Locating Ships on the High Seas and in Ports

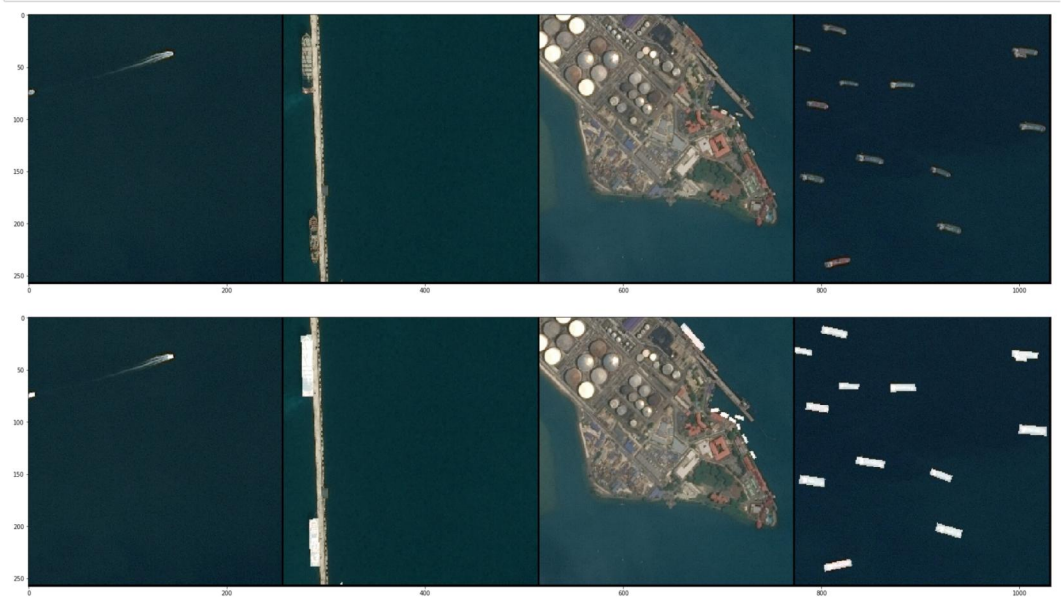
CS230 Project Final Report (Category: Computer Vision)
by Lee Arthurs: larthurs@stanford.edu, December 16, 2018

1 Overview

For our project for CS230, we plan to detect all ships in satellite images as quickly as possible. This is a recent kaggle competition, Airbus Ship Detection Challenge:(`https://www.kaggle.com/c/airbus-ship-detection`). Ship detection has a wide range of real life applications, in the areas of maritime safety, marine traffic and transportation planning, monitoring of marine pollution, illegal fishing, piracy and irregular migration, border control, etc. Our goal is to locate ships in the images and provide segmentation masks of where the ships are located with in the satellite images.

2 Data

The dataset has been provided by the sponsoring company on Kaggle. It is 29 GB and consists of 192,561 images. 150,000, 78%, of the images do not have ships in them. There is a a file which lists the encoded pixel masks for each ship in satellite image. The picture below shows four sample images and then in the bottom row masks added to the images, where the ships are.



We divided the data randomly into 172,561 Training Images, 10,000 Validation Images and 10,000 Test Images.

3 Data Set Challenges

There are a number of challenges specific to our task that we may have to address. Images may have the following features.

- Presence of clouds near the ships or ships underneath haze and other atmospheric phenomena
- Presence of large wake behind the ship which should not be included in the bounding box

- Complexity of other objects in ports and marinas (buoys, barges, wind turbines), coastal elements (tiny islands)
- Sun reflectance on the sea, wave phenomona due to high winds
- Ships that are partially visible in the image, multiple ships coupled together, variation in ship sizes, etc
- Our dataset is unbalanced with almost 78 percent of the images in the dataset have no ships in them at all.

