Analysis and Optimization of Machine Learning Models for Ship Detection in Satellite Imagery

Jay Patel, Bansikumar Mendapara, and Pavankumar Mahadik

***Abstract*—Shipping traffic is growing fast. More ships increase the chances of infractions at sea like environmentally devastating ship accidents, piracy, illegal fishing, drug trafficking, and illegal cargo movement. This has compelled many organizations to have a closer watch over the open seas. Satellite imagery provides unique insights into various markets, including agriculture, defense and intelligence, energy, and finance. This flood of new imagery is outgrowing the ability for organizations to manually look at each image that gets captured. To this end, we investigated the use of machine learning and computer vision to detect ships from satellite images.**

I. Introduction

In this paper we discuss the analysis and optimization of three different machine learning algorithms as they are applied to the task of detecting ships in satellite imagery. Firstly, We trained our model using image data which contains NoShip and Ship images with labels. After that, We try to find ship in satellite image using sliding window method. Specifically, we compare the use of Convolution Neural Network(CNN), Support Vector Machine(SVM), and VGG16. We then tune the hyper parameters in order to increase the precision of these models. The ideal model will detect all ships in satellite image. This research can be applied to the development of ship monitoring system to anticipate threats, trigger alerts, and improve efficiency at sea.

II. Dataset

A. Description

Ships in Satellite Imagery 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 labeled with either a ”ship” or ”no-ship” classification. Provided is a zipped directory shipsnet.zip that contains the entire dataset as .png image chips. Each individual image filename follows a specific format which includes label, unique scene id, longitude and latitude. 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 ”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. The ”no-ship” class includes 3000 images. It includes a random sampling of different landcover features that do not include any portion of an ship. ”Partial ships” that contain only a portion of a ship, but not enough to meet the full definition of the ”ship” class are also included in NoShip images. Satellite imagery used to build this dataset is made available through Planet’s Open California dataset, which is openly licensed.

B. Analysis

As with all models, the first step is to load the data. In this case we are going to load the JSON file. Once we load the file we convert into dataframe to view the data of first and last 5 images:

	data	labels	locations	scene_ids
0	[82, 89, 91, 87, 89, 87, 86, 86, 86, 86, 84, 8...	1	[-118.2254694333423, 33.73803725920789]	20180708_180909_0f47
1	[76, 75, 67, 62, 68, 72, 73, 73, 68, 69, 69, 6...	1	[-122.33222866289329, 37.7491755586813]	20170705_180816_103e
2	[125, 127, 129, 130, 126, 125, 129, 133, 132, ...	1	[-118.14283073363218, 33.736016066914175]	20180712_211331_0f06
3	[102, 99, 113, 106, 96, 102, 105, 105, 103, 10...	1	[-122.34784341495181, 37.76648707436548]	20170609_180756_103a
4	[78, 76, 74, 78, 79, 79, 79, 82, 86, 85, 83, 8...	1	[-122.34852408322172, 37.75878462398653]	20170515_180653_1007
	data	labels	locations	scene_ids
3995	[126, 122, 124, 138, 165, 186, 195, 199, 203, ...	0	[-122.08693255500694, 37.77781408256089]	20170815_180821_102d
3996	[130, 134, 139, 128, 117, 126, 141, 147, 142, ...	0	[-122.10549691828378, 37.76946626247702]	20170730_191230_0f21
3997	[171, 135, 118, 140, 145, 144, 154, 165, 139, ...	0	[-122.48298739296371, 37.684929808845375]	20161116_180804_0e14
3998	[85, 90, 94, 95, 94, 92, 93, 96, 93, 94, 94, 9...	0	[-122.29028216570079, 37.71632091139081]	20170211_181116_0e16
3999	[122, 122, 126, 126, 142, 153, 174, 190, 185, ...	0	[-122.49531387721586, 37.698557210117706]	20180206_184438_1043

