Ships Identification in Satellite Images Using Machine Learning Techniques

Weilun Zhang
(A53239629)
wez242@eng.ucsd.edu

Zhaoliang Zheng
(A53254665)
zhz503@neg.ucsd.edu

Mingchen Mao
(A53248197)
m8mao@ucsd.edu

Abstract

Satellite imagery provides unique and important insights in many aspects of our daily life. As a result, there is a need for machine learning algorithms to help automate the satellite image analysis process, which can be applied to issues such as monitoring port activity and supply chain analysis. In this project, we investigated different machine learning algorithms and their effects associated with corresponding data processing techniques. Our results indicated that the Convolutional Neural Network algorithm, with a Stochastic Gradient Descent optimizer, provides the best performance among various algorithms. Our results also revealed the fact that using more training data is not always the wisest choice, especially when an overfitting issue is very likely to happen.

1. Introduction

1.1. Background

Satellite imagery provides unique insights into various markets, including agriculture, defense, intelligence, energy, and finance. New commercial imagery providers, such as Planet and BlackSky, are using constellations of small satellites to exponentially increase the amount of images of the earth captured every day. This flood of new imagery is outgrowing the ability for organizations and individuals to manually look at each image that gotten captured, and there is a need for machine learning and computer vision algorithms to help automate the analysis process. Automating this process can be applied to many issues including monitoring port activity levels and supply chain analysis.

1.2. Data

The dataset used for our training purpose consists of cropped images extracted from planet satellite imagery collected over the San Francisco Bay area. Totally 3600 80x80 RGB images are included in this dataset, 900 of them were labeled as “ship” while the rest are labeled as “no-ship”, which were used as disturbances and counterparts to examine our accuracy later for our proposed algorithms.

Each of these 3600 images is stored as a list of 19200 integers, which represent the color information in the classical 0 to 255 scale. The RGB channels are evenly distributed in this list. The first 1/3 integers represent the red channel, the last 1/3 integers represent the blue channel and the rest are representations of the green channel.

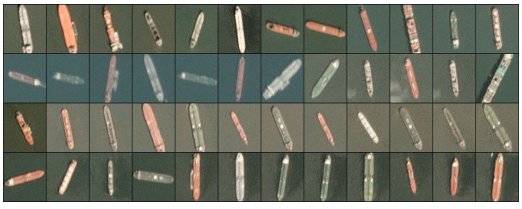

Figure 1. Samples of images with label “ship”

Ships of different sizes, orientations, and atmospheric collection conditions are included in our dataset, which reflect the real imagery of satellite images.

While images that are labeled "no-ship" are more diverse, which mainly fall into 3 categories.

- Images of land cover features - water, vegetation, bare earth, buildings, etc.
- Images of incomplete ships which should be obviously identified to not fall into the "ship" class.
- Images of previously mislabeled models, which are typically caused by bright pixels or strong linear features

Figure 2. Samples of images with label "no-ship"

1.3. Data preprocessing

Data preprocessing includes the following steps:

- Reshape data
- Normalize data
- Shuffle all indexes
- Output encoding
- Separate row pictures into 3 channels

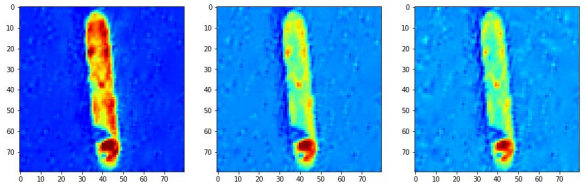

Figure 3. Sample picture displayed in rgb form

2. Mathematical Methods

2.1. Algorithms

2.1.1. KNN (K-Nearest Neighbours)

The KNN classifier is a non parametric, instance-based supervised learning algorithm. In classification problems, the K-nearest neighbor algorithm essentially play a role in forming a majority vote between the K most similar instances to a given "unseen" observation. Similarity is defined according to a distance metric between two data points. Euclidean distance is popular in deciding whether an unseen observation is a neighbour of the given point while Manhattan, Chebyshev and Hamming distance can also be used.

$$d(x,x') = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2 + \ldots + (x_n - x_n')^2}$$

It runs through the whole dataset computing d between the given point x and each training observation. We'll call the K points in the training data that are closest to x the set S. It then estimates the conditional probability for each class. Finally, our input x gets assigned to the class with the largest probability.