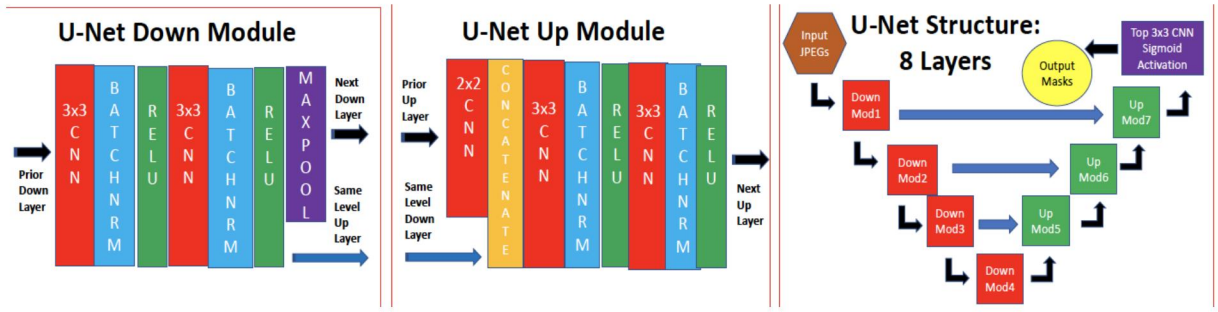
4 Architecture Search

Our task falls under the category of object detection and segmentation. This is an area of active research. Our review initially suggested that Mask R-CNN [5] (for segmentation) and YOLO [9] (for fast object identification/localization) should be our initial focus with objective of using transfer learning. We spent extensive time building transfer learning models with YOLO3, training a range of top layers from 3 to 50. We were not successful of getting any good or useful results with this approach. As mentioned later in Section 12, Contributions/Extenuating Circumstances, my original partner, Suren Talla, was building models using Mask R-CNN, but due to significant family issues that work was not completed and this became a solo project for just me instead of the original group project.

After not finding success with YOLO3, we examined in more detail the successes in medical scans and biology. The paper by Zhang et all [12], examined different structures to perform segmentation and prediction of lunch cancer tumor regions. Their results found that the U-Net structure worked best for their data, which seemed in many ways similar to our data.

5 U-Net Structure

Here we briefly describe the U-Net structure. The key difference between the U-Net structure and more vanilla CNNs is that information both flows through the network and also flows between different levels. This is best described visually in this attached chart. The U-Net is composed of two different modules: a down module and an up module. The structure of these modules here is our final best structure and you can see the different components. The key to note on the down module is that information flows out of the module from two different spots and flows to two different places. The Maxpool output goes to the next module in the chain. The RELU output (skipping Maxpool) goes across the model to the same "height" up module. You can see this on the far right showing an U-Net eight level structure. In the same way that the down module outputs information at two spots the up module receives information at two spots as you can see in the picture. The up module uses Concatenate to combine this information after the 2x2 CNN. The final top layer is a 3x3 CNN followed by a Sigmoid activation to output the projected mask.



6 Evaluation and Loss Functions

We decided to use the dice coefficient calculated using the actual masks and the projected masks to evaluate or model results. This is commonly used in tasks that our generating masks. Values range from 0 to 1, with 1 representing the images being the same. The closer to 1 the better the result. The formula is as follows:

$$Dice\ Coefficient = \frac{2\sum((P_i * T_i) + 1}{\sum P_i + \sum T_i + 1} \tag{1}$$

In this formula, P_i is the projected mask from the model for each satellite image and T_i represents the actual provided true mask for each satellite image. The plus one on the top and the bottom is a smoothing value. The loss function used in the U-Net structure was the negative of the Dice Coefficient.

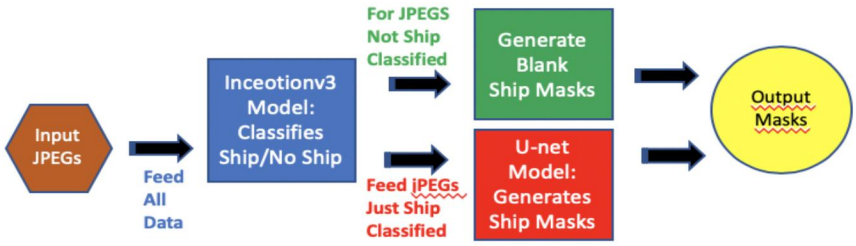
7 U-Net Structural Choices and Tuning

Since we were training the U-Net from scratch (not through transfer learning) we reduced the size of the input images and masks from (768,768,3) and (768.768,1) to (96,96,3) and (96,96,3) respectively using numpy functions to make the model and data more manageable. We also normalized the input satellite images using the mean and variance of the training data set. We experimented initially training the model using smaller data to see what size U-Net and structure would give the best results. We found increasing the size of the U-Net from 8 layers to 10 layers by adding an additional Down Module and Up Module actually slightly reduced the results. We found a significant benefit from adding Batch Normalization to the model in the spots shown in a module descriptions above. We could both increase the learning rate and achieve significantly better results. The original U-Net structures which found in the papers and Github did not have Batch Normalization.