Fig. 1. Some samples from the Ship dataset

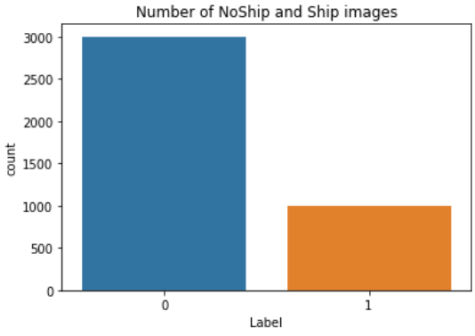

Fig. 2. Number of NoShip and Ship Images

Like we went over before, the data is split into two parts, one with the label, NoShip or Ship, and one with the data values. Next, let’s do a little analysis with our data to see what we are working with. First of all, We looked into data distribution. For that we used bar graph to represent number of NoShip and Ship images. Even we can use Matplotlib to

create a pie chart to show the breakdown of Noship and Ship images.

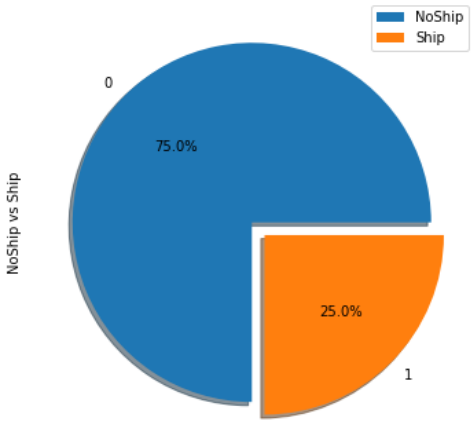

Fig. 3. Distribution of NoShip and Ship images

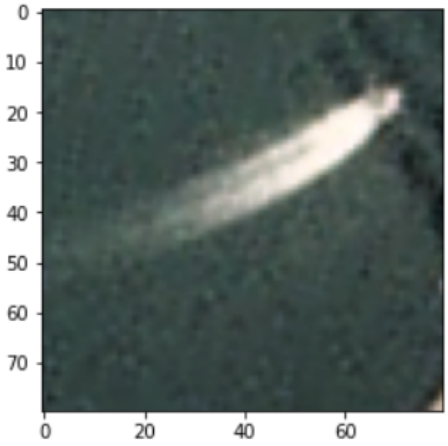

Fig. 4. Random NoShip image from the dataset

As you can see, a lot of our training data is NoShip. About 75% of our dataset consists of NoShip images. While we split our data into training and testing data we will have to use stratified sampling or shuffling otherwise we have a chance of our training model being skewed toward NoShip images. If the sample we choose to train our model consists mostly NoShip images, it may end up predicting everything as NoShip due to the over abundance of NoShip training data. We want to keep in mind that it's not a big deal if we miss the odd NoShip image but we definitely do not want to mark a ship image as NoShip image, therefore precision is very important. In figure 4, we read one random NoShip image from our dataset. Refer to figure 5 for the sample ship image from our dataset.

C. Preprocessing

Data preprocessing is an integral step in Machine Learning as the quality of data and useful information that can be derived from it directly affects the ability of our model to learn. Therefore, it is extremely important that we preprocess our data before feeding it into our model. Our dataset labels were in the form of 0(NoShip) or 1(Ship). To convert them into categorical value we have used to_categorical function from keras library. For image processing, We have tried canny function of cv2 library but it did not work out well as there were few samples under NoShip label which looks almost same.