$$P(y = j \mid X = x) = \frac{1}{K} \sum_{i \in A} I(y^{(i)} = j)$$

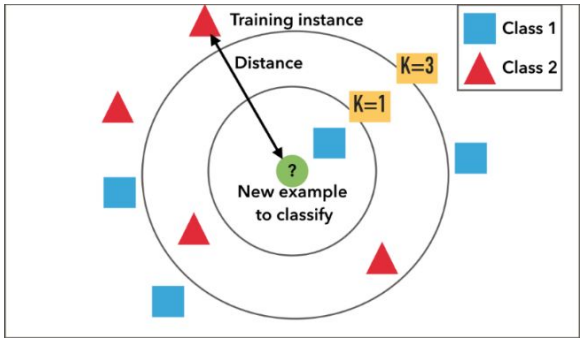

Figure 4. Chart representation of K-neighborhood algorithm

2.1.2. Random Forest

Random forest is a popular machine learning algorithm that is typically used for classification and regression. A random forest classifier consists of multiple decision trees trained by different batches of the provided training data. In particular, the random forest trees tend to overfit their training sets. However,

random forest averages multiple decision trees to reduce the variance and thus preventing our training from overfitting.

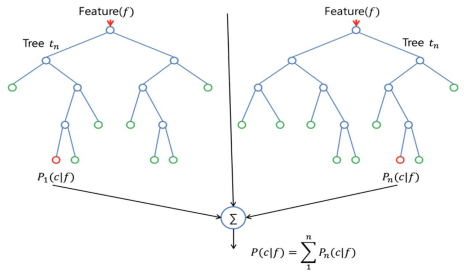

Figure 5. Illustration of a random forest tree

Bootstrap aggregating or bagging is the fundamental technique of the random forest training algorithm. A typical bootstrap process is as follow:

$$C_b(x) = arg(x)\ max\{C(S^k, x)\}^{m:1\rightarrow M}$$

Where $C_b(x)$ represents the bagged classification, S is the training set, and x is the predicted value from our training set. M random training subsets are extracted from the training set and bootstrapped. Again, such randomness is able to help reduce the variance and overfitting.

Random forest is bootstrapping based but not exactly the same, the main difference is that random forest uses a modified tree learning algorithm that selects a random subset of each feature through each learning process. The logic behind this process is that most times a few features are dominant to our predictions, which will be selected for training more often than other features and eventually leads to a correlation among these dominant features.

A rule of thumb in random forest split is that for a classification problem with p features, $\sqrt{p}$ features should be used in each split, a round to the closest integer should be performed when necessary.

2.1.3 Support Vector Machine

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. The algorithm gives an optimal hyperplane that can categorized new examples.

2.1.3.1 Maximal-Margin Classifier

The Maximal-Margin Classifier is a hypothetical classifier that best explains how SVM works in practice and this is why we will introduce this method first. In SVM, a hyperplane is selected to best separate the points in the input variable space by their class, either class 0 or class 1. In two-dimensions you can visualize this as a line and let's assume that all of our input points can be completely separated by this line. An straightforward example would be:

$$b0 + (w1 * X1) + (w2 * X2) = 0$$

Where the coefficients (w1 and w2) that determine the slope of the line and the intercept (b0) are found by the learning algorithm, and X1 and X2 are the two input variables. By plugging in input values into the line equation, we can calculate whether a new point is above or below the line. Above the line, the equation returns a value greater than 0 and the point belongs to the first class (class 0). A value close to the line returns a value close to zero and the point may be difficult to classify. The distance between the line and the closest data points is referred to as the margin. The best or optimal line that can separate the two classes is the line that as the largest margin. This is called the Maximal-Margin hyperplane. The margin is calculated as the perpendicular distance from the line to only the closest points. Only these points are relevant in defining the line and in the construction of the classifier. These points are called the support vectors. They support or define the hyperplane. The hyperplane is learned from training data using an optimization procedure that maximizes the margin.

