Object Detection and Classification of Ships in Satellite Images

Daniel Kim
CSULB
SeUn.Kim@student.csulb.edu

Luis Gonzalez
CSULB
Luis.Gonzalez08@student.csulb.edu

Visal Hok
CSULB
Visal.Hok@student.csulb.edu

***Abstract*—The goal for this project was to correctly detect and classify ships in large satellite images using a Convolutional Neural Network (CNN) that was trained on 80 by 80 images of ship and non-ship data in conjunction with a custom object detection algorithm. Our approach is able to efficiently and accurately detect ships in satellite images of varying size ratios. In comparison to an approach found on Kaggle.com that uses a sliding window across the entire image to detect the ships within 15 minutes, our project pipeline is able to detect and output bounding boxes in under 5 seconds. The code for our project has been made available at: https://github.com/PenguinDan/DeepLearningShipDetection**

***Index Terms*—Convolutional Neural Network, Object Detection, OpenCV**

I. Introduction

Although Object Detection and Classification models such as Mask R-CNN and YOLO exist, they are limited by the fact that the model must either already have been trained on images similar to our inputs, or introduced to a large set of labeled images for fine tuning. However, our dataset only includes 8 large non-labeled satellite images with varying size ratios where most of the image is considered to be background. Most of the object detection and classification models that exist today have been trained on large images where the object is clearly visible and appear to be large. The objects that we are trying to detect and classify in our images are each less than 1 percent of the entire image. This complication has forced us to design and create our own project pipeline in order to solve the task of detecting and classifying ship imagery in large satellite images.

II. The Dataset

The dataset retrieved from Kaggle contains a total of 2800 80 by 80 RGB images that are labeled with either a "ship" or "not-a-ship" classification. There are 700 images classified as "ship" with varying ship sizes, orientations, and atmospheric conditions. The other 2100 images include random samplings of landcover features such as water and vegetation, partial images of ships, and other images that have been previously mislabeled as ships by machine learning models. As for the large satellite data, we are only given 8 non-labeled images near coasts that include both landmass and ships. This limitation of large satellite images was the main inspiration behind the structure of the project pipeline.

III. Hardware

TABLE I
Hardware Used

Component	Name
CPU	Intel i7 4790k
GPU	NVIDIA GTX 1080
Memory	16GB ddr3

IV. Related Works

Fig. 1. Attempt by top Kaggle contributor Vitaly Burachonok

The current model as introduced by Vitaly Burachonok in Kaggle as the top contributor is very inefficient and inaccurate as one can see in Fig. 1. This model trains a CNN on small 80 by 80 images but uses a rather slow sliding window technique to detect and classify the objects in the larger satellite image. Using the hardware mentioned on Table 1, this model took a total of 876 seconds to create bounding boxes around the items in which it believes to be ships with multiple false positives.

V. The Project Pipeline

A. Image Preprocessing Module

Each large satellite image must be preprocessed before they are used as an input into the subsequent object detection algorithm. The main goal behind the preprocessing step is to enhance the accuracy and speed up the performance of the task of detecting and classifying ship imagery. This step helps the object detection algorithm to confidently output less region proposals while also minimizing the amount of classifications ran in the object classification network.

B. Object Detection Algorithm

The object detection algorithm searches through a preprocessed image and outputs a list of bounding boxes in which it believes to contain an object. These bounding boxes are then scaled to a perfectly square size while trying to keep the objects in the center of the box. This is to avoid losing any dimensional information when they are being scaled down to be used as inputs to the CNN.

C. Convolutional Neural Network

The CNN receives a cropped list of 80 by 80 images extracted from the original large satellite imagery. Those images are then used as inputs and classified as whether they are a "ship" or "not-a-ship". The bounding boxes around the images that are classified as "not-a-ship" are removed while the bounding boxes around "ship" images are kept in the original satellite image.

VI. Image Preprocessing

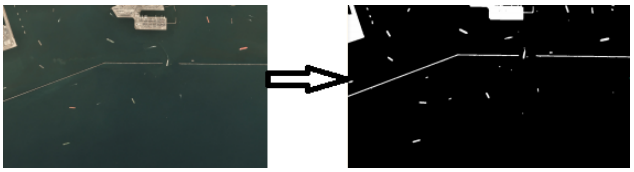

Fig. 2. Image Preprocessing Example.