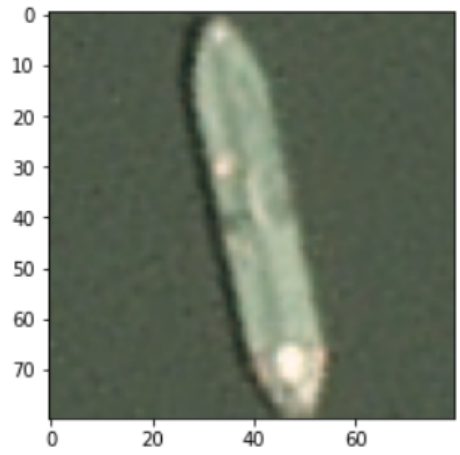

Fig. 5. Normal Ship image from the dataset

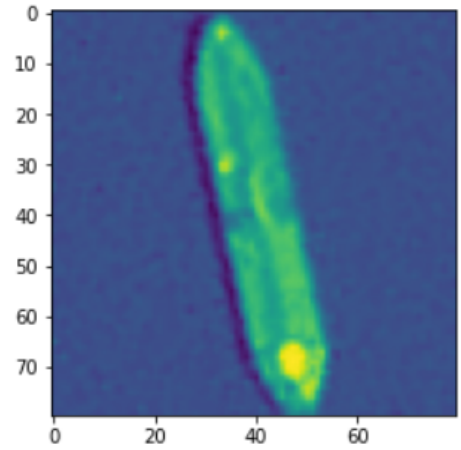

Fig. 6. Grayscale image of ship

For SVM, we tried so many image processing techniques and Histogram of Oriented Gradients (HOG) gave the best results. HOG, or Histogram of Oriented Gradients, is a feature descriptor that is often used to extract features from image data. The HOG descriptor focuses on the structure or the shape of an object. HOG is able to provide the edge and edge direction as well. This is done by extracting the gradient and orientation of the edges. Additionally, these orientations are calculated in 'localized' portions. This means that the complete image is broken down into smaller regions and for each region, the gradients and orientation are calculated. We

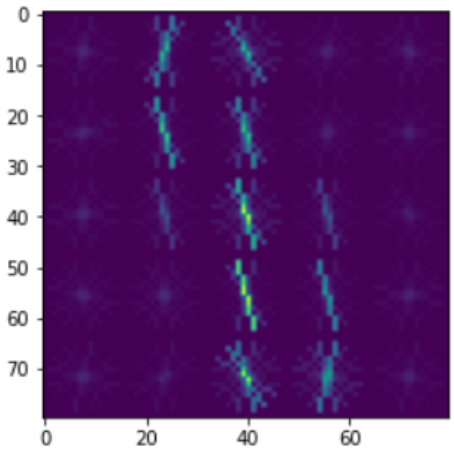

Fig. 7. HOG features of Ship image

will discuss this in much more detail in the upcoming sections. Finally the HOG would generate a Histogram for each of these regions separately. The histograms are created using the gradients and orientations of the pixel values, hence the name 'Histogram of Oriented Gradients'. But to work with HOG we have to convert RGB image to gray scale image. After that we used hog function of skimage library which helps us to find hog features from the given image. Figure 6 and 7 represent grayscale image and hog features of the same ship image respectively.

For CNN and VGG16, We do not need to do any kind of feature extraction because convolution layer is enough capable to get features from the images.

III. Machine Learning Models

A. Support Vector Machine

Support Vector Machines are a set of supervised learning methods used for classification, regression, and outliers detection. The objective of a support vector machine is to find a hyperplane that separates classes in feature space. While there are many possible hyperplanes that can be used to separate data, we want to find the separating hyperplane that provides the maximum margin between data points of both classes. This solution is optimal because it is robust to outliers and has strong generalization ability.

In particular, the code uses SVC, or Support Vector Classifier, which is a kernel based implementation of support vector machines. Kernel SVM is used when the relationship between the classes in feature space is non-linear. As such, linear SVM is not feasible when the relationship between features is non-linear, so we must look to other methods in order to separate features. Kernel SVM uses a kernel defined in higher dimensional space that ideally is able to separate the features. The kernel defines the shape of the separating hyperplane so careful consideration must be given to the proper kernel choice.