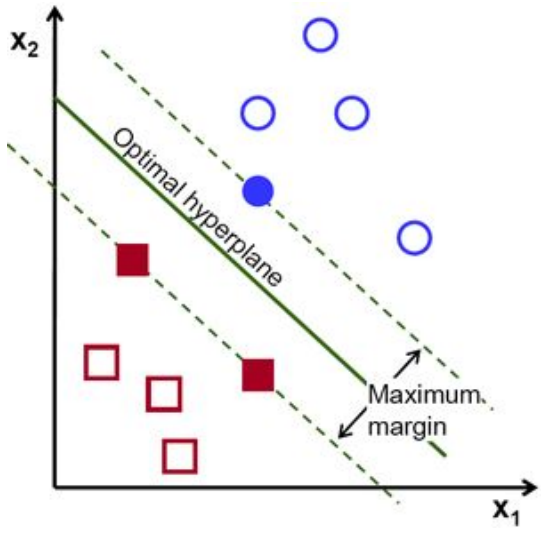

Figure 6. Fundamentals of support vector machine

2.1.3.2 Soft Margin Classifier

The maximal margin method is too ideal to deal with complex data in realise so a soft margin classifier is introduced. The constraint of maximizing the margin of the line that separates the classes is relaxed in this algorithm. This change allows some points in the training data to violate the hyperplane. A tuning parameter is introduced called simply C that defines the magnitude of the wiggle allowed across all dimensions. The C parameters defines the amount of violation of the margin allowed. A C=0 is no violation and we are back to the inflexible Maximal-Margin Classifier described above. The larger the value of C the more violations of the hyperplane are permitted. By allowing certain points to cross the separating line we will get a classifier with more tolerance. Applying a soft margin classifier can not only prevent overfitting in an extent but also can decrease the standard deviation which means the classifier is more stable than before.

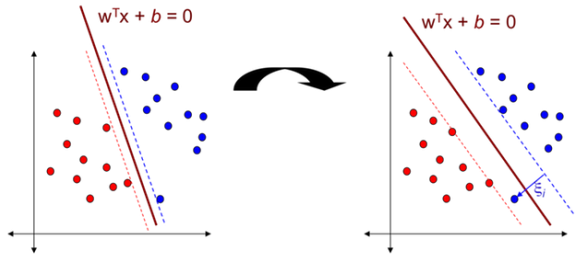

Figure 7. Typical soft margin classifier

2.1.3.3 Linear Kernel SVM

The SVM algorithm is implemented in practice using a kernel. A powerful insight is that the linear SVM can be rephrased using the inner product of any two given observations, rather than the observations themselves. The inner product between two vectors is the sum of the multiplication of each pair of input values. The equation for making a prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

$$f(x) = b0 + sum(a_i * (x, x_i))$$

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients b0 and ai (for each input) must be estimated from the training data by the learning algorithm. The dot-product is called the kernel and can be re-written as:

$$K(x, x_i) = sum(x * x_i)$$

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs.

2.1.4 Convolution Neural Network

2.1.4.1 Neural Network Model

A neural network is put together by hooking together many of the simple “neurons,” so that the output of a neuron can be the input of another.Taking the figure below as an example,

we will explain here the framework of Neural Network.

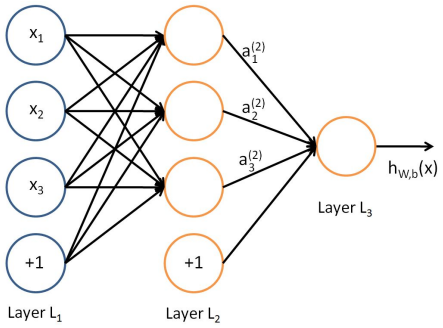

Figure 8. A layer representation of CNN

Our neural network has parameters $(W,b)=(W^{(1)},b^{(1)},W^{(2)},b^{(2)})$, where $W_{ij}^{(l)}$ denotes the parameter associated with the connection between unit j in layer l, and unit i in layer l+1. Also, $b_i^{(l)}$ is the bias associated with unit i in layer $l+1$. We will write $a_i^{(l)}$ to denote the activation of unit i in layer j. Given a fixed setting of the parameters W,b, our neural network defines a hypothesis $h_{W,b}(x)$ that outputs a real number. Specifically, the computation that this neural network represents is given by:

$$a_1^{(2)}=f(W_{11}^{(1)}x_1+W_{12}^{(1)}x_2+W_{13}^{(1)}x_3+b_1^{(1)})$$
$$a_2^{(2)}=f(W_{21}^{(1)}x_1+W_{22}^{(1)}x_2+W_{23}^{(1)}x_3+b_2^{(1)})$$
$$a_3^{(2)}=f(W_{31}^{(1)}x_1+W_{32}^{(1)}x_2+W_{33}^{(1)}x_3+b_3^{(1)})$$
$$h_{W,b}(x)=f(W_{11}^{(2)}a_1^{(2)}+W_{12}^{(2)}a_2^{(2)}+W_{13}^{(1)}a_3^{(2)}+b_1^{(2)})$$

