Capstone Project

Youssef Mubarak

Machine Learning Engineer
Nanodegree

October 09, 2018

Statoil: Iceberg Classifier

I. Definition

Project Overview

Statoil, an international energy company operating worldwide, has worked closely with companies like C-CORE which has been using satellite data for over 30 years and has built a computer vision based surveillance system. To keep operations safe and efficient, Statoil is interested in getting a fresh new perspective on how to use machine learning to more accurately detect and discriminate against threatening icebergs as early as possible.

The remote sensing systems used to detect icebergs are housed on satellites over 600 kilometers above the Earth. The Sentinel-1 satellite constellation is used to monitor Land and Ocean. Orbiting 14 times a day, the satellite captures images of the Earth's surface at a given location, at a given instant in time. The C-Band radar operates at a frequency that "sees" through darkness, rain, cloud and even fog. Since it emits its own energy source it can capture images day or night. Satellite radar works in much the same way as blips on a ship or aircraft radar.

When the radar detects an object, it can't tell an iceberg from a ship or any other solid object. The object needs to be analyzed for certain characteristics - shape, size and brightness - to find that out. The area surrounding the object, in this case ocean can also be analyzed or modeled. Many things affect the backscatter of the ocean or background area. High winds will generate a brighter background. Conversely, low winds will generate a darker background.

Some images of these objects can be visually classified. But in an image with hundreds of objects, this is very time consuming. Also, some images are challenging to identify, so this process needs to be automated using machine learning techniques given satellite imagery to identify icebergs and differentiate them form ships.

Many different academic researches have been made to address the problem of "Ship-Iceberg Discrimination", using Satellite Radar Imagery (SAR) but often with higher resolution images [1][2]. So, working on this satellite imagery is expected to be challenging.

---

[1] https://media.asf.alaska.edu/uploads/News%20and%20Notes/documents/asfnn_7-2.pdf
[2] https://elib.dlr.de/99079/2/2016_BENTES_Frost_Velotto_Tings_EUSAR_FP.pdf

Problem Statement

Drifting icebergs present threats to navigation and activities in areas such as offshore of the East Coast of Canada. Currently, many institutions and companies use aerial reconnaissance and shore-based support to monitor environmental conditions and assess risks from icebergs. However, in remote areas with particularly harsh weather, these methods are not feasible, and the only viable monitoring option is via satellite.

So, we're challenged [3] to build an algorithm that automatically identifies if a remotely sensed target is a ship or iceberg from satellite imagery. Improvements made will help drive the costs down for maintaining safe working conditions

Mainly, we will be using Convolutional Neural Networks (CNNs) to address the problem of classification, where the output is a binary label representing the image object; 1 for an iceberg and 0 for a ship. We will apply different techniques such as transfer learning and data augmentation to achieve better results.

Metrics

As the data is approximately balanced and it doesn't show any significant imbalance in class sizes (53% ships and 47% icebergs); a high accuracy will indicate high classification ability. So, we will mainly use ***Accuracy*** as an evaluation metric.

$$Accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

We will also make use of classification report that shows the *precession*, *recall* and *f1-score*, however, to seek a good balance between precession and recall we will focus on *f1-score*.

$$f1 - score = 2 * \frac{precision\ .\ recall}{precision\ +\ recall}$$

So these are the metrics that will be used to quantify the performance of our solution model.

---

[3] This problem was presented in a Kaggle competition and you can find it here.

II. Analysis

Data Exploration

The dataset [4] is obtained as satellite imagery and the labels were provided by human experts and geographic knowledge on the target. All the images are 75x75 images with two bands.

The data is presented in *json* format as *'train.json'* and *'test.json'*. The training data size is 1604, where the testing data size is 8424.

Data fields

The files consist of a list of images, and for each image, you can find the following fields:

	band_1	band_2	id	inc_angle	is_iceberg
0	[-27.878360999999998, -27.15416, -28.668615, -...	[-27.154118, -29.537888, -31.0306, -32.190483,...	dfd5f913	43.9239	0
1	[-12.242375, -14.920304999999999, -14.920363, ...	[-31.506321, -27.984554, -26.645678, -23.76760...	e25388fd	38.1562	0
2	[-24.603676, -24.603714, -24.871029, -23.15277...	[-24.870956, -24.092632, -20.653963, -19.41104...	58b2aaa0	45.2859	1
3	[-22.454607, -23.082819, -23.998013, -23.99805...	[-27.889421, -27.519794, -27.165262, -29.10350...	4cfc3a18	43.8306	0
4	[-26.006956, -23.164886, -23.164886, -26.89116...	[-27.206915, -30.259186, -30.259186, -23.16495...	271f93f4	35.6256	0

- **id** - the id of the image
- **band_1, band_2** - the flattened image data. Each band has 75x75 pixel values in the list, so the list has 5625 elements. Note that these values are not the normal non-negative integers in image files since they have physical meanings - these are float numbers with unit being dB. Band 1 and Band 2 are signals characterized by radar backscatter produced from different polarizations at a particular incidence angle. The polarizations correspond to HH (transmit/receive horizontally) and HV (transmit horizontally and receive vertically). More background on the satellite imagery can be found here.
- **inc_angle** - the incidence angle of which the image was taken. Note that this field has missing data marked as "na", and those images with "na" incidence angles are all in the training data to prevent leakage.
- **is_iceberg** - the target variable, set to 1 if it is an iceberg, and 0 if it is a ship. This field **only** exists in train.json.

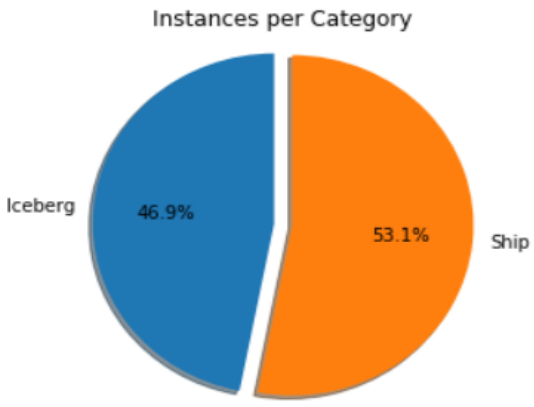
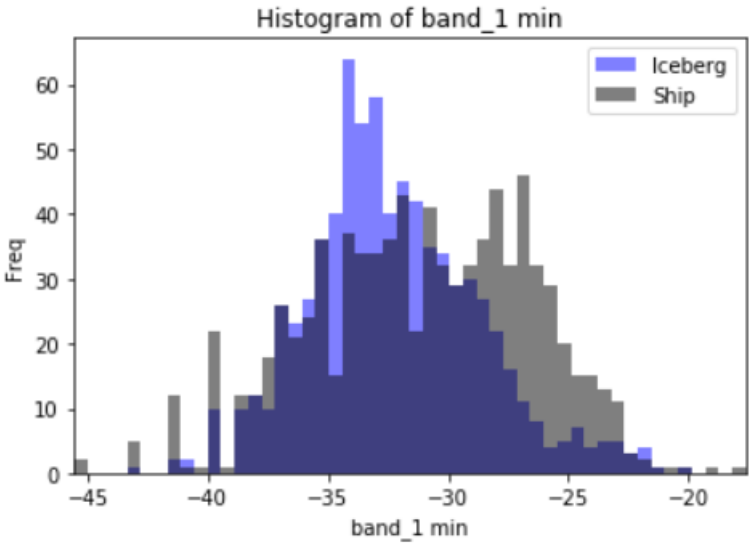