As mentioned before, for feature extraction we use HOG features which give edge and edge directions. We split out

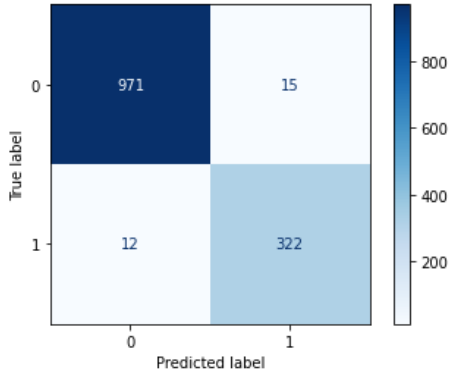

Fig. 8. Confusion Matrix before Hyperparameter Tuning

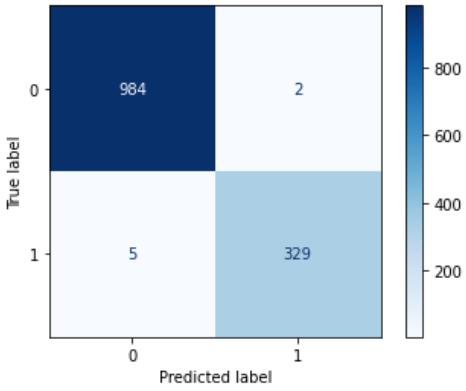

Fig. 9. Confusion Matrix after Hyperparameter Tuning

the data into training and testing sets. Once we tuned the hyperparameters by testing them on the testing set, we used the optimal parameters on the satellite image to evaluate its performance. We decided to prioritize the precision metric above the overall accuracy for tuning the hyperparameters.

```
Accuracy: 0.9795454545454545
```

	precision	recall	f1-score	support
0	0.99	0.98	0.99	986
1	0.96	0.96	0.96	334
accuracy			0.98	1320
macro avg	0.97	0.97	0.97	1320
weighted avg	0.98	0.98	0.98	1320

Fig. 10. Classification Report before Hyperparameter Tuning

The reason we did this is because we believe it is acceptable, although inconvenient, to misclassify NoShip, but we wanted to minimize the misclassification of ship. We tried hyperparameter tuning for SVC parameter C, kernel and gamma. We used GridSearchCV to tune the hyperparameters with cross-validation. I got best parameter values for SVM. We used this parameters to detect the ships in satellite image.

```
Accuracy: 0.9946969696969697
```

	precision	recall	f1-score	support
0	0.99	1.00	1.00	986
1	0.99	0.99	0.99	334
accuracy			0.99	1320
macro avg	0.99	0.99	0.99	1320
weighted avg	0.99	0.99	0.99	1320

Fig. 11. Classification Report after Hyperparameter Tuning

In the confusion matrix, 0 represents NoShip and 1 represents Ship. We can see that before hyperparameter tuning false negatives are high which is not good for our problem definition. When we performed hyperparameter tuning, number of false negative decrease.

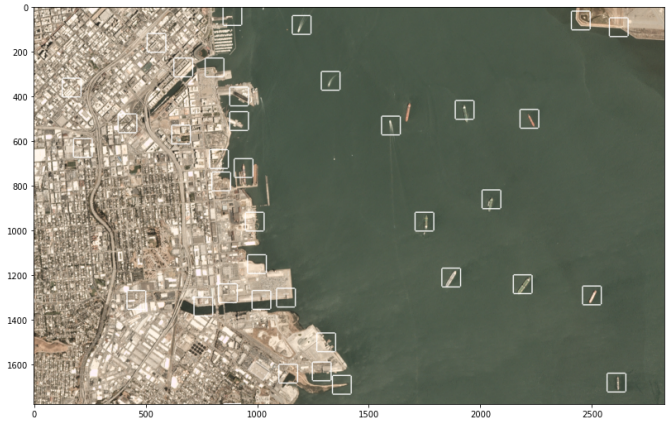

Fig. 12. Testing before Hyperparameter Tuning

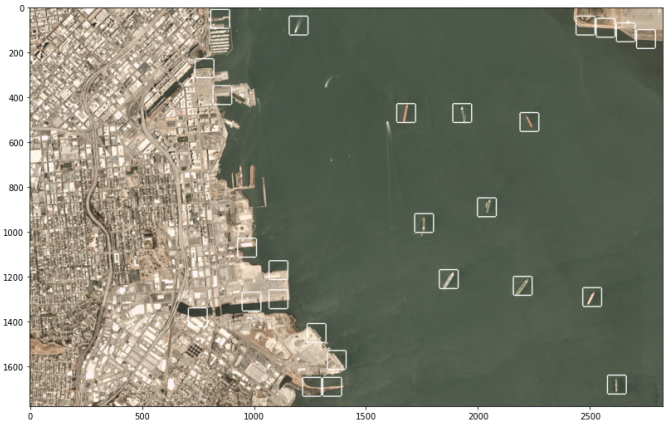

Fig. 13. Testing after Hyperparameter Tuning

Figure 10 and 11 represents classification report and accuracy score before and after hyperparameter tuning. We tested both model on our test image using sliding window method. First model gave so many false positives. Even it could not detect one ship from the image. But second model performed very well as it detected all ships and gave less false positives also.

