Identifying ships from Space Borne Photography

Definition

Project Overview

Space Borne Photography (Satellite Imagery) provides various aspects including agriculture, meteorology, energy, and cartography. New commercial imagery providers, such as Planet are using of small satellites to capture Earth images every day.

This flood of new imagery is outgrowing the ability for organizations to manually look at each image that gets captured, and there is a need for machine learning and computer vision algorithms to help automate the analysis process.

The aim of this project is to address the difficult task of detecting the location of large ships in satellite images. Automating this process can be applied to many issues including monitoring port activity levels and supply chain analysis.

Satellite imagery used to build this dataset is made available through Planet's Open California dataset, which is openly licensed. As such, this dataset is also available under the same CC-BY-SA license. Users can sign up for a free Planet account to search, view, and download their imagery and gain access to their API.

Problem Statement

The goal is to create a model which takes various pieces of a satellite image and trains on finding a ship in it; the tasks involved are the following:

1. Download the data from kaggle website
2. Pre-process the data
3. Train a cnn model on the image data
4. Finally evaluate the capacity of model.

Metrics

Accuracy is a common metric for binary classifiers;

Accuracy = (images correctly classified) / (total no. of images)

With addition to accuracy we also calculate precision and recall as metrics

Precision = True positives / (True Positives + False Positives)

Recall = True positives / (True Positives + False Negatives)

In here I would mainly consider the f1-score as crucial metric as the dataset contains unbalanced data of 1000 belonging to one class and 3000 belonging to other. In order to calculate f1-score we have to calculate precision and recall.

During development, a validation set was used to evaluate the model. I want to use a small set of training images as my validation images. For validation I want to use "categorical_crossentropy" as loss metric for CNN, optimizer as 'SGD' and 'ADAM'.

Analysis

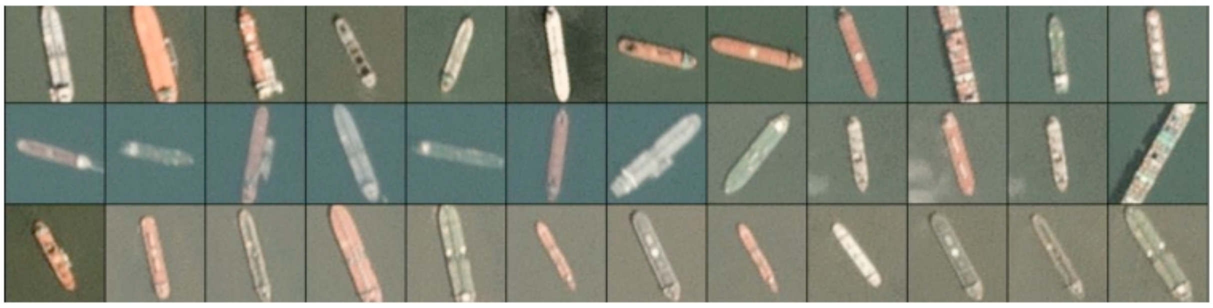
Data Exploration

The dataset consists of image chips extracted from Planet satellite imagery collected over the San Francisco Bay and San Pedro Bay areas of California. It includes 4000 80x80 RGB images labelled with either a "ship" or "no-ship" classification. Image chips were derived from Planet Scope full-frame visual scene products, which are orthorectified to a 3 meter pixel size.