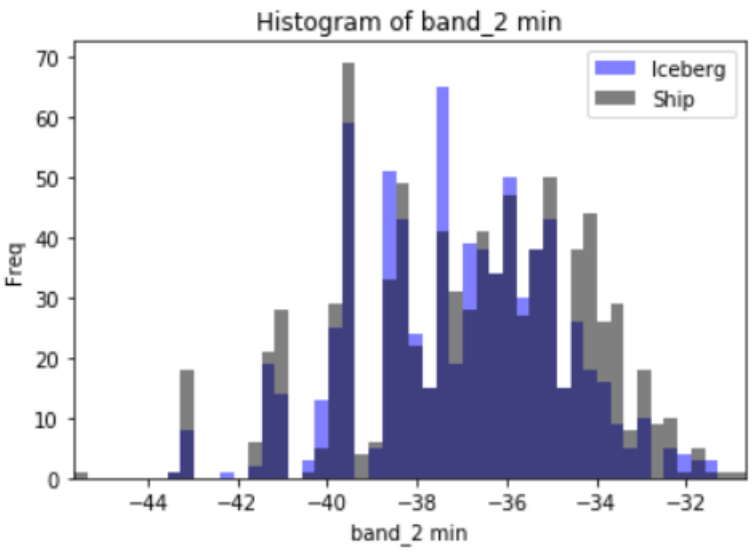
Figure 1: Percentage of Class size

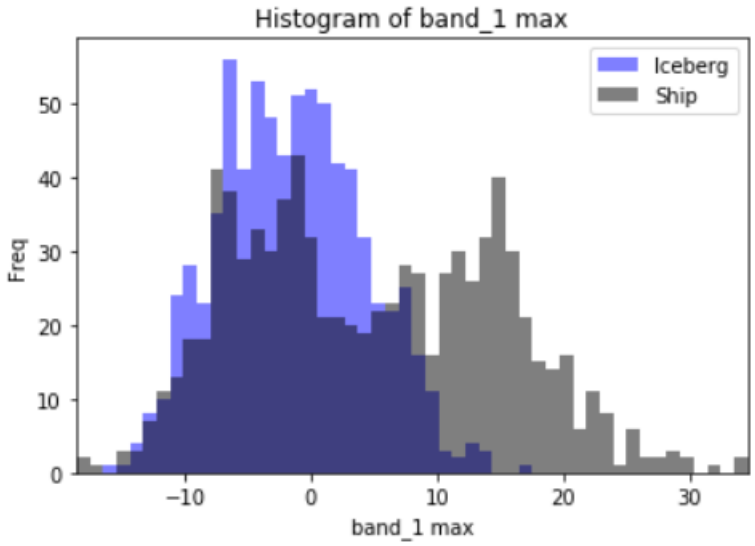
The training data has 53% ships and 47% Icebergs where the ships has only 98 more images, so the data can be considered balanced.

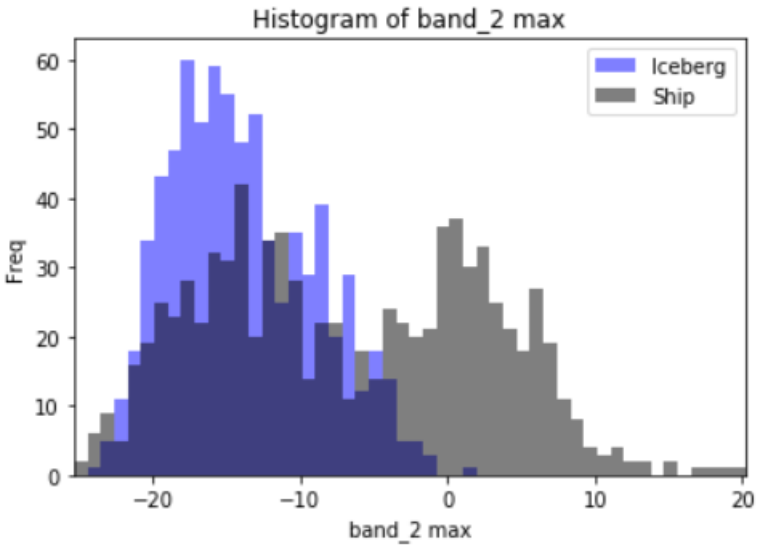
There are 133 missing data in `*inc_angle*` out of 1604 entries (that is 8.3% of data) where the other columns had no missing values.

To explore features further, regular aggregations per band will be considered such as minimum, maximum, mean, and standard deviation:



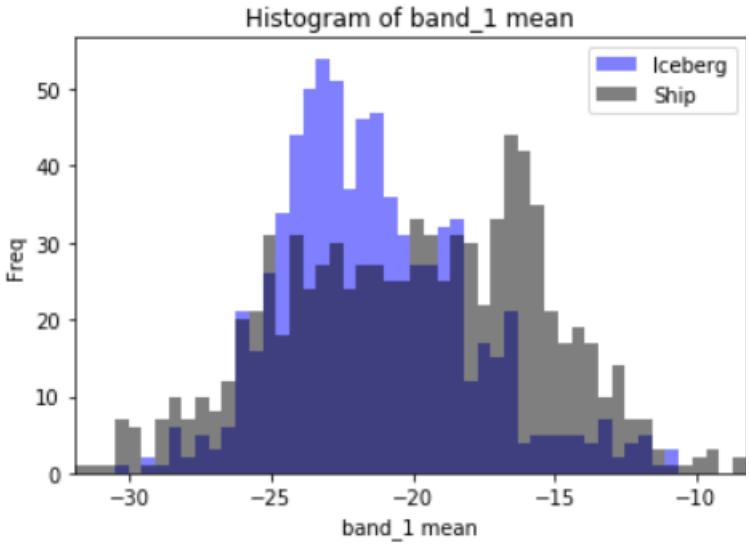


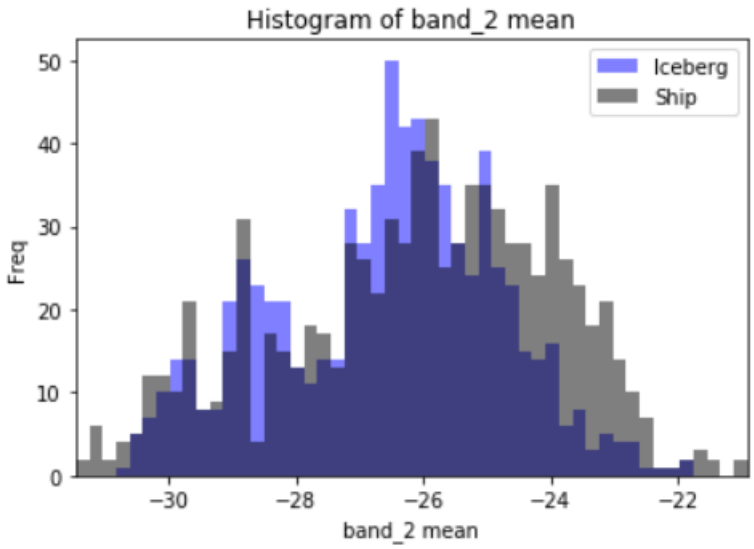


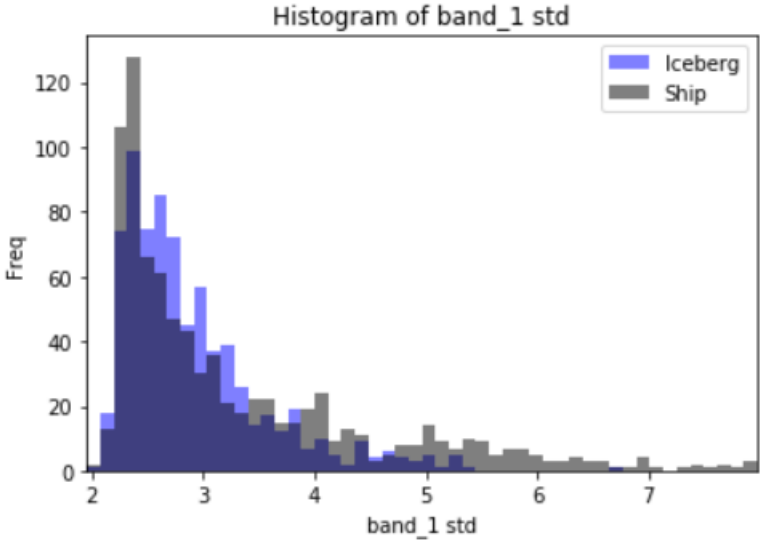


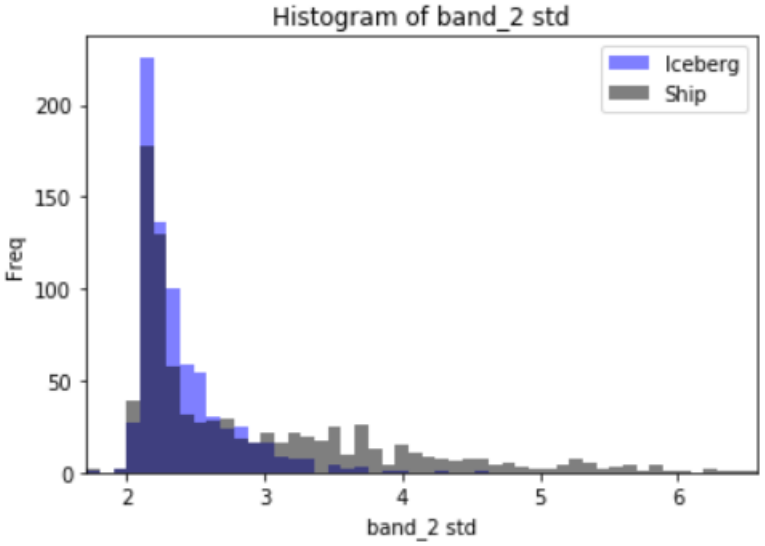
---

[4] The dataset can be found here: https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/data









As we can see from the above histograms for different aggregation signals, both bands of different polarization; band_1 (HH: transmit/receive horizontally) and band_2 (HV: transmit horizontally and receive vertically) exhibit approximately similar results most of the time.

