Ship and Iceberg Image Classification

Yutong Zhang, Yun Gao, Yutong Liu

Abstract — **Image classification is a complicated and popular topic. Also, nowadays, Machine Learning tools such as Convolution neural network (CNN) and Support Vector Machine (SVM) are extensively used in image classification. In this report, we have developed a suitable way to process data. Meanwhile, we used the traditional SVM, CNN and Boosting algorithms to identify if a remotely sensed target is a shop or iceberg and made a comparison with them. The results achieve the high accuracy of binary classification, and indicate the high capability of our model.**

I. INTRODUCTION

Drifting icebergs present threats to navigation and activities in areas. There are some ways to monitor environmental conditions to assess risks from icebergs, such as aerial reconnaissance and shore-based support. However, these methods are not suitable for harsh weather, long distance, and some other unpredictable conditions. In this case, we consider to using Image Processing method to distinguish whether the target is a ship or an iceberg with the help of satellite.

The remote sensing systems used to detect icebergs are housed on satellites above the Earth and it can capture images day or night. Satellite radar works in much the same way as blips on a ship or aircraft radar. It bounces a signal off an object and records the echo, then that data is translated into an image. Our dataset is consisted of these images. When the satellite radar detects an object, it will show a bright spot in the image because the object reflects more energy than its surroundings. The object may be a ship or an iceberg and we can't tell it directly. So we need extract the main features and analyze them. Here we choose the radar polarization property as our data feature, which represents how the radar transmits and receives the energy. Based on this, we get two-channel data, including HH (transmit/receive horizontally) and HV (transmit horizontally and receive vertically). The Fig.1 shows the specific form of two-channel data. We also get the incidence angle, which means how the satellite sees the image area. These can play an important role in the object characteristics, since objects tend to reflect energy differently. In summary, our dataset concludes four parts, HH, HV, angle, and label.

The previous effort has been made with CNN models, which is classic for solving image classification problems. Such models often involve pre-trained ImageNet weights. The performance was not very decent, because the ImageNet model are focusing on regular object classification or localization, and should not work very well in this problem. That is why we are seeking alternatives. Even with CNN, we tried to add our own layers and train our own weights. Fortunately, the dataset is not too big for a commodity level machine.

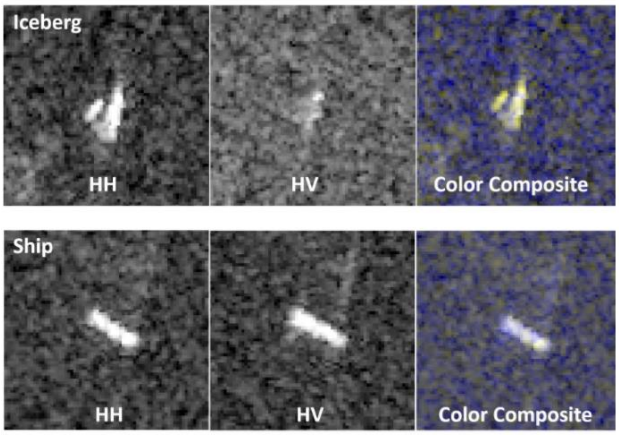

Figure 1: Examples of Iceberg and Ship

The solution was motivated from the idea of boosting, which is combining multiple "weak classifiers" into a "strong" one. At first, we decide to use SVM and CNN models to generate "weak classifiers". To get multiple classifiers, we will tweak the parameters and change the inputs by pre-processing the input data. In the Project Accomplishment Part, will explain the three algorithms we experimented with, and how we pre-processed the data based on "incident angles".

II. PROJECT ACCOMPLISHMENT

A. SVM

Support Vector Machine (SVM) is a relatively new supervised classification technique originally motivated by advances in statistical learning theory. It is usually used to analyze data for classification and regression. It first applies a transformation of the initial input space into a new potentially very high dimensional transformed input space where classes are very likely to be linearly separable and then deriving a hyperplane to separate each pair of classes in this transformed input space. In addition to performing linear classification, SVM can perform a nonlinear classification using kernel trick. Here we use Keras to create SVM model and separate 80 percentage of data as training data and the rest of them as test data.

First, we combine two-channel data into one image which size is 75*75*2, and add another channel which is the average of two-channel data as the third channel. Then we use an image which size is 75*75*3 as each data. Using the method in the class, we choose Keras "svm" function to train the model

directly, which kernel is 'rbf', gamma is 0.01. And get 53% accuracy, which is not good. This might be caused the dataset, because most of images are similar, and it's hard for human to tell which is the ship or iceberg. When we process the whole image, we consider the background of the image, which takes a big proportion and is not helpful for the model. So we plan to reprocess the data before training.

Here we select Histogram of Oriented Gradients (HOG) Descriptor to process the image. HOG is a feature descriptor used to detect objects in image processing. It counts occurrences of gradient orientation in localized portions of an image-detection window, or region of interest. The algorithm's implementation is that at first, we divide the image into cells and for each cell compute a histogram of gradient directions for the pixels within the cell. Then discretize each cell into angular bins and compute its weighted gradient. Finally, we normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor we want.