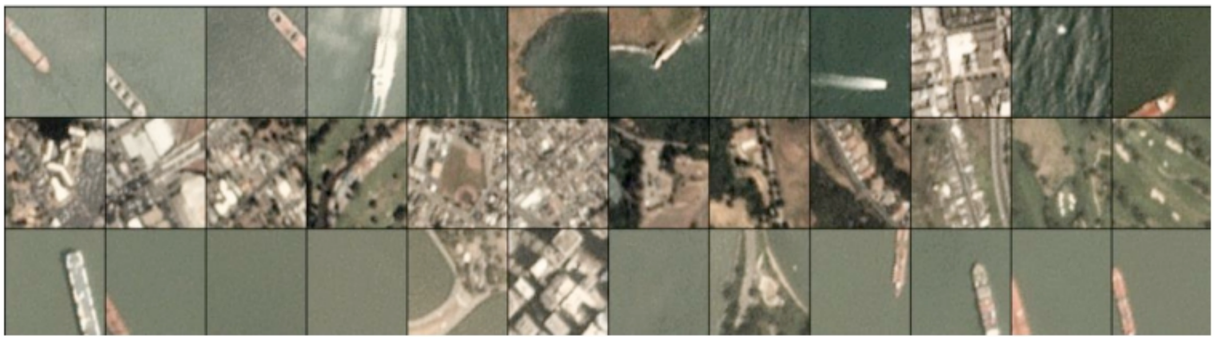
The dataset is also distributed as a JSON formatted text file shipsnet.json. The loaded object contains **data**, **label**, **scene_ids**, and **location** lists.

The pixel value data for each 80x80 RGB image is stored as a list of 19200 integers within the **data** list. The first 6400 entries contain the red channel values, the next 6400 the green, and the final 6400 the blue. The image is stored in row-major order, so that the first 80 entries of the array are the red channel values of the first row of the image.

The dataset is distributed as a JSON formatted text file. The loaded object contains data, label, scene_ids, and location lists.The "ship" class includes 1000 images. Images in this class are near-centered on the body of a single ship. Ships of different sizes, orientations, and atmospheric collection conditions are included. Example images from this class are shown below.



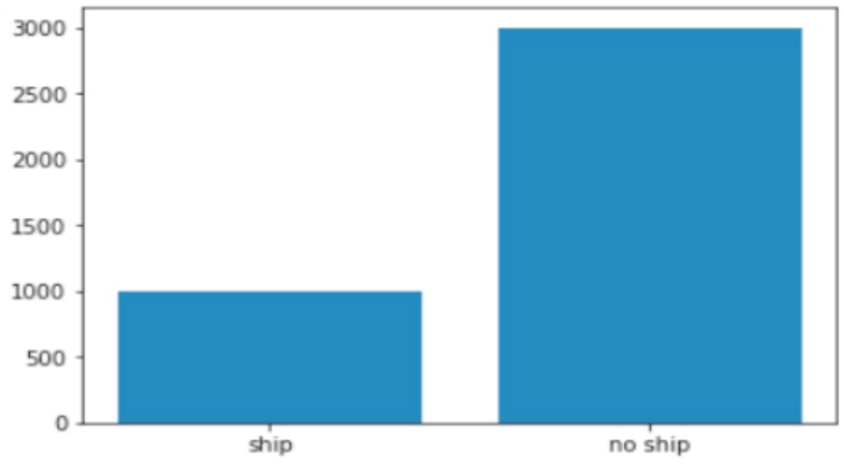
The "no-ship" class includes 3000 images. A third of these are a random sampling of different landcover features - water, vegetion, bare earth, buildings, etc. - that do not include any portion of an ship. The next third are "partial ships" that contain only a portion of an ship, but not enough to meet the full definition of the "ship" class. The last third are images that have previously been mislabeled by machine learning models, typically caused by bright pixels or strong linear features. Example images from this class are shown below.



Exploratory Visualization

There are 2 categories
There are 4000 total images
Total training images are 3240
Total testing images are 400
Total validation images are 360

Fig. 1: This figure shows the distribution of image in classes.



Algorithms and Techniques

The classifier is a Convolutional Neural Network, which is the state-of-the-art algorithm for most image processing tasks, including classification. It needs a large amount of training data compared to other approaches. The algorithm outputs an assigned probability for each class; this can be used to reduce the number of false positives using a **threshold**. (The trade-off is that this increases the number of false negatives.) The following parameters can be tuned to optimize the classifier:

- ❖ Training parameters
  - ➢ Training length (number of epochs)
  - ➢ Batch size (how many images to look at once during a single training step)
  - ➢ Solver type (what algorithm to use for learning)
  - ➢ Learning rate (how fast to learn; this can be dynamic)
  - ➢ Weight decay (prevents the model being dominated by a few "neurons")

        - ➢ Momentum (takes the previous learning step into account when calculating the next one)
- ❖ Neural network architecture
    - ➢ Number of layers
    - ➢ Layer types (convolutional, fully-connected, or pooling)
    - ➢ Layer parameters
- ❖ Pre-processing parameters

During training, both the training and the validation sets are loaded into the RAM. After that, random batches are selected to be loaded into the GPU memory for processing. The training is done using the Adam algorithm.