Exploratory Visualization

When we view sample images in different bands, where each band_1 is aligned with its corresponding band_2, we see the following:

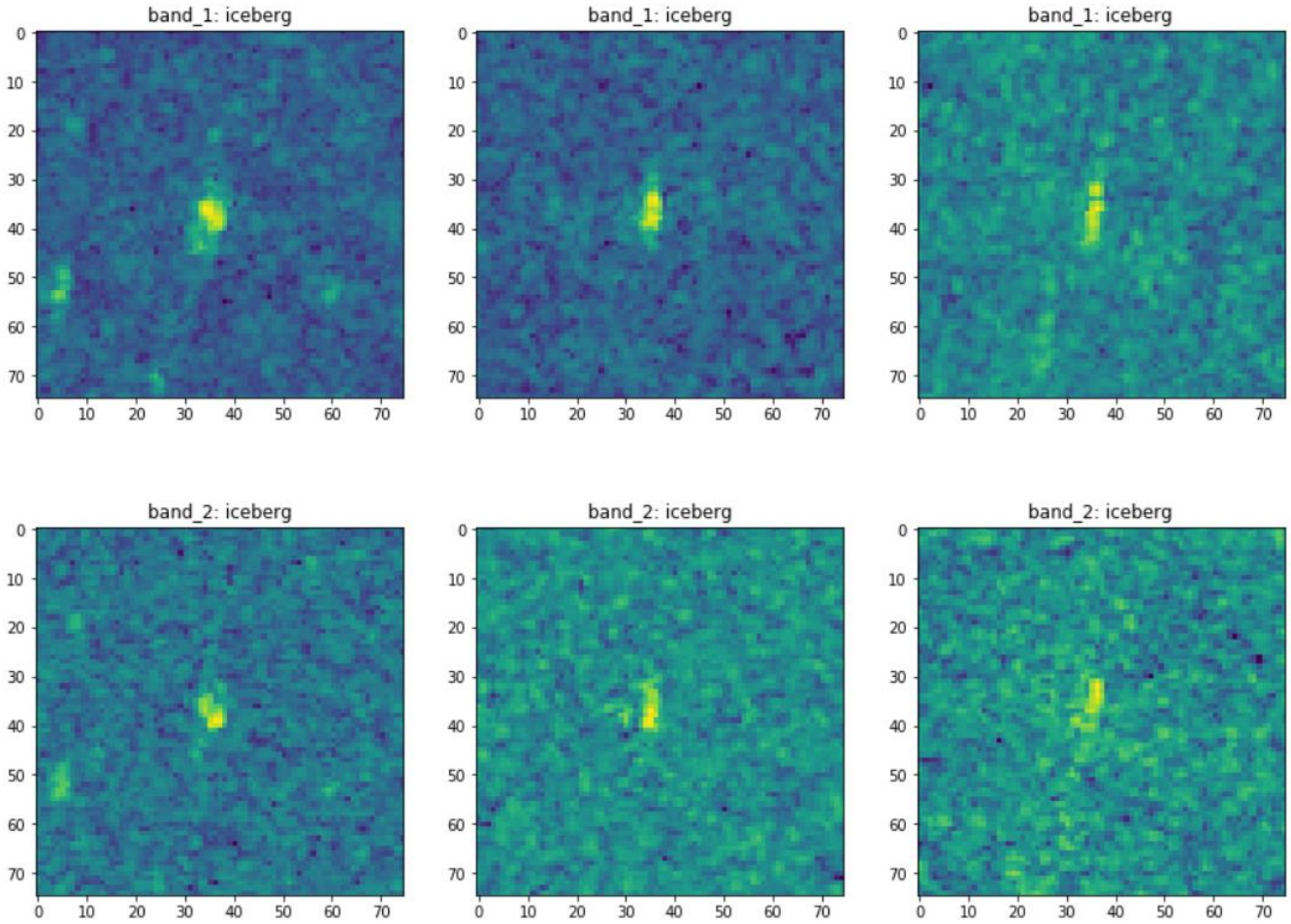


Figure 2: sample images of icebergs in band_1 and band_2

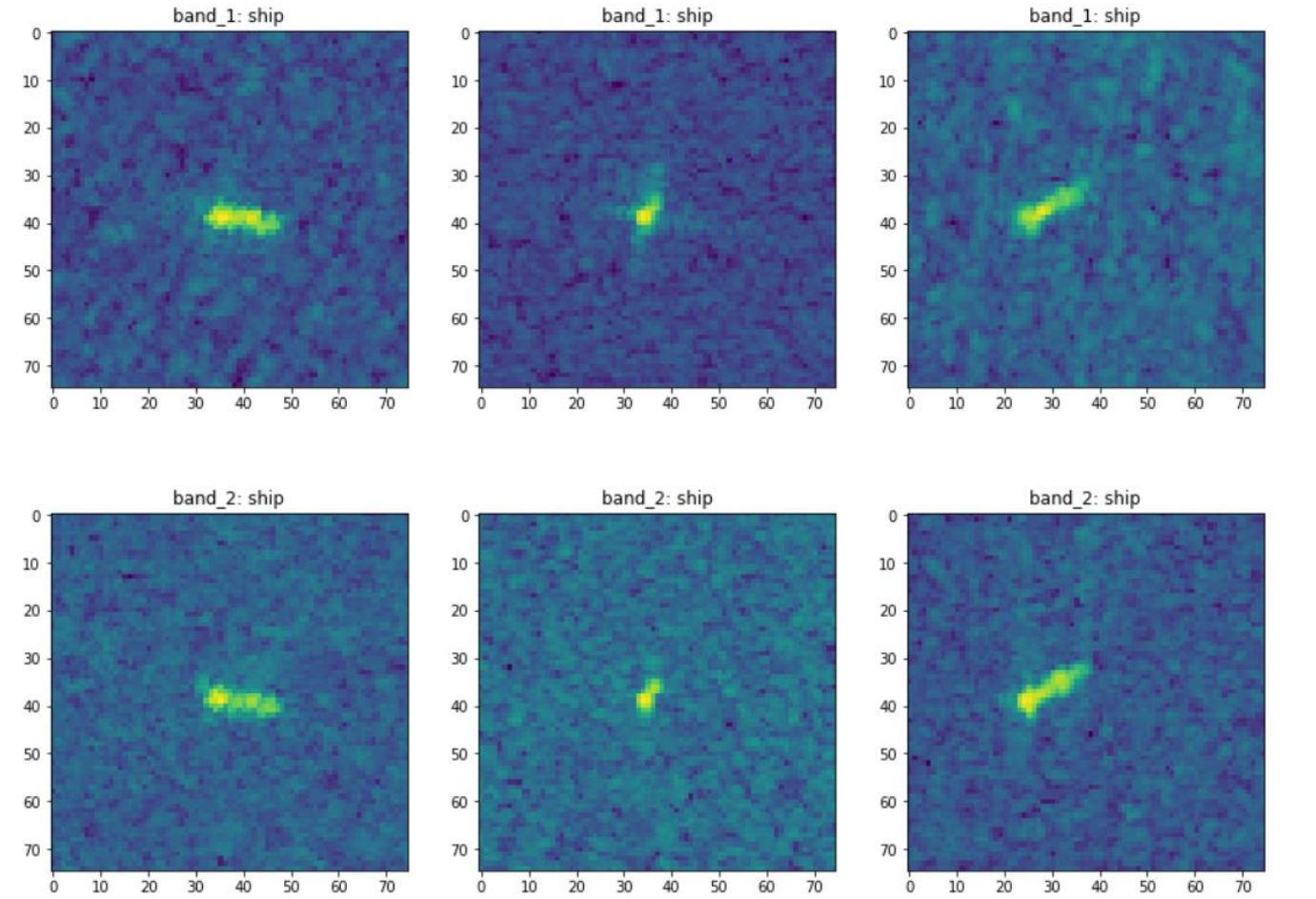


Figure 3: sample images of ships in band_1 and band_2

There are subtle differences between the images that we can be hard to spot. However, we will use both bands to construct an image with two channels with shape 75X75X2.