Each large satellite image must be preprocessed before they are used as an input into the subsequent object detection algorithm. Images are grayscaled and input into a function that incorporates a grayscale threshold value to turn each pixel in the image into a value of 0 or 255. If the value equals or exceeds a given threshold, it's pixel value is scaled up to 255, otherwise it is set to 0. OpenCV, a python computer vision library, is then utilized to denoise and enhance the image by closing small holes inside foreground objects using a series of dilating and eroding operations alongside multiple square kernel sizes. The image is then normalized by dividing all of the pixel values by 255 so that the image matrix is now a set of 0 and 1 values.

VII. Object Detection

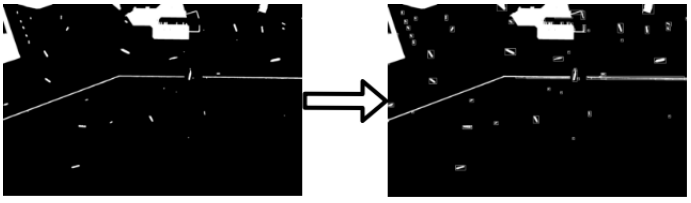

Fig. 3. Image Object Detection Example.

A. Methodology

Due to the nonexistence of a reliable object detection and region proposal algorithm on satellite imagery, we were inspired to create our own.

The algorithm begins by first receiving a normalized image matrix and sliding a 100 by 100 window with 100 stride across the image. At each iteration, the total weight value of the window is calculated to see if it surpasses an activation threshold of 3000. If the weight is greater than or equal to the threshold, the sum of each row within the 100 by 100 window is then calculated to determine the location of a pixel to begin our depth first search algorithm, otherwise, it ignores the location and continues through its iterations.

Our modified depth first search algorithm only strives to find the edge pixels rather than the entirety of the object. Once it determines that it has properly outlined the object, the algorithm then logs the starting (x,y) and ending x coordinates. Next, the coordinates are used to find the total weight of the item within the created box and compared to an object threshold value of 8000 to ensure that it is not part of the main land. If the weight is less than or equal to the object threshold, the coordinates are saved, and if it is greater, the coordinates are thrown away. After either scenario, the area within the coordinates is blacked out to prevent a redundant procedure.

Finally, the list of all of the saved coordinates are then scaled and used to crop the image proposals from the original unpreprocessed satellite image and input in the CNN classifier.

B. Speed Comparison

Fig. 4. Kaggle Detection vs Our Object Detection Example.

One of the biggest advantage with developing our own object detection and region proposal algorithm is that we were able to dramatically increased the speed and accuracy of the ship detection. As one can see from Fig.4, not only is our algorithm more accurate in terms of detecting the proper location of ships with less false positives, it is able to accomplish this feat in under 5 seconds.

C. Algorithm

```
Initialize all variable

MAXXAXIS = imageMatrix[y-Axis]
MAXYAXIS = imageMatrix[x-Axis]

while startY < MAXYAXIS
        find endY
    while(starX < MAXYAXIS)
        find endX
        find weight
        if(weight < activationThreshhold)
                Add findObjectList to ObjectProposalList
        startX = endX
    startX = 0
    endx = 0
    startY = endY

return ObjectProposalList
```

Fig. 5. Pseudo Code

VIII. CNN Architecture and Model Training

A. Training Parameters

The optimizer used to train all the models was Stochastic Gradient Descent (SGD). Unless otherwise specified, the parameters for SGD were a learning rate of 0.001, a decay rate of 0.9, and a momentum value of $1 \cdot 10^{-6}$. The models were all trained on batches of size 50. All this was coded using Keras with Tensorflow as the backend.

B. Architecture

The CNN architecture used is based on VGG-16. The original VGG-16 architecture could not be used because the training image size is only 80 by 80 pixels while there are 5 pooling layers in the original architecture. This presents an issue because 80 cannot only be halved 4 times total. As a result, one of the pooling layers must be taken out. The final classification function also needed to be changed to a sigmoid function because this is a binary classification problem.

5-fold validation was used to test all five possible cases. The first case was where the first pooling layer was deleted, second case deleted the second pooling layer, etc. The models were trained on 100 epochs for every fold. The results, display below in Table II, show that the optimal pooling layer to delete is the first poling layer.

The model, using the optimal architecture from Table II, was then trained on all 2500 training images with 100 epochs. This trained model was used to classify the proposed objects from our object detection algorithm on a satellite image. The objects that were classified as "ships" by the model were then boxes and the result is shown in Fig. 6.