Theory behind Scenes:-

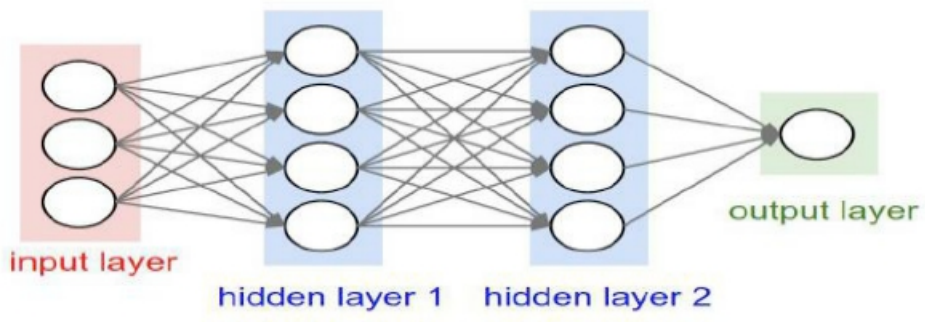
Artificial neural networks (ANNs): ANNs are computing systems vaguely inspired by the biological Neural Networks that constitute animal brains and humans. These systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

A typical brain contains something like 100 billion miniscule cells called neurons (no-one knows exactly how many there are and estimates go from about 50 billion to as many as 500 billion).

Each neuron is made up of a cell body (the central mass of the cell) with a number of connections coming off it: numerous dendrites (the cell's inputs—carrying information toward the cell body) and a single axon (the cell's output—carrying information away). Neurons are so tiny that you could pack about 100 of their cell bodies into a single millimetre. Inside a computer, the equivalent to a brain cell is a tiny switching device called a transistor. The latest, cutting-edge microprocessors (single-chip computers) contain over 2 billion transistors; even a basic microprocessor has about 50 million transistors, all packed onto an integrated circuit just 25mm square (smaller than a postage stamp)!

ANN is a set of connected neurons organized in layers:

- **Input layer**: brings the initial data into the system for further processing by subsequent layers of artificial neurons. Like dendrites in human neuron
- **Hidden layer**: a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function. Like nucleus in human neuron.
- **Output layer**: the last layer of neurons that produces given outputs for the program. Like axon in human neuron



Typical architecture of a neural network

Working Procedure

Step 1- Model initialization: -

A random initialization of the model is a common practice. The rationale behind is that from wherever we start, if we are perseverant enough and through an iterative learning process, we can reach the pseudo-ideal model.

Step 2- Forward propagate:-

The natural step to do after initializing the model at random, is to check its performance.
We start from the input we have, we pass them through the network layer and calculate the actual output of the model straightforwardly.

Step 3- Loss function:-

At this stage, in one hand, we have the actual output of the randomly initialized neural network.
On the other hand, we have the desired output we would like the network to learn. Then we define what we call: loss function (for intuition just think loss function like absolute difference or squared difference, but it changes with model to model). Basically, it is a performance metric on how well the NN manages to reach its goal of generating outputs as close as possible to the desired values.

Step 4- Differentiation:-

We can use any optimization technique that modifies the internal weights of neural networks in order to minimize the total loss function that we previously defined.

Step 5- Back-propagation:-

Then error in loss function is propagated from output layer to input layer by using differentiation we solved in step-4

Step 6- Weight update:-

Then corresponding weights in the network are changed in order with learning rate.
New weight = old weight — Derivative Rate * learning rate

Step 7- Iterate until convergence:-

Just iterate the procedure from step2 to step6 until convergence

In here the convergence depends on different factors:

- Depends on learning rate
- On optimization method
- Different meta-parameters

The following parameters can be tuned to optimize the model:

- Number of epochs
- Batch size
- Number of layers
- Different layers like convolutional, fully-connected or pooling etc,.