Also, if we compare images from figures 2 and 3, in some examples it's hard to tell whether it's a ship or an iceberg, so, our model needs to be complex enough to capture the unseen differences.

Algorithms and Techniques

Feature Extraction and PCA

Each data band is 75X75 pixels, most of which doesn't contain any parts of objects (ships or icebergs). We want to extract important features and reduce the dimensionality of the data and project it to a lower dimensional space using Principal Component Analysis (PCA), which is a dimension-reduction tool that can be used to reduce a large set of variables to a small set that still contains most of the information in the large set. It transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. It allows us to set the parameter ' *n_components* ' [5] that defines the number of components to keep.

After this we will apply different classifiers on the reduced dataset and see how they behave:

- Random Forest: A classifier that fits a number of decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The parameter '*n_estimators'* defines the number of trees in the forest.
- K-nearest neighbors : one of the simplest machine learning image classification algorithms. Simply, an object is classified by a majority vote of its neighbors, that is, it classifies unknown data points by finding the most common class among the k-closest examples. The parameter '*n_neighbors*' defines the number of neighbors to use by default for k- neighbors queries.
- Logistic Regression: A statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome, that is, a binary dependent variable. Mathematically, a binary logistic model has a dependent variable with two possible values, where the two values are labeled "0" and "1". The goal of logistic regression is to find the best fitting model to describe the relationship between the binary characteristic of interest (outcome variable) and a set of independent variables.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are the state-of-the-art algorithms for most image-processing tasks, including classification. CNN can be a powerful model that can develop a kind of awareness of the spatial information and complex patterns in input images, as it takes an image matrix rather than a vector (like in regular Multi-layer Perceptrons). The algorithm takes and input image and outputs an assigned probability for each class. As this is a binary classification problem, it outputs a single probability that's closer to one if an object is an iceberg, and closer to zero if an object is a ship. So, we will make use of CNNs to build our main models. Here, we have many parameters to set and tune like: the number of layers, the number of nodes per layer, the layers types, and the number of epochs the model will be trained for.

Data Augmentation

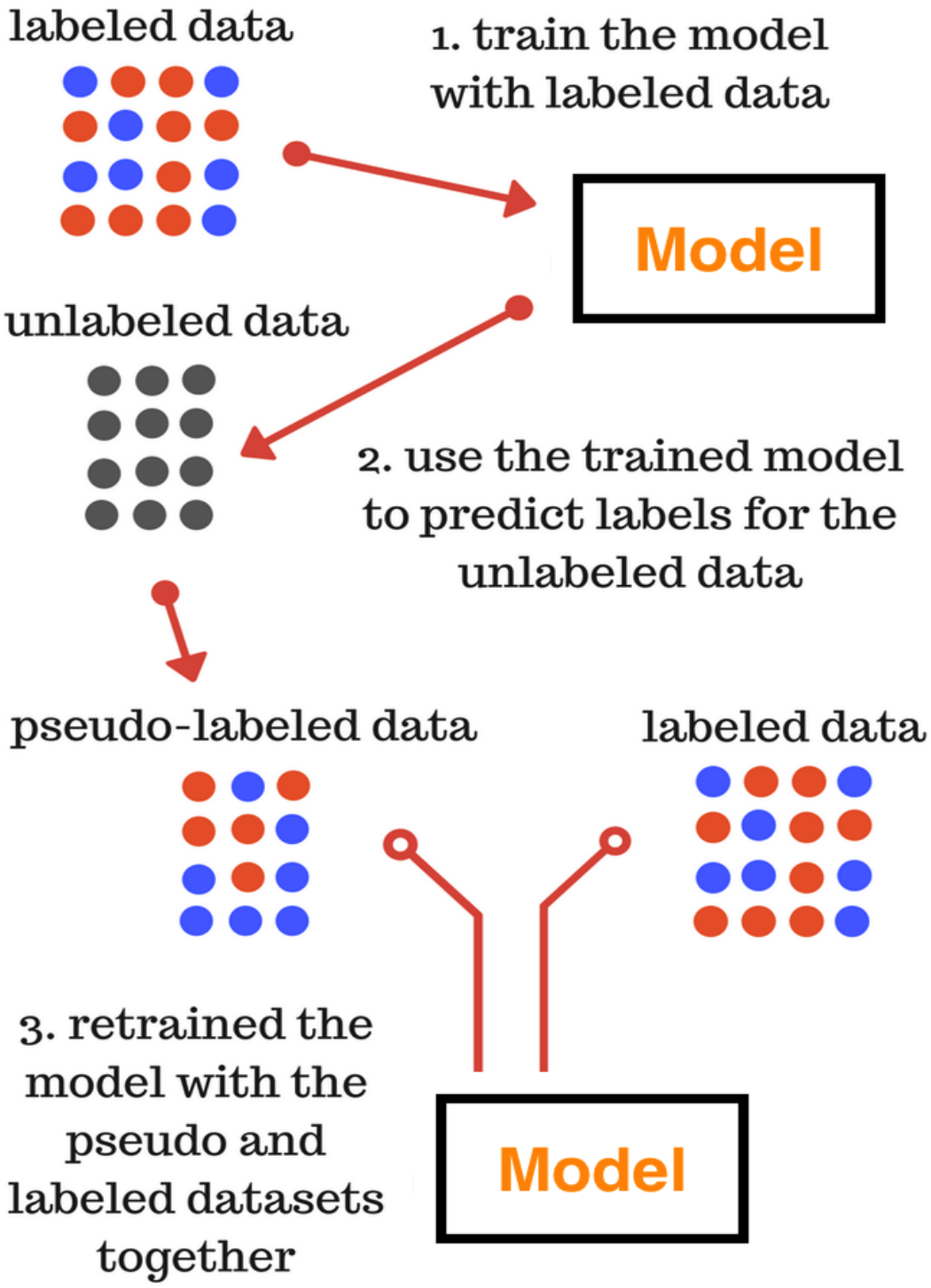
As the training data is small, we will make use of data augmentation [6]. A technique used to introduce more instances of the images and making the model less prone to over-fitting on certain views, angles, positions and sizes of the object of interest found in the image. This can be done by different means, like shifting, flipping or rotating the images. Such parameters [7] can be manipulated to generate different results.

Transfer Learning

It can be a very helpful learning technique when a model trained on one task is re-purposed on a second task. These pre-trained models have learned so much about how to find patterns in image data which we can make use of. A pre-trained source model is chosen from available models [8], this pre-trained model acts like a feature extractor where the last layers are replaced with new ones to adapt the architecture to our own classification task instead of constructing a CNN from scratch.

Pseudo-Labeling

As the training data is relatively small compared to the unlabeled testing data we will apply a semi-supervised learning technique; *pseudo-labeling* [9]. The idea is to increase the training data and make use of the large unlabeled testing data. So, part of the testing data is fed into a model which predicts the data labels. These pseudo-labeled data is then combined with the original data, and the final model is retrained on it. (See illustration below)



The process of pseudo-labeling

---

[5] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
[6] https://towardsdatascience.com/paper-summary-the-effectiveness-of-data-augmentation-in-image-classification using-deep-9160dc87806b
[7] https://keras.io/preprocessing/image/#imagedatagenerator-class
[8] https://keras.io/applications/
[9] https://www.analyticsvidhya.com/blog/2017/09/pseudo-labelling-semi-supervised-learning-technique/

Benchmark

The benchmark model will be a simple CNN classifier. Then we will try to improve the model performance by different means and then compare the results. We will also consider submitting the final solution to the Kaggle competition to show where it stands among the scores in the competition leaderboard.

III. Methodology

Data Preprocessing

Handling missing values

As we mentioned earlier, there are 133 missing data in `*inc_angle*` out of 1604 entries (that is 8.3% of data) where the other columns had no missing values. So we decided to drop this column.