After reprocessing data, we build a new svm model. The procedure is first to combine two-channel data into a three-dimensional image and create a data frame. And then normalize data frame, calculate feature vectors using HOG method. Finally, train the model and test it. The accuracy we got is 60%. Compared with the first version without image processing, we improve 7%. However, it is still not good. The reason might be that the image dataset itself is not suitable for SVM and also SVM is not the best method for image binary classification problem. There exists some improvement to create a better SVM model and we will try it in the future. Next step, we try to use CNN and Boosting method to see if we can get a better accuracy.

B. **CNN**

Convolutional Neural Networks are a special kind of multi-layer neural networks. [1] They are often used in image recognition systems. A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers and then followed by one or more fully connected layers. One CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

Our CNN model contains eight layers: four conv layers, three fully connected layers.

The CONV layer computed the output of neurons that are connected to local regions in the input. We applied RELU as activation, the non-saturating activation function. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

All conv layers are followed by one pool layer and one dropout layer.

The POOL layer will perform a down sampling operation along the spatial dimensions (width, height). We chose max pooling, which uses the maximum value from each of a cluster of neurons at the prior layer.

The DROPOUT is easily implemented by randomly selecting nodes to be dropped-out with a given probability (20%) each weight update cycle.

The four conv layers are generally same except the number of filters, so the output sizes are different. Our input size is (75,75,3), the first conv layer outputs data in size (36,36,64), the second is (17,17,128), the third is (7,7,128) and the last one is (2,2,64).

A FLATTEN Layer is followed by three FULLY CONNECTED layers. Flatten layer is aim to flat the output data, so it can be processed in fully connected layer. The output data size from the flatten layer is 256, and data after the first dense layer(fully connected layer) is in size 512.

Both of the first two fully connected layers are followed by RELU layer and DROPOUT layer. They are doing the same job as they did in conv layers, RELU chose all positive data and dropout dropout 20% of nodes.

The last fully connected layer is followed by one SIGMOID layer. Sigmoid function produces outputs between 0 and 1, and the outputs are the final probability of their being icebergs.

At last, because only training data has labels for us to train and test, we used 75 percents of training data to train the model and 25 percents to test, after 40 epochs, the test accuracy is around 85%.

C. **Boosting**

One heuristic approach is "Boosting". Generally speaking, it is about combining many "weak" classifiers into a "good" one. There are some classifiers are build based on this idea, yet none of them, as far as a concerned, combining the CNN. When we think about this approach, we consider one pre-trained CNN as one classifier, and we trained many such classifiers in a different way. After we got a set of classifiers, we can combine those classifiers with the idea of Boosting. We might get a better result by doing so.

Suppose we have a set of classifiers, and there must be a greatest classifier among those. However, even though it is the greatest we have right now, it still makes mistakes. Let's denote the best classifier as C1, the second best C2, etc. The idea is to combine them by giving each classifier a vote. Assume we only let the top 3 classifiers vote because the others are really stupid classifiers. Any sample is classified as an iceberg, only if 2 or more than 2 classifiers said it is an iceberg. It is a naive mimic of democracy.

This heuristic sounds very effective, yet it has the same problem as democracy, what if most people vote for a bad result? Let's draw a Venn's Diagram of the sample space and the errors to demonstrate this problem.

The black box is the whole sample space, and the red circle is the errors made by C1, as showed in Fig.2. And then, we add the other two classifiers, as showed in Fig.3.

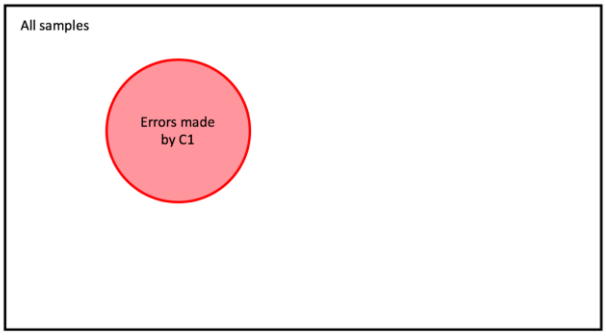

Figure 2

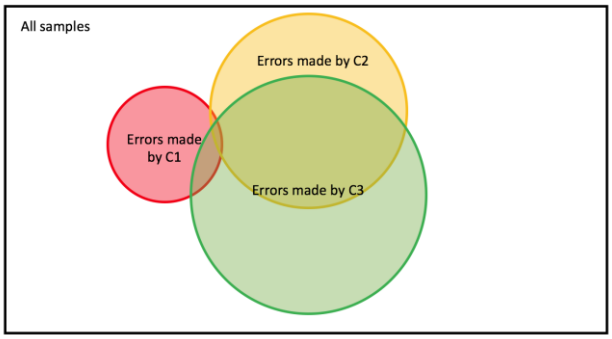

Figure 3

We can see that, if a sample fell into the area where C2 and C3 both made an error, the result is even worse than just have C1 as a "dictator", because that area is larger than the red circle. Therefore, there is a much better and a slightly sophisticated way to elect the representatives.