Benchmark

Bench mark model: Any CNN model that gives accuracy around 26% is my benchmark model. This model contains basic model of a cnn with minimum possible layers and so that this would fit for our benchmark.
My created model
Conv2D layer with 2 filters, kernel_size is equal to 3, and input shape is (80, 80, 3).
Max Pooling layer with pooling size is equal to 2.
Flatten layer and Dense layers with 2 units and activation of "SoftMax".
At compilation phase loss='categorical_crossentropy', optimizer='SGD', metrics= ['accuracy'] are used.
Check pointer is used with 'weights.best.bm_model.hdf5', as file path and save_best_only is tuned to True
Bench model is trained on fit_generator method in order to achieve the data augmentation with 32 as batch size and 110 steps per epoch and number epochs are 1.

Architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 78, 78, 2)         56        
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 39, 39, 2)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 3042)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 6086      
=================================================================
Total params: 6,142
Trainable params: 6,142
Non-trainable params: 0
_________________________________________________________________
```

Methodology

Data Pre-processing

The pre-processing done in the “Prepare data” notebook consists of the following steps:

1. The list of numerical data should be converted into images and randomized.
2. The images are divided into a training set and a validation set
3. The images are resized into 3D tensor with shape 80x80 with 3 channels (RGB).
4. All images are normalized by dividing by 255.
5. All labels are converted to 2 categories by using one-hot encoding.

Implementation

As in implementation model I have created model using different layers and the architecture is as below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 80, 80, 32)        896       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 80, 80, 32)        9248      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 40, 40, 32)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 40, 40, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 40, 40, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 40, 40, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 20, 20, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 20, 20, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 20, 20, 128)       663680
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 20, 20, 128)       1327232
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 10, 10, 128)       0
_________________________________________________________________
global_average_pooling2d_1   ((None, 128)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               66048
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 1026
=================================================================
Total params: 2,123,554
Trainable params: 2,123,554
Non-trainable params: 0
_________________________________________________________________
```

Steps involved in implementation:

1. Initially I thought of creating a model with three layers each layer containing a convolution 2d layer followed by a max pooling layer.
2. And then a flatten layer at the end of the three layers followed by dense layers.
3. This model yielded very good training scores but this model classifies many no ships as ships.
4. So I thought of creating a little bit complicated model containing same three layers but this time each layer contains two convolution layers followed by max pooling layers and then a drop out layer at the end. This model of mine was inspired by VGG16 and implemented a portion of it.And in this model I have added drop out layers in order to overcome overfitting.
5. In place of flatten which creates large amount of parameters I took global average pooling layers which reduced the parameters by a million.
6. This new model took very less time for each epoch of just 8 secs. And also the scores of this model were overwhelming.
7. During evaluation of this model, I became very confident that there is no need in further extension of this model, as the scores of this model were very high.
8. I came to a conclusion that there is no need for taking another new model for this dataset.
9. The most challenging part of this model is to find the right layers and positioning the layers in right order. And also the number of filters in each layer also plays a very important role as the number increases the parameters increases and the model takes lot more time. The replacing of GAP layer instead of flatten really helped the model in reducing the parameters and also in performance.

Compilation phase:

Trained model has compiled using “ADAM” as optimizer and 'categorical_crossentropy' as loss function, used

Metrics is “accuracy”.
Check pointer is used with 'weights.best.f_model.hdf5', as file path and save_best_only is tuned to True
Model is trained by using fit method with parameters 20 as epochs 32 as batch size
While 90% of training data is using for training purpose remaining 10% of data is used for validation purposes.
And used data augmentation.

Results

Model Evaluation and Validation

	Precision	recall	f1-score	support
No Ship	0.99	0.99	0.99	293
Ship	0.98	0.96	0.97	107
Micro avg	0.98	0.98	0.98	400
Macro avg	0.98	0.98	0.98	400
Weighted avg	0.98	0.98	0.98	400

Classification report for model to evaluate results:-

As the classification is binary we got very good results almost 1. We can see that the model have 0.99 f score for no ship class and 0.97 for ship class. This is very high score which makes our model a really good one. And also the test accuracy of this model is 97% which makes the model really trustworthy. From the scores we can conclude that the model is robust to unseen data and is able to generalize the data pretty well.