Concatenating bands

The flattened image data have two bands. Each band has 75x75 pixel values in the list, so the list has 5625 elements represented as floats.

These bands are converted to numpy arrays with the data type being 32-bit float, and then reshaped to 75X75 pixels. These bands are then concatenated where each of which represent a channel in the final image that has a shape of 75X75X2.

Implementation

As the testing data is unlabeled we considered the training data as our main input. So, whenever we say "data/dataset" we mean the originally training data in '*train.json'*.

Dimensionality reduction with PCA

We applied dimensionality reduction on the data with PCA where n_*components*=50, and then the resultant reduced data set was split into 75% training data and 25% testing data.

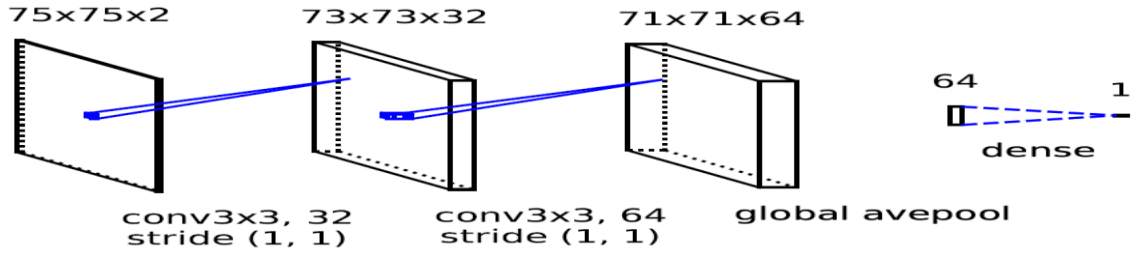
A Random Forest model was trained with *n_estimators*=10, and a K-NN model was trained with *n_neighbors*=3. Finally, we trained the Logistic Regression model.

These methods result in 71%, 74%, and 72% accuracies, respectively.

CNNs

The original dataset (data pre-reduction) was split into 75% training data and 25% testing data. This corresponds to 1203 training samples and 401 testing samples.

After that, a simple CNN model was built as a benchmark. Below is the basic model architecture:




```
X_train shape: (1203, 75, 75, 2)
1203 training samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 73, 73, 32)        608
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 71, 71, 64)        18496
_________________________________________________________________
global_average_pooling2d_2 ( (None, 64)                0
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 65
=================================================================
Total params: 19,169
Trainable params: 19,169
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Benchmark model architecture

The model was compiled with the following parameters:

- **optimizer**: Adam, **loss**: binary cross-entropy, **metrics**: accuracy

A helper function *train_with_kfold (model, checkpoint_path, epochs=50, K=4, batch_size=None)* was defined to train models with Stratified K-Fold. It takes the model we want to train, the checkpoint to which we want to save the model weights, the number of epochs, the number of folds (K), and the batch size. Finally, it returns the model history.

The basic model was trained with K=2 and number of epochs=150. It results better results than the previous methods with ~83% accuracy, 0.35 loss, and about 0.88 f1-score for each class.

Transfer Learning and Data augmentation

The next step was to apply transfer learning together with data augmentation, so we used the pre-trained model **VGG16** from **keras** [10]. The pre-trained VGG16 model takes an input image with three

dimensions where the 3rd dimension corresponds to the RGB three channels. As our input has only two channels we added a new on by averaging the two bands: band_1 and band_2:-

- 1st channel: band_1
- 2nd channel: band_2
- 3rd channel: $\frac{\text{band\_1} + band\_2}{2}$

The modified data was split into 75% training and 25% testing sets. To augment the training data we used Keras *ImageDataGenerator* [11] to apply image transformations, like horizontal and vertical flips, width and height shifts, and random rotations.

The VGG16 model's last layers were dropped and new fully-connected layer were added to suit our own classification problem. Below is a summary of the modified VGG16 architecture:

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 75, 75, 3)	0
block1_conv1 (Conv2D)	(None, 75, 75, 64)	1792
block1_conv2 (Conv2D)	(None, 75, 75, 64)	36928
block1_pool (MaxPooling2D)	(None, 37, 37, 64)	0
block2_conv1 (Conv2D)	(None, 37, 37, 128)	73856
block2_conv2 (Conv2D)	(None, 37, 37, 128)	147584
block2_pool (MaxPooling2D)	(None, 18, 18, 128)	0
block3_conv1 (Conv2D)	(None, 18, 18, 256)	295168
block3_conv2 (Conv2D)	(None, 18, 18, 256)	590080
block3_conv3 (Conv2D)	(None, 18, 18, 256)	590080
block3_pool (MaxPooling2D)	(None, 9, 9, 256)	0
block4_conv1 (Conv2D)	(None, 9, 9, 512)	1180160
block4_conv2 (Conv2D)	(None, 9, 9, 512)	2359808

Figure 5.1: Modified VGG16 model architecture