2.1.4.2 Backpropagation Algorithm

We can train our neural network using batch gradient descent. In detail, for a single training example (x,y), we define the cost function with respect to that single example to be:

$$J(W,b;x,y)=\tfrac{1}{2}\|h_{W,b}(x)-y\|^2$$

We can then define the overall cost function to be:

$$J(W,b)=[\tfrac{1}{m}\sum_{i=1}^{m}J(W,b;x^{(i)},y^{(i)})]+\tfrac{\lambda}{2}\sum_{l=1}^{m-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

The first term in the definition of $J(W,b)$ is an average sum-of-squares error term. The second term is a regularization term that tends to decrease the magnitude of the weights, and helps prevent overfitting. Then the backpropagation algorithm is give below. First perform a feedforward pass, computing the activations for layers L_2,L_3, and so on up to the output layer L_{n_l}. For each output layer unit i in layer n_l, set:

$$\delta_i^{(n_l)}=\frac{\partial}{\partial z_i^{(n_l)}}\tfrac{1}{2}\|y-h_{W,b}(x)\|^2=-(y_i-a_i^{(n_l)})\cdot f'(z_i^{(n_l)})$$

For each node i in layer l, set

$$\delta_i^{(l)}=(\sum_{j=1}^{s_{l+1}}W_{ji}^{(l)}\delta_j^{(l+1)})f'(z_i^{(l)})$$

Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b;x,y)=a_j^{(l)}\delta_i^{(l+1)}$$
$$\frac{\partial}{\partial b_i^{(l)}}J(W,b;x,y)=\delta_i^{(l+1)}$$

To train our neural network, we can now repeatedly take steps of gradient descent to reduce our cost function $J(W,b)$.

CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on CNNs.

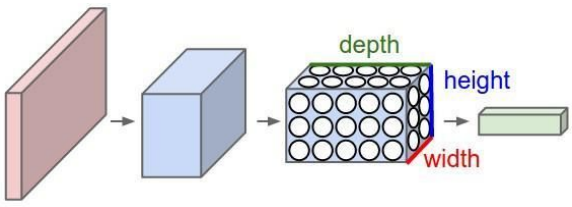

Figure 9. How CNN method proceeds

2.1.4.3. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a commonly used optimization algorithm. It straightforwardly adopted the purpose of minimizing an objective function that has a form of sums. A random batch is selected for each iteration, namely stochasticly, under which the objective function will be minimized through each iteration.

$$w_{new} = w - \eta \nabla Q(w)$$

2.1.5. Other Algorithms

Except for the algorithms mentioned above, we also tested 6 other algorithms with less significance. Gaussian Naive Bayesian Classifier (GNB)

$$p(x = v|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Lagrangian Support Vector Machine (LSVM)

Decision Tree Classifier (DTC)

Entropy: $E(S) = \sum_{i=1}^{c} -p_i log_2 p_i$

Linear Regression (LR)

$$y_i = x_i^T \beta + \varepsilon_i$$

Gaussian Bayes Classifier (GBC)

$$p(y = i|x) = \frac{\frac{1}{(2\pi)^{m/2}|S_i|^{1/2}} e^{-0.5(x_k-\mu_i)^T S_i (x_k-\mu_i)} p_i}{p(x)}$$

XGBoost (XGB)

More details see reference #.

2.2. Accuracy and Cross Validation

2.2.1 K-Fold Cross Validation

K-fold cross validation is a method that is widely used in machine learning validation problems. While performing K-fold cross validation, the dataset is firstly partitioned into K subsets. Then perform K iterations such that each subset is used as validation set while the others are used as training sets. Train the machine learning model using the cross validation training set and calculate the accuracy of your model by validating the predicted results against the validation set. Finally, estimate the accuracy of the machine learning model by averaging the accuracies derived in all the *k* cases of cross validation.