B. Convolutional Neural Network (CNN)

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. The objective of the Convolution Operation is

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 80, 80, 32)	896
conv2d_2 (Conv2D)	(None, 80, 80, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 40, 40, 32)	0
dropout_1 (Dropout)	(None, 40, 40, 32)	0
conv2d_3 (Conv2D)	(None, 40, 40, 32)	9248
conv2d_4 (Conv2D)	(None, 40, 40, 32)	9248
max_pooling2d_2 (MaxPooling2	(None, 20, 20, 32)	0
dropout_2 (Dropout)	(None, 20, 20, 32)	0
conv2d_5 (Conv2D)	(None, 20, 20, 32)	9248
conv2d_6 (Conv2D)	(None, 20, 20, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 10, 10, 32)	0
dropout_3 (Dropout)	(None, 10, 10, 32)	0
conv2d_7 (Conv2D)	(None, 10, 10, 32)	102432
max_pooling2d_4 (MaxPooling2	(None, 5, 5, 32)	0
dropout_4 (Dropout)	(None, 5, 5, 32)	0
flatten_1 (Flatten)	(None, 800)	0
dense_1 (Dense)	(None, 512)	410112
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 2)	1026

```
Total params: 560,706
Trainable params: 560,706
Non-trainable params: 0
```

Fig. 14. Summary of Model

to extract the high-level features such as edges, from the input image. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to

the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

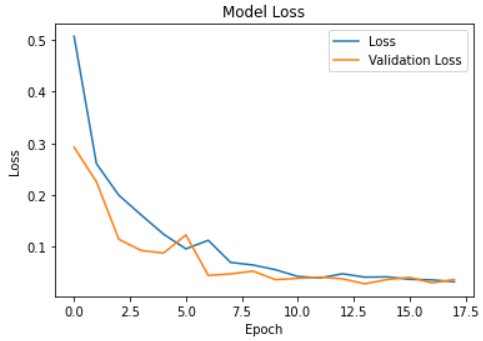

Fig. 15. Loss Curve

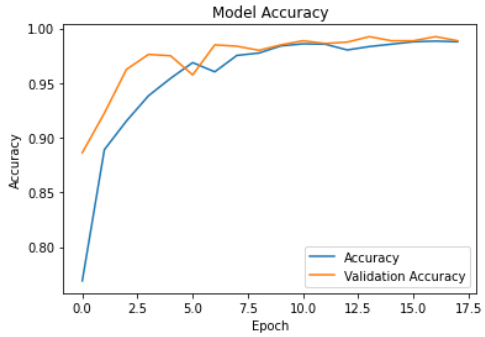

Fig. 16. Accuracy Curve

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. Next, Dropout is

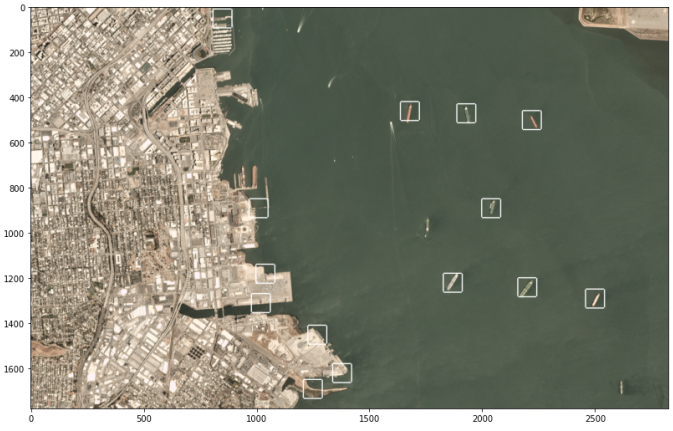

Fig. 17. First Model Testing

a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase. The Fully-Connected layer is learning a possibly non-linear function in that space. Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique. Figure 14 shows summary of our CNN model.