This model exceeds our benchmark model and also outperforms it. And also we can absolutely trust this model for future improvisations.

Justification

When compared with my benchmark model, my model gives training accuracy of 97%.

Whereas my testing accuracy is around 98%.

The results obtained from my model above satisfactory as both training and testing have accuracy scores are above 97% which is pretty good.

I have chosen this model because I have got very decent score in accuracy which is about 97% and also the f score for this model is very high and about 0.99 for detecting there is no ship and 0.97 for detecting a ship.

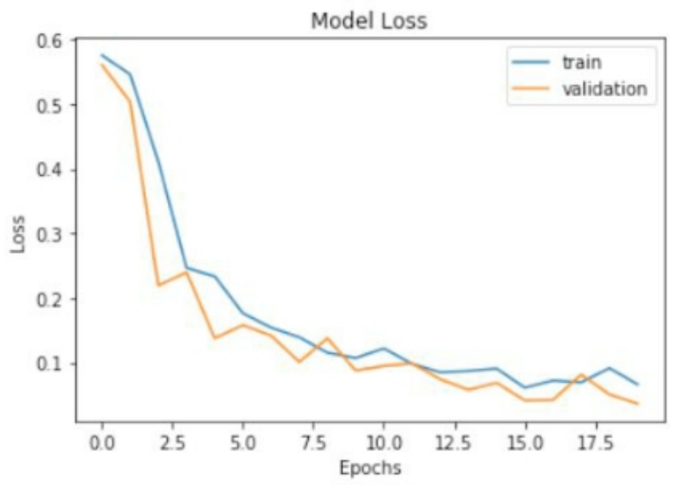
And also I have observed that from 16 epoch the model weights didn’t improve which made me realise that the performance of the model reached an optimum value.

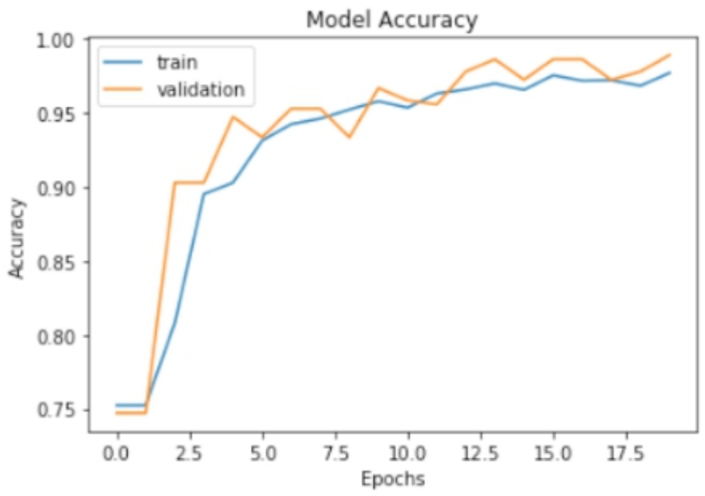
This clearly made me so confident that this model will definitely generalizes well on unseen data.

