path, selected datasets, mask encodings, transform functions and hyper-parameters in train and validation data loaders for training.

3 Models and Algorithms

3.1 Vanilla U-net Network

For models, we first implement a U-net[2]. U-net is a deep convolutional neural network designed for image segmentation. The network is based on the fully convolutional network, and its architecture was modified and extended to work with fewer training images and to yield more precise segmentation. It contains a usual contracting network and modifies a large number of feature channels in the successive layers (replacing max pooling with up-sampling), which allow the network to propagate context information to higher resolution layers. Segmentation of a 768 by 768 pixels image takes less than a second on a recent GPU.

For U-net, we use some Pytorch default functionalities to help us in backpropagation, but we implement the 2 by 2 convolution cell and 1 by 1 convolution cell ourselves. However, it is still pretty hard for us to implement everything, so we search for some inspiration from the kernels in Kaggle[3]. After comparing our chunky implementation and their implementation, we decide to convert to their implementation almost completely.

However, the U-net model we implement diverges a lot from their implementation. First, since we would like to have high accuracy, we change the image that used for training into its full resolution format. Second, we change the batch size and the depth of the U-net architecture in order to get a better result.

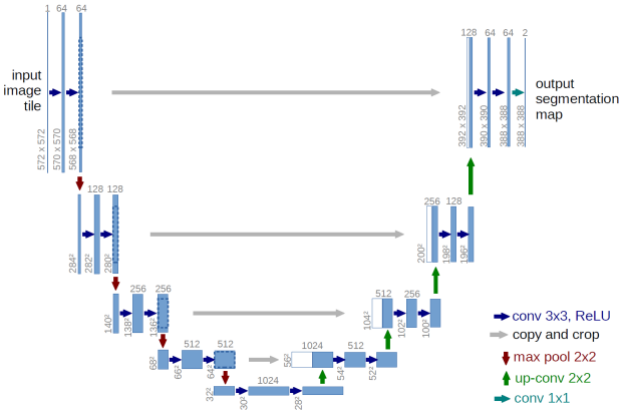

Figure 5: A general "U-shaped" U-net structure, not in our specific case

In our implementation, we have a total of 20 layers network, 10 layers downward and 10 layers upward. This solution works for all situations, ships or no ship or terrain. And it has decent inference accuracy. However, there still exists a major problem — the slow inference speed, which is around 1 ~ 1.5s per image, even on a dedicated GPU from Colab.

3.2 Two-Stage Network

In order to reduce the size of our input and potentially improve our detection speed, we have done a statistical

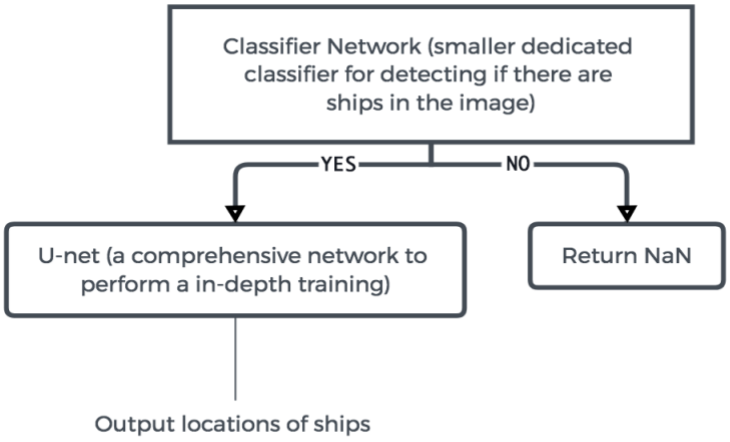

Figure 6: The two-stage inference network structure in our implementation

[2] Ronneberger, Olaf; Fischer, Philipp; Brox, Thomas (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". arXiv:1505.04597 [cs.CV].

[3] Kaggle Kernel. https://www.kaggle.com/frlhell/ship-detection-with-pytorch-unet

analysis on our dataset, shown in Figure 7, and find that around 70% percent of images do not contain ships. Thus, it is not justified to use an expensive network like U-net to locate where the ship is when there is no ship at all in the image.

Therefore, we build a two-stage reference structure (Figure 6) which comprises a convolutional neural network (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; Otherwise, the images will be simply ignored.

The first network is a classifier network (Figure 8), which contains 5 convolution layers and 2 fully connected layers. This network is simple enough that it takes a comparably little amount of time to do the prediction task, yet powerful enough to give us a good performance.

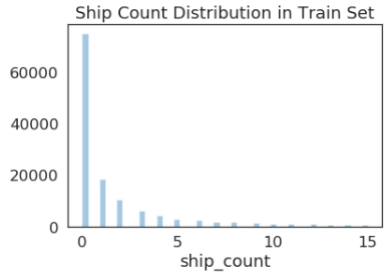

Figure 7: Statistical analysis on the dataset. Number of ships in one image is represented on x-axis; Number of images is represented on y-axis

We will still use U-net as our second network to obtain location information for those images that contain ships. And at this time, a large number of images will be useful inputs for our U-net model to make predictions.

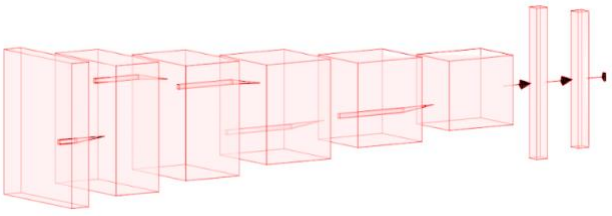

Figure 8: Structure of classifier network

To achieve this hierarchical network structure, we train the classifier network and U-net network independently and save the models after training. Then an inference net will load the models produced by these two networks to output the locations of ships according to this hierarchical structure.