```
block4_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
global_max_pooling2d_6 (Glob (None, 512)               0
dense_45 (Dense)             (None, 128)               65664
dense_46 (Dense)             (None, 64)                8256
dense_47 (Dense)             (None, 1)                 65
=================================================================
Total params: 14,788,673
Trainable params: 14,788,673
Non-trainable params: 0
```

Figure 5.2: Modified VGG16 model architecture

The model was compiled with the *Adam* optimizer and the *binary cross-entropy* as a loss function, and trained on three folds (K=3) with 50 epochs each. With the default learning rate of 0.001 the model wasn't learning at all so we increased it a little bit to e^{-6}. This model didn't achieve significant results; it has an accuracy of 89%, loss of 0.28 and f1-score of 0.88.

To apply the pseudo-labeling technique we needed the best obtained model, so this step was done at the end, after final refinements were made. More details found in the next section.

Complications

Using Kaggle platform and training with the available GPU saved a lot of time, however, in order to generate output files, or save model weights for future use, you have to commit your kernel (running it all from the start to end) in order to get a downloadable output, which, in many times, isn't the same as the one you intended to save (since the code ran for a different round and resulted in a different output). Unless this isn't a critical issue for, you better find another platform.

---

[10] More about Keras pre-trained models and VGG16 model: https://keras.io/applications/#vgg16

[11] https://keras.io/preprocessing/image/#imagedatagenerator-class

Refinement

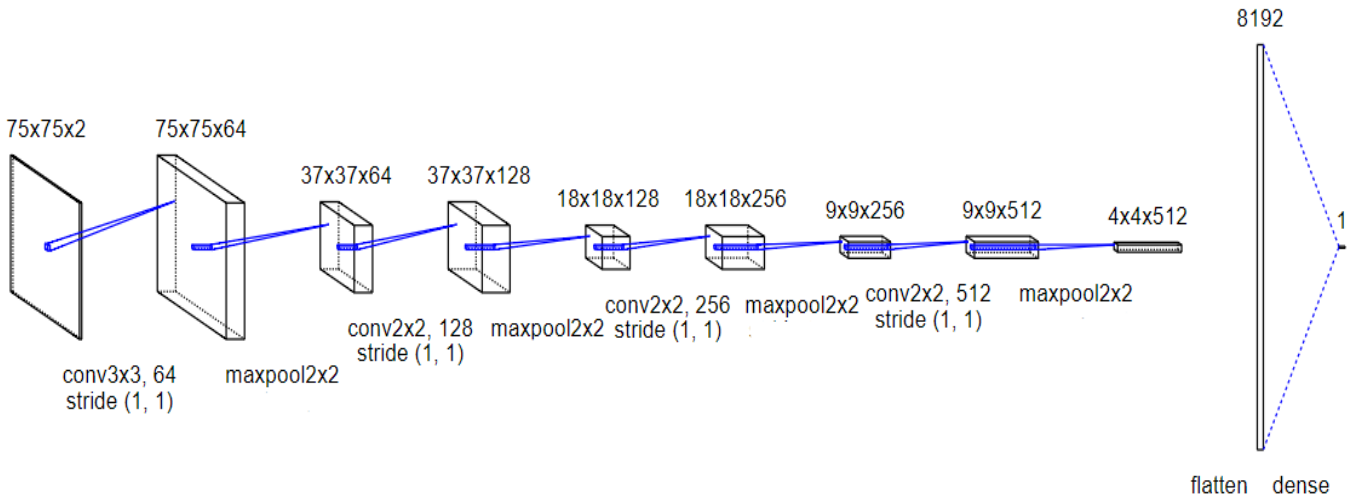
PCA, random forest, K-NN, and logistic regression

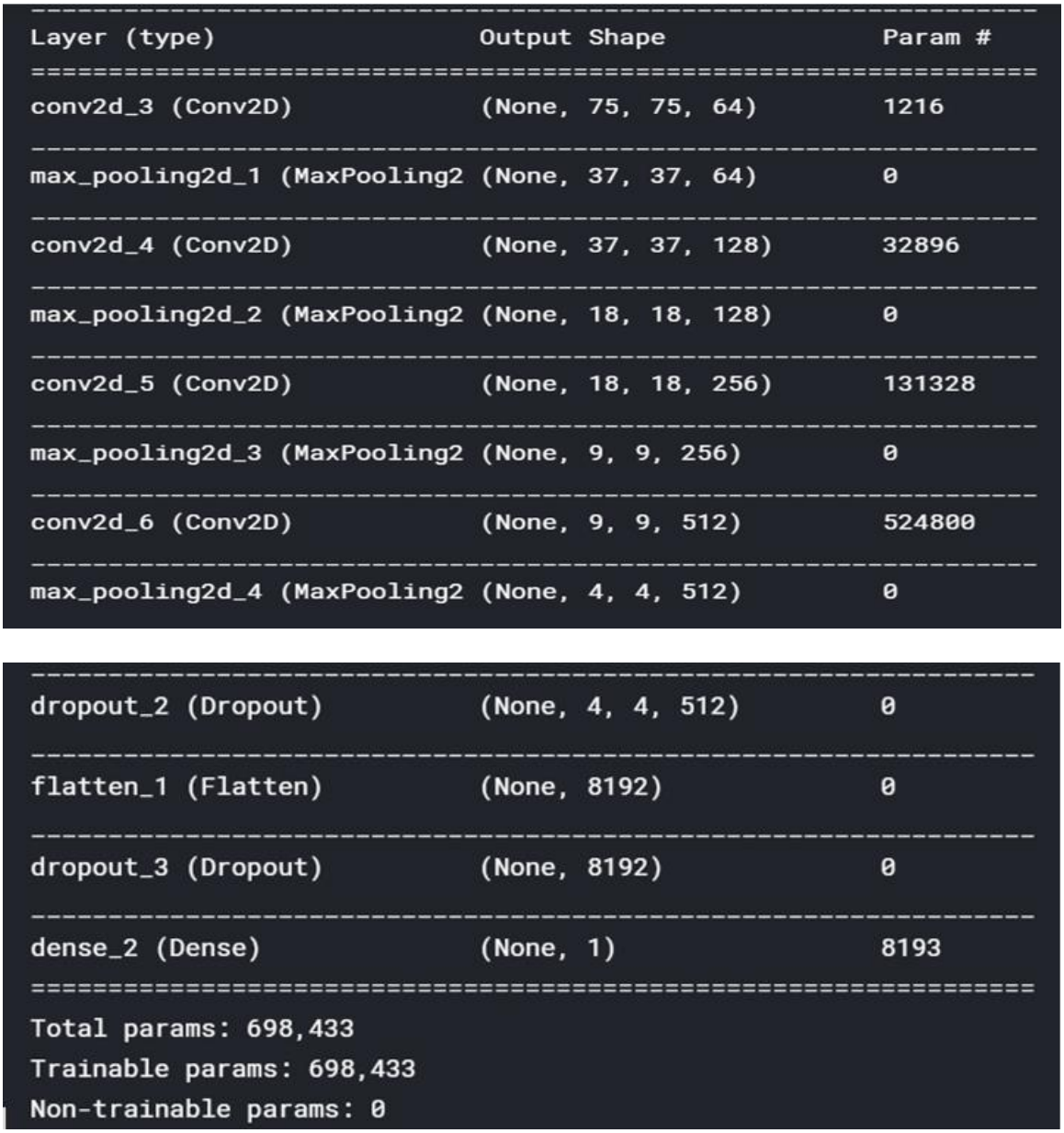
As we first applied dimensionality reductions and tried several classifiers: random forest, K-NN and logistic regression, when tweaking the parameters of these methods, random forest didn't exceed 79% accuracy, K-NN didn't exceed 74% accuracy and logistic regression didn't exceed 72% accuracy.

CNNs