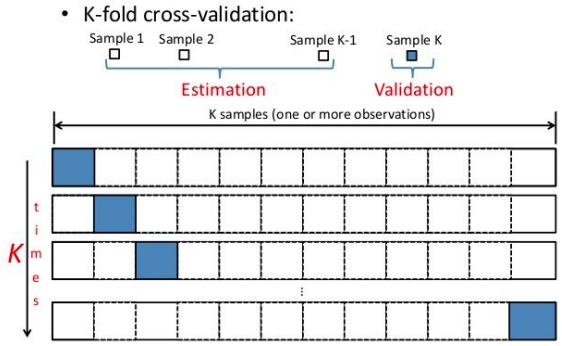

Figure 10. K-fold cross-validation

2.3. Hyper-tuning on parameters

In these algorithms, there are lots of parameters we could change in it, and in fact, some of these parameters could make a great impact on the result. We call the process of tuning these parameters as hyper-tuning.

For simple algorithm like logistic regression, Gaussian naïve bayes, support vector machine, there are no much parameters that need to be tuned. But for XGB, random forest tree, there are lots of parameters which need to be tuned.

Take random forest as an example, there are 4 parameters need to be adjusted: max_features, max_depth, min_samples_leaf, min_samples_split.

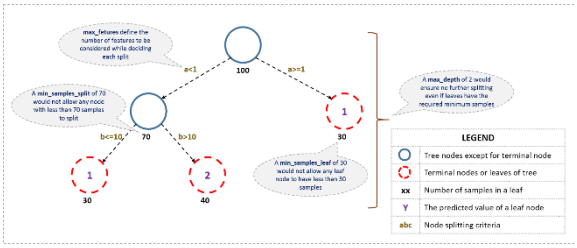

Figure 11. hyper -parameters in random tree

After hyper-tuning, the result usually will increase 1-2%, but this process would normally takes a lot time to finish.

3.Result Analysis

Basically two categories of tests were conducted to investigate the ability of proposed models in terms of classifying pre-cropped and labeled images (small size) and raw images (large size) taken from satellites respectively. The small image tests were mainly aimed to examine the effectiveness of our data pre-processing techniques and the accuracy our various models. As a reference for the big image tests, the small image tests provided insights of each model and algorithm to make the big image test more efficient and pertinent, whose results examined our data post-processing techniques and validated the results of the small image tests.

3.1.Small-Image Test

10 different models and algorithms were investigated in this test, which were used as references in the future for the big-image tests. These models and algorithms were selected from a large common-used machine learning algorithm pool for various purposes. For example, the Gaussian Naïve Bayesian model was based on an improper assumption and its failure can be predicted, so it was selected as a benchmark and validation tool to examine our methods and other models. The XGBoost algorithm has over 10 parameters, which guarantee this model great potential for us to discover, we also hoped that it became the best model after hypertunning.

The K-fold validation was used to examine the accuracy and consistency of our algorithms. We used 90 % of data for training purposes and 10 % of data for validation, which is equivalent to a 10-fold validation. The mean of accuracy and standard deviation of accuracy for these 10 tests were summarized in the table below.

	Mean of Accuracy	Standard Deviation of Accuracy
GNB	0.642014	0.028610
SVM	0.746875	0.028986
LR	0.874653	0.022312
LSVM	0.875000	0.024006
DTC	0.893056	0.018228
KNN	0.919792	0.014768
RF	0.940625	0.016757
GBC	0.947222	0.012306
XGB	0.953472	0.011948
CNN with SGD	0.966725	0.009720

Figure 12. Summary of mean accuracy and standard deviation of accuracy for various models

This result basically indicates 3 points. Firstly, the model that is more accurate tends to be more consistent as well, which helps us to eliminate the concern of worrying the consistency of a potential model with decent accuracy. Secondly, the CNN model with SGD optimizer gives the best results, which has over 96.6 % of confidence to identify the right category of a given labeled small image. Unsurprisingly, the GNB model yields a terrible accuracy and ranks the least among all models. This on the other hand helps to validate the correctness of our methods. Lastly, the standard deviation of the best model is approximately 1/100 of the mean accuracy, which indicates that this CNN models shall be fairly reliable.