Fig. 18. Saved Model Testing

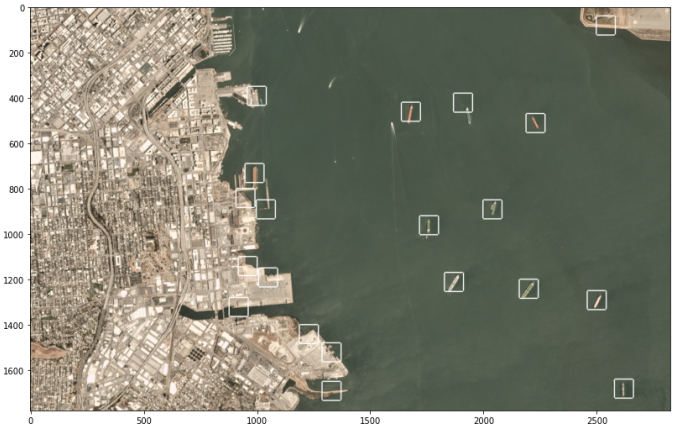

Fig. 19. Testing after Hyperparameter Tuning

For CNN, We have printed loss and accuracy curve so we can analyze our performance at each epoch. Here we have seen that after some epoch model performance is stable and sometimes problem of over-fitting also occur. To overcome this issue, we have used callbacks with EarlyStopping and ModelCheckpoint methods which will stop the training if variation in loss or accuracy is not greater than the threshold and save the model. To view performance difference, we have tested trained model and saved model on our testing image and we came to know that saved model perform very well. We want to try different parameters and tweak the model. To implement this we used GridSearchCV with dropout_rate,

optimizer and activation. At the end we got best parameter value and applied our test image. Testing of all models is shown in figure 17, 18 and 19. Overall, saved model and best parameter model performed very similar.

C. VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. Most unique

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 80, 80, 3)	0
block1_conv1 (Conv2D)	(None, 80, 80, 64)	1792
block1_conv2 (Conv2D)	(None, 80, 80, 64)	36928
block1_pool (MaxPooling2D)	(None, 40, 40, 64)	0
block2_conv1 (Conv2D)	(None, 40, 40, 128)	73856
block2_conv2 (Conv2D)	(None, 40, 40, 128)	147584
block2_pool (MaxPooling2D)	(None, 20, 20, 128)	0
block3_conv1 (Conv2D)	(None, 20, 20, 256)	295168
block3_conv2 (Conv2D)	(None, 20, 20, 256)	590080
block3_conv3 (Conv2D)	(None, 20, 20, 256)	590080
block3_pool (MaxPooling2D)	(None, 10, 10, 256)	0
block4_conv1 (Conv2D)	(None, 10, 10, 512)	1180160
block4_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block4_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block4_pool (MaxPooling2D)	(None, 5, 5, 512)	0
block5_conv1 (Conv2D)	(None, 5, 5, 512)	2359808
block5_conv2 (Conv2D)	(None, 5, 5, 512)	2359808
block5_conv3 (Conv2D)	(None, 5, 5, 512)	2359808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 512)	1049088
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 2)	1026

Total params: 15,764,802
Trainable params: 8,129,538
Non-trainable params: 7,635,264

Fig. 20. Summary of VGG16

thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output.