In this test case only seven of the ten ships were correctly classified as a "ship". Moreover, there is a portion of the image which was misclassified as a "ship" (false-positives).

TABLE II
Mean Loss and Accuracy Across All 5-Folds

Test Case	Average Loss	Average Accuracy
1	0.1206	98.20%
2	0.1562	97.84%
3	0.1493	97.80%
4	0.1233	98.12%
5	0.1269	98.16%

[a]Sample of a Table footnote.

Fig. 6. Test for first trained model.

This model could not be improved through further training with the limited data as the training already plateaued.

C. Data Augmentation

As the current problem seemed to be the limited data size, it was increased through data augmentation. The data was augmented using Keras ImageDataGenerator class. The generator was initialized to include the use of a rotational range of 45 degrees, width shift range of 0.2, height shift range of 0.2, and zoom range of 0.2 and includes both horizontal and vertical flips of the images. This generator augments the images in batches, which was initialized to 50, as the model is being trained.

The first model trained using this data augmenter was only trained on 100 batches per epoch so the model was trained on 5,000 images per epoch; moreover, the data is augmented on the fly in every batch so it trains on different augmented images on every epoch. This massively increased the accuracy on the overall test image. This model perfectly classified the image from Fig 6. However, as seen in Fig. 7, when ran through a different test image there was a false positive in the bottom left of the image.

The next model was trained using the same data augmenter, but the number of batches used per epoch was increased to 200 so the model was trained on 10,000 images per epoch. This model perfectly classified all the ships in both test images with no false positives in either.

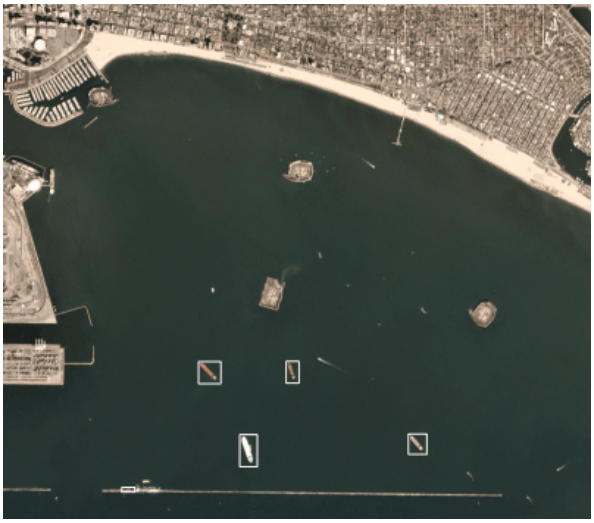

Fig. 7. Test for first model trained using data augmentation.

Fig. 8. Test for Mdel Trained on Set 2 with Data Augmentation.

D. Transfer Learning

To exhaust other possibilities, the original VGG-16 architecture with an input size of 224 by 244 was tested using our pipeline. The training dataset and object proposals were scaled up to 224 by 224. The models all used pretrained weights for the convolutional layers and these layers were also frozen so the weight values would not change. The fully connected layers were then trained from scratch along with the sigmoid classifier.

Six different models were trained and tested to find the optimal hyperparameters to train a model using transfer learning. Three of the models were trained using the original training dataset while the other three were trained using data augmentation. Table III depicts the results of the testing.

TABLE III
VALIDATION LOSS AND ACCURACY VALUES FROM TESTING

Hyperparameter Set	w/o Data Augmentation		w/ Data Augmentation	
	Loss	Accuracy	Loss	Accuracy
1	0.1450	98.60%	3.7394	76.80%
2	0.0601	97.80%	0.0395	98.60%
3	0.1274	95.80%	0.0575	97.00%

Hyperparameter set 1 is the same as the default hyperparameters with a epoch size of 10. Set 2 has a learning rate of $1 \cdot 10^{-4}$, decay rate of 0.75, and an epoch size of 10. Set 3 has a learning rate of $1 \cdot 10^{-5}$, decay rate of 0.65, and an epoch size of 30.

Most of the values in Table III look decent, disregarding set 1 with data augmentation. The issue does not surface until these models are tested using the object proposals from the larger satellite images. There are on average around five false positive in each test image, such as in Fig 8.

Also, the runtime between the modified VGG-16 architecture and the original VGG-16 architecture when running our entire pipeline is pretty significant. On average, the models trained with transfer learning are approximately 30% slower than the models with the modified VGG-16 architecture.