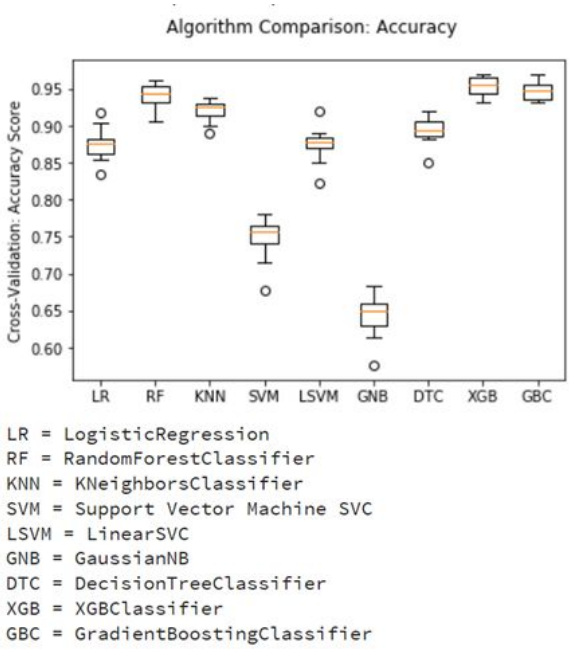

Figure 13. A better representation of our algorithm accuracy comparison

3.2.Big-Image Test

Based on the results from the small-image test, only the CNN model with SGD optimizer was implemented on this big-image test due to the trust to our small-image test and the sake of saving time (it takes a fairly long time to test one model for once).

The first issue we encountered is multi-identification, which is illustrated in the picture below.

Figure 14. A multi-identification example

As Figure # indicated, a single ship and a single ship-like object were identified more than once in our model, which is unnecessary and time-consuming. This is mainly due to the limit of our data preprocessing technique, which failed to prevent the model to identify one single item multiple times. Our solution to this problem was to add extra lines of code called "not near", which assumed the size of a ship based on experience, once our model identified a ship, it will automatically ignore the area that is not sufficient for the present of another ship near this identification. The success of this method relies on the fact that our model has the ability to tell whether or not an object is an intact ship as we trained it with incomplete ship images labeled as "no-ship". The improvement was shown in Figure 12.

The second problem that comes with the big-image test is overfitting. Ships look like ships in similar ways while non-ship objects can be similar to ships in various ways, which greatly confused our model and impaired its capability to distinguish a similar-to-ship object to a real ship. Although most of our training data (2700) are no-ship images, this is still not enough to enumerate all the possible patterns of no-ship objects. As a result, our algorithm identified many ship-like objects as ships, such as extrudes of ports, landscapes with sharp ends and bridges across the sea.

Figure 15. Multiple identification was overcome but many ship-like objects were identified as ships mainly due to overfitting

A better representation of overfitting was illustrated by a confusion matrix in Figure #. As it shown, the model was pretty good at identifying an item that is not a ship, it has 96.2 % accuracy to identify such category. However, as discussed before, due to the irregular shapes lying in our satellite images and the overfitting issue, our model tends to identify many items which are not ship to ships, this number even exceeds the number of real ships.

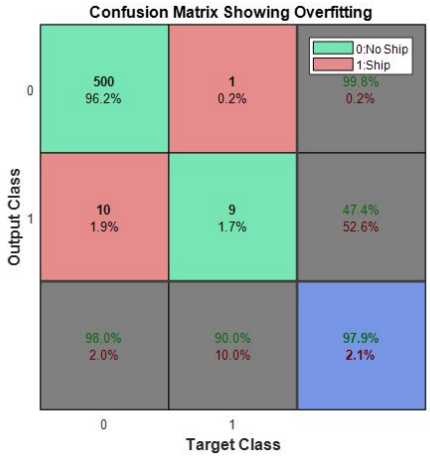

Figure 16. The overfitting issue represented by a confusion matrix

After analysis, we interpolated that the reason of overfitting is mainly that we have used too many training data in our training. Although intuitively more training data tends to yield better accuracy, but this is lying on the foundation that these data are well categorized and preprocessed. As a result, the overfitting issue indicated that our data preprocessing was not mature and powerful enough. Taking a view from this dilemma, we can either better pre-processing our data to make them more distinct but smooth, or we can reduce the amount of training data to avoid this overfitting issue. The latter method was adopted because data preprocessing is really difficult and requires a large scope of projects.