Given the significant amount of images with no ships, we experimented with training the model just using Train data and Validation data with ships which reduced the train data to from 172,561 to 38,049 images and the validation to 2,193 images, which made training time for running the model reasonable. After experimenting with a range of training rates from 1e-3 to 1e-5 we were our best results with the model structure in above chart with a Dice Coefficient of .972 on the training data and .952 on the validation data (both only on the images with ships) using our best learning rate of 1e-4 after 150 epochs. These results of course do not reflect our final results because they do not include the images with out ships.

8 Add Another Model Layer: Inceptionv3

The next was to an image classification layer to just before the U-Net to determine whether an image contained any ships or not. We trained the top layer of an Inceptionv3 model for this from Tensorflow Hub, `(https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1)`.



As can see from the chart the model will now first use Inceptionv3 to classify images as either Ship(s) or No Ship. The images which are classified as no ship will automatically be outputting empty masks and those

images, which are classified as having ships will then be fed into the U-Net model and the outputs from U-Net model will be the final results.

The Inceptionv3 was trained using with a variety of learning rates as you can see in the chart the best Validation results were for a learning rate .005 at a 94.1% rate. 63% of the incorrectly categorized images in the Validation set were images with no ships that were categorized as having ships.

InceptionV3 Results by Learning Rate

LR	Train %	Val %	Test %
.002	93.7	93.6%	93.5 %
.005	94.2%	***94.1%***	93.8%
.00875	94.1%	94.0%	93.8%
.01	94.1%	93.9%	93.9%
.02	93.8%	93.6%	93.6Z%
.05	93.7%	93.3%	94.3%

9 Putting the Models Together

Next we look at the results combining the best case U-Net and the best case InceptionV3. As can be seen in this chart, combining the InceptionV3 and the U-Net together has a dice coefficient of .953 on running all the test data. Interestingly if we run the U-net which was just trained on the no ship images on the full test data we get a dice coefficient of .943 so the InceptionV3 is improving our results by only .01. I also ran a hypothetical scenario as to what our results would be if the Classification Model has been perfect and our results would have been .961. This suggests that efforts to improve results are better spent on improving the U-Net than the classification stage.

Final Results

Model	TEST Data Dice Coefficient
Combined Inceptionv3 and U-Net	.953
Just U-Net	.943
U-Net with Perfect Classification Model	.961