4 Results and Analysis

4.1 Results

During the training for our classifier network, we train on full image resolution for 18 epochs where each epoch contains 1000 images.

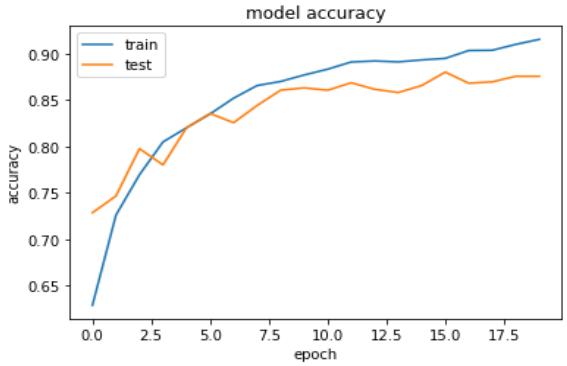

Figure 9: Classifier Network accuracy for train and validation set over epochs

As shown in Figure 9, the training accuracy for classifying whether an image contains ships or not reaches approximately 93% and the validation accuracy reaches approximately 87%. This result from our trained classified model confirms that our simple convolutional neural network can indeed have a good performance on identifying images with and without ships.

For U-net, we train our model on full image resolution with 0.0001 learning rate, 10 layers' depth and 1 batch size for 15 epochs. We use cross-entropy function as our optimization function to calculate the loss with the input: the output produced by our model and the ground truth labels. The output and the labels calculated have the format: [N, H, W], where N stands for the number of batches, H stands for the height of the matrix and W stands for the width of the matrix.

The red plot and the blue plot in Figure 10 show the validation losses and the train losses over iterations respectively. The graph also confirms that the losses are indeed decreasing and converging as the number of trained images increases.

For prediction, we use Intersection of Union (IoU), as our accuracy metric for location detection. IoU is a statistical technique used for measuring the similarity and diversity of sample sets, as shown in Figure 11. In our task, we calculate the area of overlap over that of the union respectively between the location of a ship output by our model and the location of a ship given by the ground truth. The area of overlap divided by the area of union gives us how similar our prediction is comparing to the label.

The two images in Figure 12 show the ground truth mask vs the prediction mask over the same image.

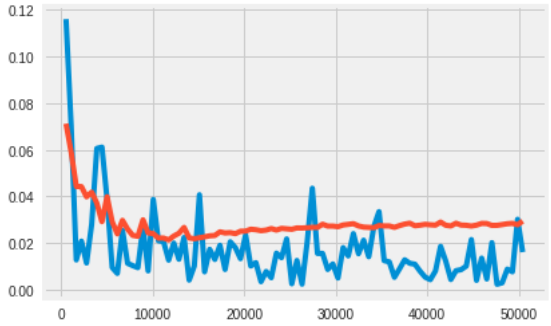

Figure 10: U-net losses over iterations

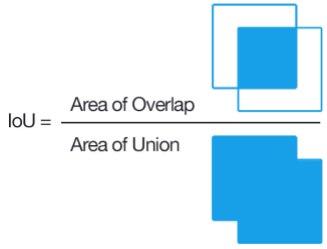

Figure 11: Calculation of Intersection of Union

Figure 12: Ground truth mask over image (Left); Predictions over the same image (Right)

4.2 Error Analysis

We reach 54% IoU accuracy in U-net. Comparing to the vanilla U-net, the inference speed increases by ~28%, where the inference accuracy of the network only drops by ~6%. After some error analysis on the decrease of inference accuracy, we assume that this is because our classifier network sometimes fails to detect false positive cases. That is to say, some images actually contain ships, but our network would classify them as no ship and simply ignore them.

Moving forward to the next stage of our project, we will first focus on addressing this problem. After further observation on our dataset, we find that a great number of images contains large areas of oceans. So, one potential solution might be to write an algorithm as an "ocean detector" or similar ideas that output the probability of containing ships. All in all, we still consider our result as a successful trade-off between prediction speed and accuracy.

5 Contribution

All of the team members have contributed a great amount of efforts to this project both collaboratively and independently. Xinyuan Lu read multiple articles to find a proper model, u-net, and implement helper functions. He also spent much time testing and develop the code. Haonan Xu did early implementation of the network, adapt kernels from kaggle to our own environment, graph and analyze during the progress. Linyin Yang implemented encoder, mask, classifier network and inference net. Together, all three members have worked much or less on every part of the code.

We split the final project report, progress report, and all other workload. Since we didn't have a dedicated GPU and we are poor to buy GPU resource online, we have to use Colab free GPUs. However, Colab GPU will automatically disconnect after 90 minutes or so. We had to manually look at the screen while it is training. On average, we spent ~10 hours per week for ~5 weeks.

6 Future Work

Furthermore, we have begun to do literature research on visual attention model which focus its "attention" only on parts of images that we are interested in. This approach would probably further reduce image processing power required, and thus, improve the speed on inference.

The best practice we believe is to create one neural net v.s. a combination of the two. We would like to work furthermore with Marcelo for on how to build a comprehensive model next quarter.

REFERENCES

[1] https://en.wikipedia.org/wiki/U-Net

[2] Ronneberger, Olaf; Fischer, Philipp; Brox, Thomas (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". arXiv:1505.04597 [cs.CV].

[3] https://www.kaggle.com/frlhell/ship-detection-with-pytorch-unet

[4] Kaggle source https://www.kaggle.com/c/airbus-ship-detection

[5] IOU metrics https://www.kaggle.com/iezepov/fast-iou-scoring-metric-in-pytorch-and-numpy

[6] RLE bounding boxes https://www.kaggle.com/julian3833/2-understanding-and-plotting-rle-bounding-boxes