Improvement was obvious after tuning the parameter of the amount of data being trained and validated. Figure # exhibits this trend.

Although the accuracy was only improved by approximately 1.5 % by reducing the amount of data being used for training, but this almost eliminated the overfitting issue. Recall that there are 510 sections on the big-image being categorized as no-ship and only 10 ship present, and the mis-identification rate from a no-ship to a ship according to the confusing matrix is approximately 1.9 %. As a result, the 1.5 % improvement almost eliminated all the cases in the "0 being confused as 1" class. Figure # clearly illustrated this conclusion.

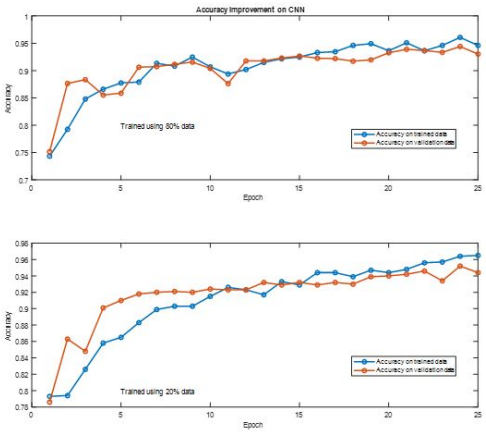

Figure 17. Comparison of training with 80% data and validating with 80% data

Figure 18. Improvement of overfitting after reducing the percentage of data being used for training. Left is the model used 80 % data to train, right is the model used 80 % data to validate.

However, the new model missed one ship parking near the shore at the (700,200) coordinate, which was identified successfully by the previous model. This indicates that there is always a trade-off between the amount of data being used for training. If we use more data to train, it overfits, many ship-like objects will be identified as ships. If we use less data to train, it may lose sensitivity to certain shapes which are actually representations of ships. After some trials, it turned out that using 83.4 % data for validation gives the most balanced model, which is exactly the right image of Figure #.

Of course, our algorithm can be more sophisticated to enhance the robustness of such trade-off. Solutions at hand are better data-preprocessing and postprocessing so that our training data meet more requirements for the big-image test demands. This will be discussed in the conclusion and future work section in detail.

4.Conclusion and Future Work

10 models were implemented to identify ships in satellite images, convolutional neural network (CNN) with stochastic gradient descent (SGD) has the overall best performance. It is able to identify ships in pre-labeled images with an accuracy of 96.6 % and it is able to identify ships in raw satellite images with a considerably accuracy after some parameter tuning.

In the future, we can apply more advanced data preprocessing techniques to better classify our training sets so that overfitting can be minimized. Tuning hyper-parameters of multi-parameter models such as XGBoost to fully exploit its potentials should be performed when more powerful computing resource is available. Nevertheless, collecting more data (both labeled and raw) to enhance confidence of our results and corresponding conclusion.

Reference

[1] T.N. Arnesen, R.B. Olsen, D.J. Weydahl, "Ship detection signatures in AP Mode data", *56th International Astronautical Congress*, pp. 06, 2005.

[2] Krogager, E.; Heiselberg, H.; Møller, J.G.; von Platen, S. Fusion of SAR and EO imagery for Arctic surveillance. In Proceedings of the NATO IST-SET-128 Specialist Meeting, Norfolk, VA, USA, 4–5 May 20.

[3] Daniel, B.; Schaum, A.; Allman, E.; Leathers, R.; Downes, T. Automatic ship detection from commercial multispectral satellite imagery. Proc. SPIE 8743 2013.

[4] Gade, M.; Hühnerfuss, H.; Korenowski, G. Marine Surface Films; Springer: Heidelberg, Germany, 2006.

[5] Towards Data Science. (2018). A Tour of The Top 10 Algorithms for Machine Learning Newbies. [online] Available at: https://towardsdatascience.com/a-tour-of-the-top-10-algorithms-for-machine-learning-newbies-dde4edffae11 [Accessed 10 Jun. 2018].

[6] Jain, Aarshay, et al. “Complete Guide to Parameter Tuning in XGBoost (with Codes in Python).” Analytics Vidhya, 4 Aug. 2016, www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/.

[7]http://steventhornton.ca/hyperparameter-tuning-with-hyperopt-in-python/