Conclusion

Free-Form Visualization

Fig. 3: Loss and accuracy graphs of the model

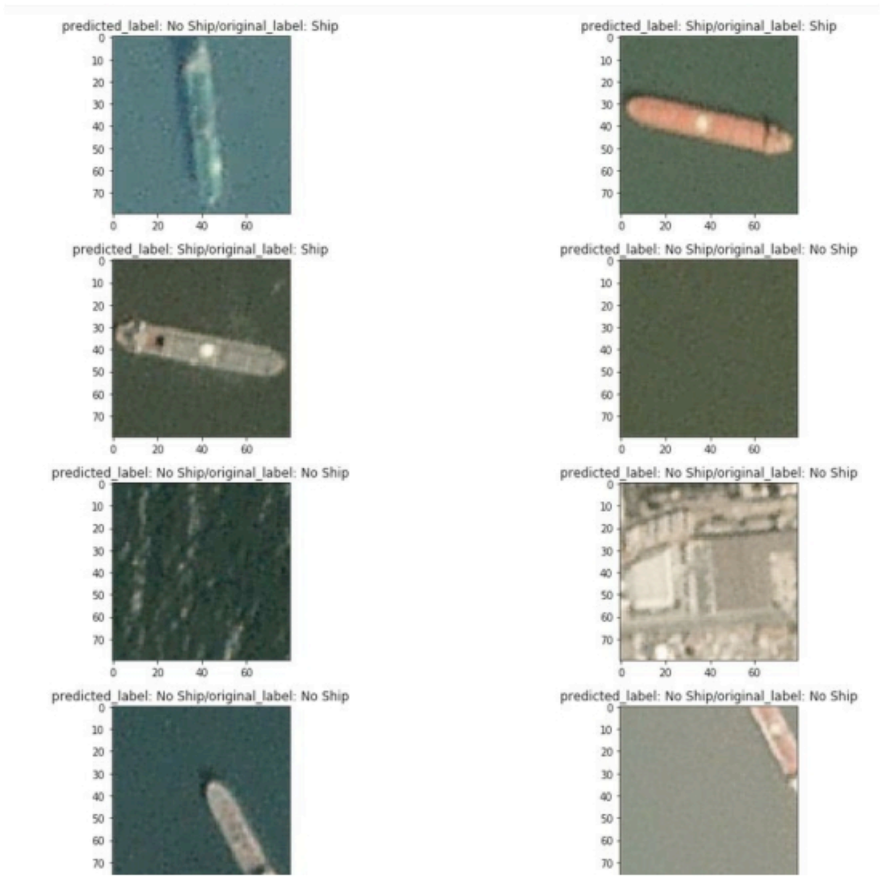




From Figure **3**:

We can see that the accuracy of the model constantly increasing with increase in epochs. And also we can see the model loss is gradually decreasing and almost reaches zero.

Fig. 4: The below figure shows the predicted and original labels of the images by the model.


predicted_label: No Ship/original_label: Ship
predicted_label: Ship/original_label: Ship
predicted_label: Ship/original_label: Ship
predicted_label: No Ship/original_label: No Ship
predicted_label: No Ship/original_label: No Ship
predicted_label: No Ship/original_label: No Ship
predicted_label: No Ship/original_label: No Ship
predicted_label: No Ship/original_label: No Ship

From the above figure we can clearly see that our model is very good at predicting the data.

Reflection

The process used for this project can be summarized using the following steps:

1. First thing is I was captivated by the working of CNN, the scope of the neural networks really amazed me. So I researched about the implementation of CNN in different fields. Finally I came across a dataset in kaggle.
2. I have learnt different concepts in this journey. Most of all I clearly understood the process of implementing a convolution neural network on a real world problem.
3. I have also learnt how to use kaggle kernels.
4. Every work from the beginning of this project was challenging for me, as this is my first project in full scale.
5. The data I was given in json format which is very strange to me. I done a research on it on how to access the data in the file.
6. And then the image data in the file is in numbers which lead me to pursue on how to change the data into image.
7. I have to then resize all the images into 80x80 pixels.
8. And also I have learnt the usefulness of the normalization of images which plays a very crucial role in the performance of the model.
9. I have understood the need of data augmentation as the data I have is very small this really helped me in creating a best model.
10. After all these I created training, testing and validation sets from the given data.
11. I have created a benchmark model with least possible neural network.

12. Here comes the heart of the project, creating a CNN model, fitting it to training and testing on test data evaluating validation curves, learn from confusion matrix doing modifications on models.
13. And finally visualizing results really helped me in evaluating the robustness of the model.

Improvement

1. We have limited amount of data, so adding more data will certainly benefit the model.
2. We can use this model for not only finding the ships but also on planes and other vehicles.
3. The image data in here is only subjected to large ships in 80x80 image pixel. All other small ships are not considered. So I would like to add those also which makes a little bit challenging.