The idea is to elect some classifier that make the fewest error that have made by the previous classifier. This includes a new variable called "error weight". Each sample correspond to its own error weight. Initially, all the sample have the same error weight, which is 1/total_number_of_samples.

The error rate of each classifier is:

$$\sum_{all\ the\ error\ it\ makes} error\ weights$$

For example, if the best classifier only makes 5 errors, then its error rate is 5/total_number_of_samples. Very intuitive.

Before we pick the second best classifier, we must rescale the weights as the following:

$$\sum_{C1\ correct} error\ weights = ½$$
$$\sum_{C1\ error} error\ weights = ½$$

Since C1 should never make more error than correct prediction, the error weights of the error samples will grow larger. Suppose there is a classifier that we make the same errors as C1 did, then its error rate calculated here is 50%. Same as tossing a quarter, which is really bad.

Of course, every classifier is chosen if it has the smallest error rate. With this mechanism, all the chosen classifier should have very small overlaps in their errors.

However, this is not enough. What if the best classifier is much better than all the others? Should we give them the same amount of votes? The answer is no, and there is a coefficient for every classifier chosen. Denote as 'a': $a = \frac{1}{2}\ln\frac{1-error\ rate}{error\ rate}$

For a binary classification problem, if we denote two result as -1 and +1, then

Result = sign(a1*C1 + a2*C2 + a3*C3 + …)

D. Pre-processing Data

Now that we have a model for boosting, where should we find so many different classifiers? We used to suggest that we should pick different kind of classifiers like we combine SVM, CNN and Decision Tree. Yet, the accuracy for SVM and Decision Tree is too far from CNN, which might lead to the problem we discussed above.

Finally, we choose to train the same model with 8 different inputs, thus the 8 models will have different parameters despite they have the same structure. The first 4 different inputs are:

Band_1
Band_2
Band_3 = (Band_1 + Band_2) / 2
Band_4 = Band_1 – Band_2

The reason we do this is that Band_1 and 2 are different because the are received perpendicular to each other. Different material can reflect differently in Band_1 and Band_2.

Other way to create new inputs is by using the "inc_angle" column in the dataset. The way we use inc_angle is based on this description: "Generally, the ocean background will be darker at a higher incidence angle." Therefore, since we have 4 inputs generated with the original band1 and band2, we can generate 4 more with some pre-processing, with an approach we referred to as a "background dimmer".

The idea is this: if the incidence angle is low, we will make the background darker than it was. We designed a mathematical model that will do this job.

Firstly, how does the computer know which pixel is the background. Well, the background should be darker than the object. It is always nice to set some threshold

$$Threshold = pixel\_\max - \lambda \cdot pixel\_var$$

Where pixel_max is the pixel with max value in this sample, and pixel_var is this sample's variation. Lambda can be any value, it can even be a trainable parameter if we can find the corresponding loss function and plug it in. For now, we just set it to be 3. Every pixel smaller than the threshold will be considered as background.

Now, what about the "Dimmer Function"? The idea is dim more of the background, if the inc_angle is small and if that background is a "local maxima" or a little bump in the figure. Basically, it just smooth out the background based on the inc_angle value. We designed a math model that looks like this:

$$Pixel = Pixel - \alpha * (\cos(inc\ angle) - \cos(inc\ angle\ max)) * \frac{distance\ between\ \min\max Pixels}{Pixels_{\max - Pixel}} * Pixels_{var}$$

The alpha is similar to lambda, it can also be treated as a trainable parameter if we found the loss function. Here, we set it to 0.01.

This model may not be true, yet it sure provide a math tool for us to pre-process the data in a way that adapt to that description, “Generally, the ocean background will be darker at a higher incidence angle.” It is turning human language into mathematical language. It might seems a naïve model, yet it proved to be useful.

Now, with this dimmer, we will be able to process Band 1 and 2, and do the same as band 3 and band 4. In total, this will give us four more different inputs, thus four more different models.

III. Summary

Our purpose is to distinguish whether the target is a ship or an iceberg. The SVM accuracy is around 59%, according to the experiment. While the CNN models have accuracy around 85%. (The accuracy may vary depend on training data size, but in our experiment, we used the same parameters for those to ensure coherence). Therefore, we decide only use CNN models and combining them with boosting.

The result is not what we fully expected. The best classifier did pretty good 90% accuracy. While the others not very good and the coefficient calculated proved that, those classifiers can hardly be combined. The good news is, that the background dimmer is a success. With it, we improved the accuracy by 5%.

The idea of combining boosting was not a total mess. It might be a powerful tool if we can construct a classifier set that is large and varies. This model can also be used to test any new ideas on any kind of classifier. We can explore how much did classifiers overlaps in their errors. In conclusion, we create a model that is suitable to this particular kind of problem and this boosting heuristic should always be an option to solve classification problems, with a well-constructed classifier set.

Also, github code is showed in [2].

References

[1] LeCun, Yann. "LeNet-5, convolutional neural networks". Retrieved 16 November 2013.
[2] GitHub repository https://github.com/yz3358/ml_project