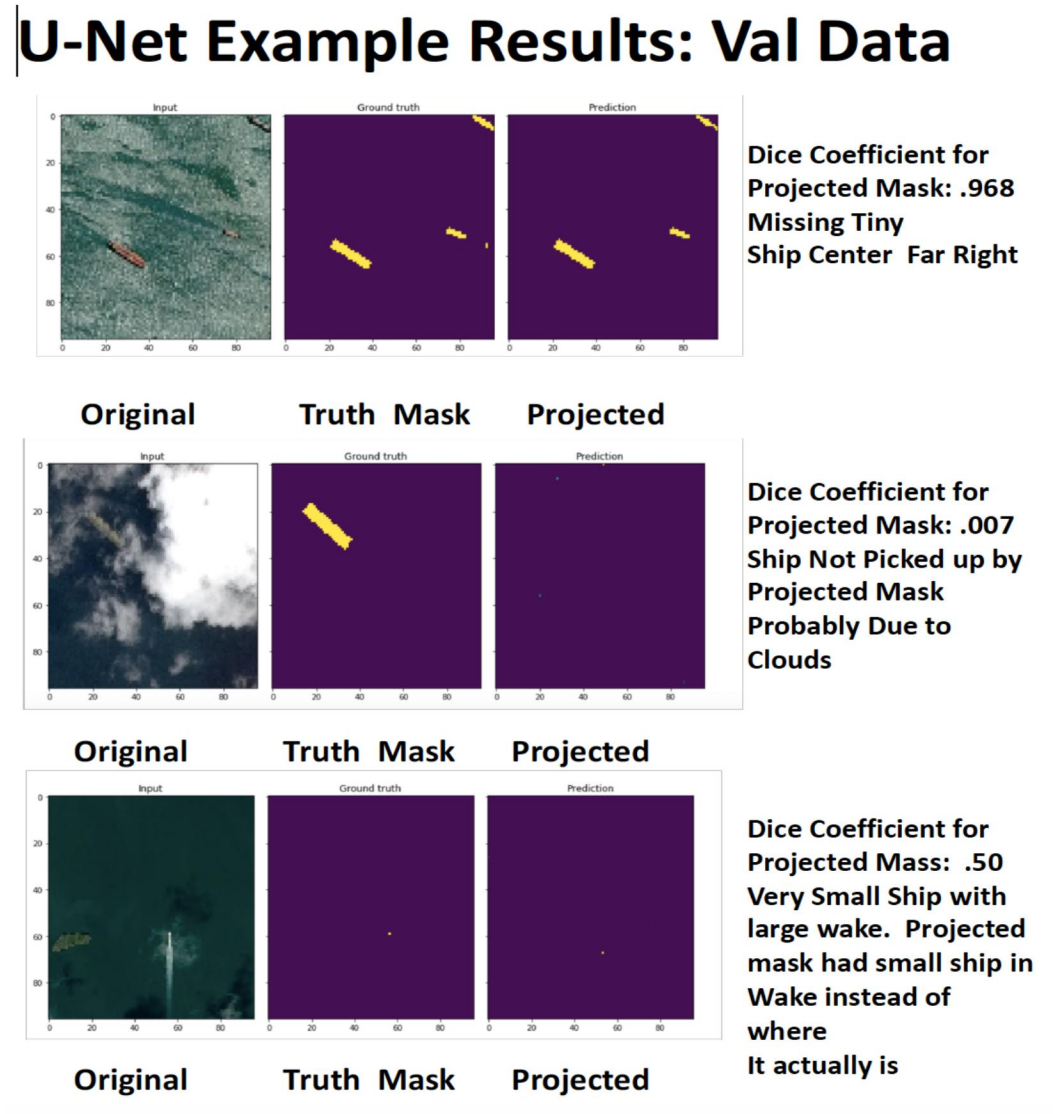
10 Code

My code is in the GitHub `https://github.com/lartthurs/CS230FInalProj`. The code was started using github code, which is noted in the Jupyter Notebooks. We then modified the code was then modified for the specific models to be run and also to organize the data to work well with this specific data.

11 Next Steps

Shown below are three examples of images ith different types of misclassifications and their resulting Dice Coefficients. The first one is missing one very small ship but does a good job with the other four. The second one totally misses the ship because of the clouds and the third one confuses the wake of the ship with the ship predicting the ship in the small ship in the wrong location.

As we mentioned in the prior section the potential benefits from improving the U-Net results is greater than than the benefits from improving the ship no ship classifier. A logical next step is data augmentation, which could help the U-Net especially for cases such as the clouds shown here. Smaller ships identification might benefit using larger image sizes in the model than the (96,96,3) and (96,96,1) which we used to train the model. This would require more processing power and time for running scenarios.



12 Contributions/Extenuating Circumstances

I had originally partnered with Suren Talla, another SCPD student in CS230. We worked together through the Milestone. Since the Milestone he had significant family issues and ultimately had to drop from the project. He contributed in choosing this project, the initial research and ideas discussed in the Milestone. All the modeling done here is all my work and has been done since the Milestone. Some of the writing here was copied from either our Proposal or Milestone so that writing was the combined work of Suren and me. All other writing is my work. When I use the term "we" in this report it refers to just me except in the cases noted here for the work before the Milestone.

A Appendices

A.1 Opensource detection and segmentation architectures: reviewed and noted in Jupyter Notebooks in Github where used as starting code

InceptionV3 TensorFLow Hub - https://tfhub.dev/google/imagenet/inception_v3/feature_vector/1
Tensorflow Hub Example - https://raw.githubusercontent.com/tensorflow/hub/master/examples/image_retrain
U-Net - https://github.com/jakeret/tf_unet
U-Net - https://github.com/OverFlow7/Ultrasound-Nerve-Segmentation
YOLO (darknet) - https://pjreddie.com/darknet/yolov1/ (C++)
YOLO v2 (darknet) - https://pjreddie.com/darknet/yolov2/ (C++)
YOLO v3 (darknet) - https://pjreddie.com/darknet/yolo/ (C++)
YOLO (tensorflow) - https://github.com/thtrieu/darkflow
Mask-RCNN - https://github.com/matterport/Mask_RCNN (Only used by Suren)
Model Zoo - https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):24812495, Dec 2017.

[2] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:834–848, 2018.

[3] Sergey Ioffe Jonathon Shlens Zbigniew Wojna Christian Szegedy, Vincent Vanhoucke. Rethinking the inception architecture for computer vision. 2015.

[4] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[6] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and et al. Speed/accuracy trade-offs for modern convolutional object detectors. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[7] Auerlien Lucchi Alexandre Refregier Joel Akeret, Chilhway Chang. Radio frequency interference mitigation using deep convolutional neural networks. 2017.

[8] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[9] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[10] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[12] Shin-Cheung Lai Jun Xiao Runze Zhang, Zhong Guan and Kim-Man Lam. Deep neural networks for lung cancer tumor region segmentation.