Fig. 21. VGG16 model testing

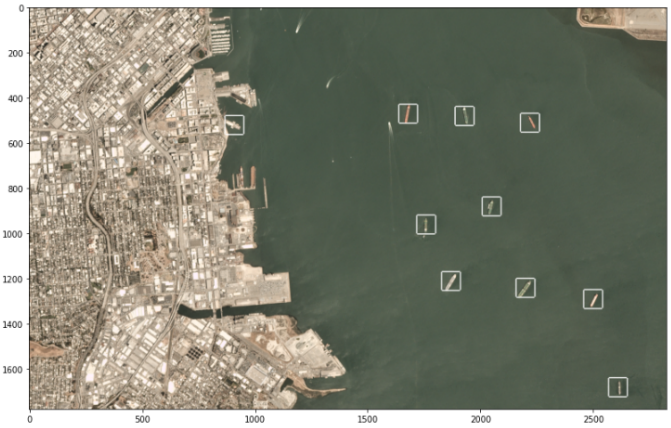

Fig. 22. Saved VGG16 model testing

Here we have used customize input and output layer according to our problem. To overcome problem of over fitting and to view internal state, we have applied callbacks with EarlyStopping and ModelCheckpoint. Undoubtedly, VGG16 is best model for computer vision. It performed very well with very less false positive and false negative. Testing is shown in figure 21 and 22. We even tried to test saved model and it performed similar to original one. So far VGG16 is the best model for this problem.

IV. Conclusion

After analyzing and tuning three different machine learning models, we can conclude that all of them did a very good job of detecting ships from satellite images. All three model were able to properly classify every ship in the satellite image. VGG16 gave only one false positive. But CNN gave few false positives. At last, SVM gave so many false positives

but still able to classifiy all true positives. By using in-class precision as the primary metric, every model succeeded in properly classifying 99-100% of the ship images. The discerning performance factor between the three models is then determined by how well they classified ship images. Additionally, we observed the importance of data preprocessing and hyperparameter tuning and the impact it has on model performance. In the case of the convolution neural network performance improved with tuning the hyperparameters. In the case of SVM, tuning the hyperparameter allows us to remove few false positives. With VGG16, pretrained network and good architecture enabled model to reach very good performance.

Lastly, we looked at some of the misclassified images to see if we could gain any insights into increasing model performance further. However, upon inspection we came to know that, there were few parts in image which were similar to ship shape. More research needs to be done in this area and gaining these insights will ultimately lead to a ship detection technique that reduces errors to a minimum.

REFERENCES

[1] Rhammell. "Ships in Satellite Imagery", 8 Oct. 2017, https://www.kaggle.com/rhammell/ships-in-satellite-imagery.

[2] Simonyan, Karen and Zisserman, Andrew. "Very Deep Convolutional Networks for Large-Scale Image Recognition", 4 Sep. 2014, https://arxiv.org/abs/1409.1556.

[3] Muneeb ul Hassan. "VGG16 – Convolutional Network for Classification and Detection", 20 Nov. 2018, https://neurohive.io/en/popular-networks/vgg16/.

[4] Rohit Thakur. "Step by step VGG16 implementation in Keras for beginners", 6 Aug. 2019, https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c.

[5] Vitali Burachonok. "Keras for search ships in satellite Image", 27 Nov. 2018, https://www.kaggle.com/byrachonok/keras-for-search-ships-in-satellite-image.

[6] Jordi TORRES. "Convolutional Neural Networks for Beginners", 23 Sep. 2018, https://towardsdatascience.com/convolutional-neural-networks-for-beginners-practical-guide-with-python-and-keras-dc688ea90dca.

[7] Eijaz Allibhai. "Building a Convolutional Neural Network (CNN) in Keras", 16 Oct. 2018, https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5.

[8] Sumit Saha. "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way", 15 Dec. 2018, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[9] Scikit-learn. "Support Vector Machines" scikit-learn.org/stable/modules/svm.html.

[10] Bhattacharyya, Saptashwa. "Support Vector Machine: Kernel Trick; Mercer's Theorem" Towards Data Science, 19 Dec. 2018, towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d.

[11] Gandhi, Rohith. "Support Vector Machine — Introduction to Machine Learning Algorithms" Towards Data Science, 7 Jun. 2018, towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[12] Manik Galkissa. "Training SVM classifier with HOG features", 6 Oct. 2017, https://www.kaggle.com/manikg/training-svm-classifier-with-hog-features.