The basic CNN benchmark model easily outperformed the previous methods, so, it was our focus of improvement. We increased the number of epochs and trained the model on more folds and the best results were when the number of epoch is 250 and the number of folds (K) is 4. The model achieved up to 90.77% accuracy, 0.264 loss and f1-score of 0.91 and 0.90 for ships and icebergs, respectively.

To further improve the results, a new improved model was built. We increased the number of layers and made use of *pooling layers* and *dropouts* to reduce overfitting. So the model architecture is as follows:





Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 75, 75, 64)	1216
max_pooling2d_1 (MaxPooling2	(None, 37, 37, 64)	0
conv2d_4 (Conv2D)	(None, 37, 37, 128)	32896
max_pooling2d_2 (MaxPooling2	(None, 18, 18, 128)	0
conv2d_5 (Conv2D)	(None, 18, 18, 256)	131328
max_pooling2d_3 (MaxPooling2	(None, 9, 9, 256)	0
conv2d_6 (Conv2D)	(None, 9, 9, 512)	524800
max_pooling2d_4 (MaxPooling2	(None, 4, 4, 512)	0
dropout_2 (Dropout)	(None, 4, 4, 512)	0
flatten_1 (Flatten)	(None, 8192)	0
dropout_3 (Dropout)	(None, 8192)	0
dense_2 (Dense)	(None, 1)	8193

Total params: 698,433
Trainable params: 698,433
Non-trainable params: 0

Figure 6: improved model architecture

The model had a learning rate of e^{-4} and was trained on three folds (K=3) with 50 epochs each. This model achieved 91.8% accuracy, 034 loss, and f1-scorse of 0.92 for ships and 0.91 for icebergs.

Transfer learning with VGG16

If we looked at the modified VGG16 model we find the following:

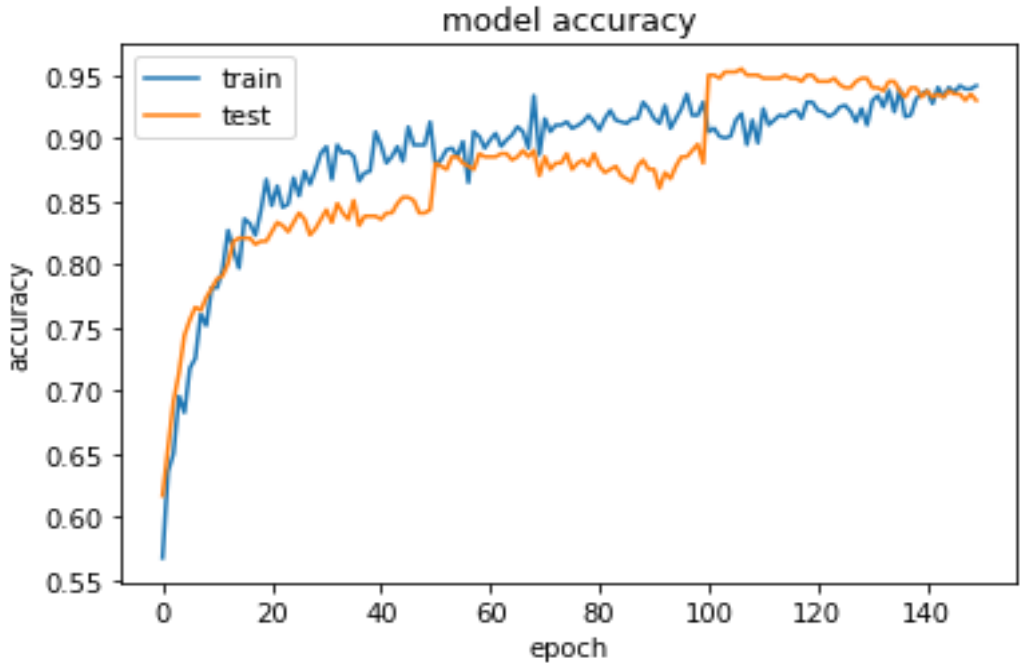


Figure 7.1: modified VGG16 model accuracy over epochs

The model training and validation accuracies were high at the end of the last epochs, but, the model didn't achieve such a high accuracy on the test set, which is a sign of overfitting.

Also, if we looked at the error rate, we see that the model has a slow learning rate:

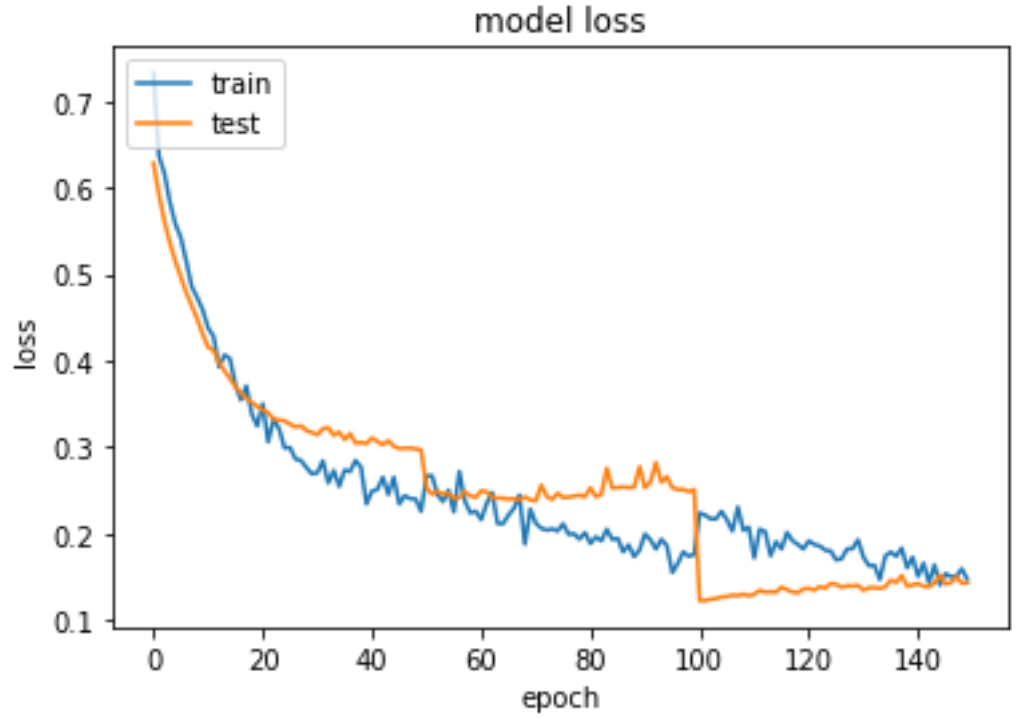


Figure 7.2: modified VGG16 model loss over epochs

We further improved the modified VGG16 model by removing the hidden fully-connected layers to reduce complexity, and using *dropout* layers to reduce overfitting. So the final layers became:

```
_________________________________________________________________
global_max_pooling2d_2 (Glob (None, 512)               0
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 513
=================================================================
```

Instead of:

```
_________________________________________________________________
dense_45 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_46 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 65
=================================================================
Total params: 14,788,673
```

Then the model was compiled with a customized *Adam* optimizer with an increased learning rate of e^{-4}.

The model results improved a little bit with 90.5% accuracy, 0.247 loss, and f1-score of 0.91. However these are not significant results compared to the previous improved models.

Pseudo-labeling

The best obtained model was the CNN *improved model*, and it was chosen to apply pseudo-labeling. In this process, we took part of the unlabeled testing data and get the predicted labels for them by our best model. These labeled testing data was then concatenated with the training data to form a new larger data set which was split into training and testing sets, and the model was retrained on this new data.

To choose a portion of the testing set, we don't want it to be too small (as this will miss our goal) and we don't want it to be too large; as the predicted labels were based on what the model has learnt, so training and validating the model on a large set based on these labels, the model will believe more in its predictions and will learn more what it's already learned instead of learning new features which leads to overfitting (a simple trial on a portion of 50% results in 97% accuracy and 0.1 loss on the new testing set, but, 90% accuracy and 0.4 loss on the old testing set). So, we chose a portion of size equals 1.5 $x\ trainig\ set\ size$ (that's 21.4% of the testing set). The new dataset had a size of 3007 samples which was split into 75% training and 25% testing sets. Before retraining, the model was re-compiled with an increased learning rate of e^{-4}.

The model was train on k= 3 folds, 50 epochs each. It achieved better results on the old testing set with accuracy of 92.02% and a loss of 0.33.

IV. Results

Model Evaluation and Validation

The final model chosen is the retrained improved model we discussed earlier. This model was obtained by picking the best performing model on the original data, and then applying pseudo-labeling to increase the training data and make use of the large unlabeled testing data. This model achieved the best results that when we analyze its performance we find the following:-

On the new test set (after pseudo-labeling: 752 samples):-

Accuracy	*Loss*	*F1-score (ships)*	*F1-score (icebergs)*
95.48%	0.1214	0.95	0.96

Table 1: Best model results (on the new test set)

On the old test set (before pseudo-labeling: 401 samples):-

Accuracy	*Loss*	*F1-score (ships)*	*F1-score (icebergs)*
92.02%	0.33	0.92	0.92

Table 2: Best model results (on the old test set)

The model has high f1-score for both classes, and if we looked at the confusion matrix (figure 8) and the classification report in (table 3) to see how the model performed in predicting the two classes, we find that the model isn't biased towards a particular class, which adds to its validity. In addition to the high f1-scores, the model got as high values in precession and recall for both classes, which indicates its robustness.

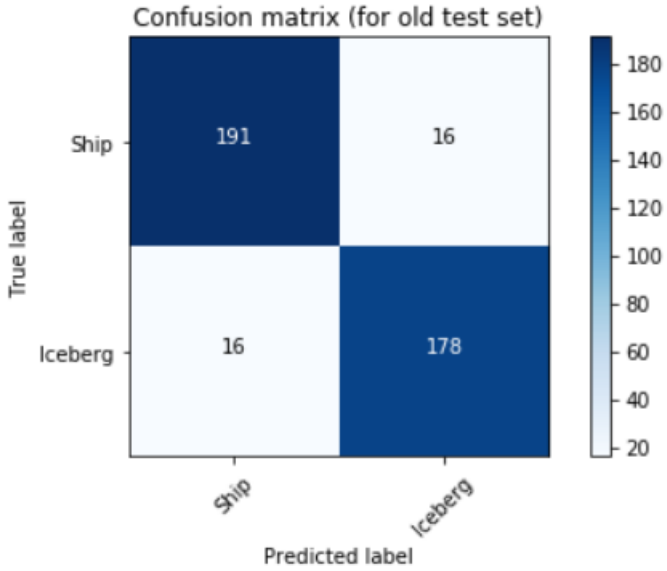


Figure 8: Confusion matrix of final model

Class	precision	recall	*F1-score*	support
Ship	0.92	0.92	0.92	207
Iceberg	0.92	0.92	0.92	194

Table 3: Classification report of final model

To further see the diagnostic ability of our binary classifier, we plot the **Receiver operating characteristic** curve (ROC), and see the value of the **Area under the Curve** (AUC):

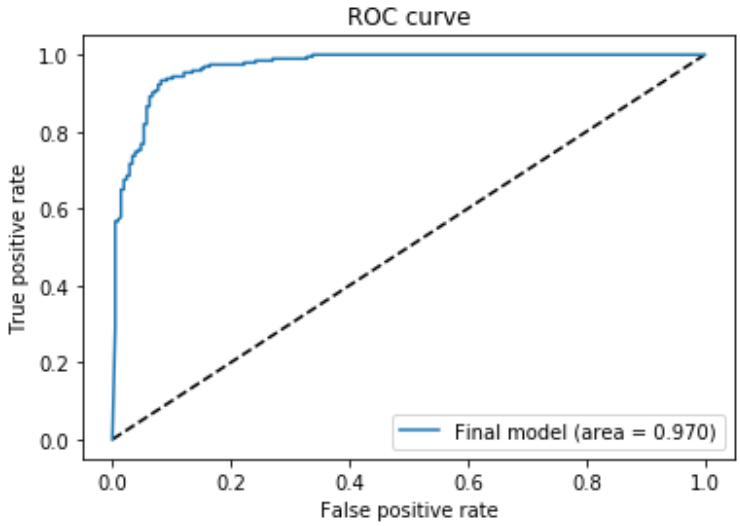


Figure 9: ROC curve of the final model

As we can see, the model shows a very good performance with a high AUC of 0.97.

To further examine the model performance and robustness, we considered submitting our solution to the Kaggle competition to see how it performs against new unseen data. More details in the next section.

Justification

In the following table, we compare between the benchmark model and the final model:

Model	*Accuracy*	*Loss*	*F1-score (ships)*	*F1-score (icebergs)*
Benchmark Model	*83%*	*0.35*	*0.88*	*0.88*
Final Model	***92.02%***	***0.32***	***0.92***	***0.92***

Table 4: Comparison between benchmark model and final model

As we can see, the final results are stronger than the benchmark result as reported.

When we compare the ROC curves of both models, we find the following:

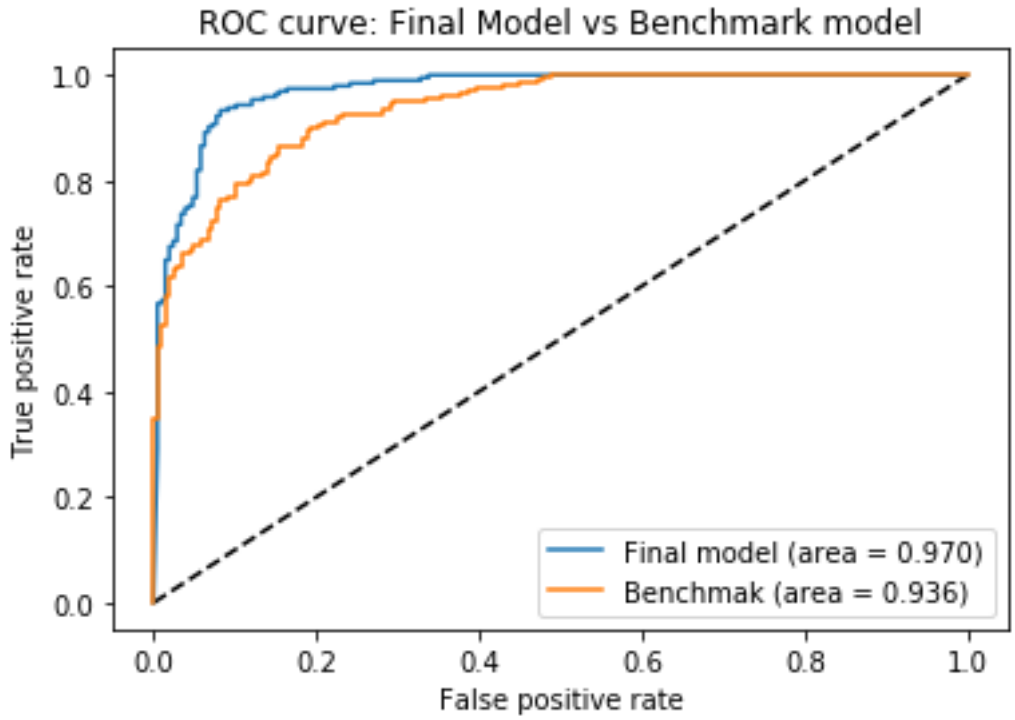


Figure 10.1: ROC curve of the final model vs the Benchmark model

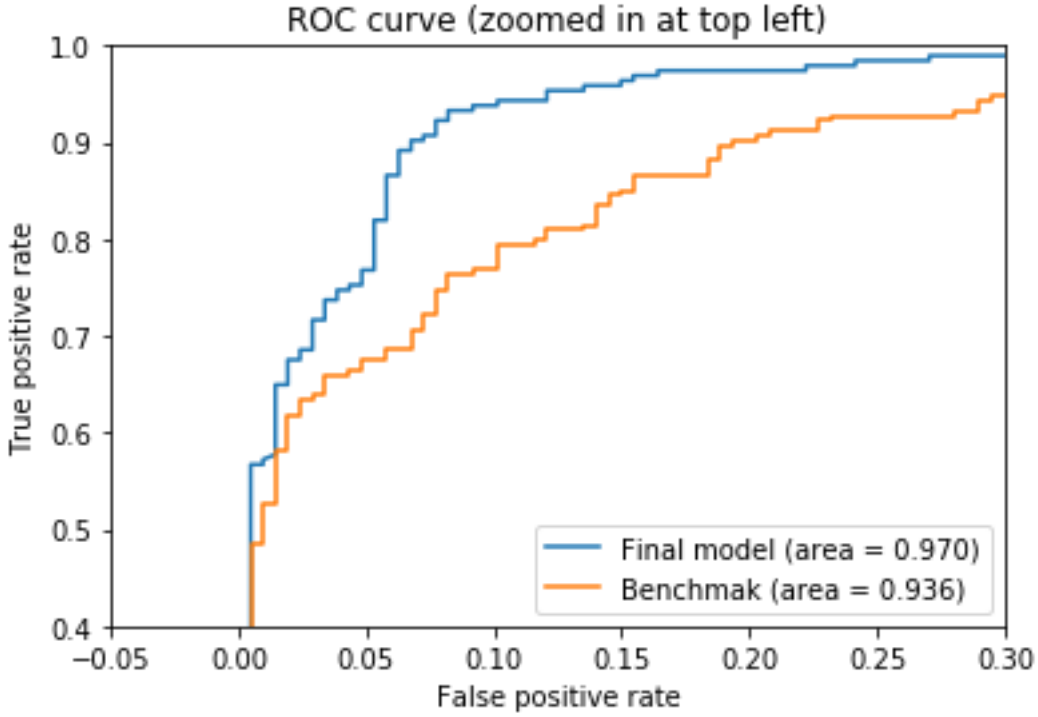


Figure 10.2: ROC curve (zoomed) of the final model vs the Benchmark model

As you can see, given the AUC metric, final model classifier outperforms the benchmark classifier.

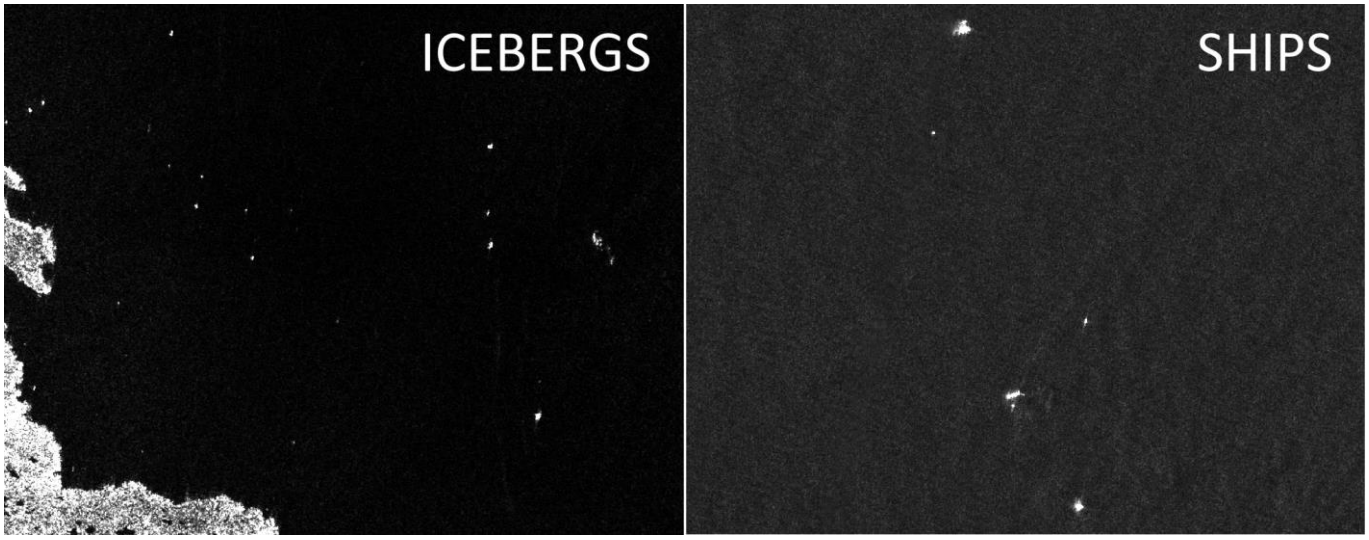
When the final solution is submitted to Kaggle competition, it gained a leaderboard score of 0.2516 on private Leaderboard and 0.2677 on public Leaderboard.

With that considered, I think that the final solution is significant enough to provide a solution to the current the problem.

V. Conclusion

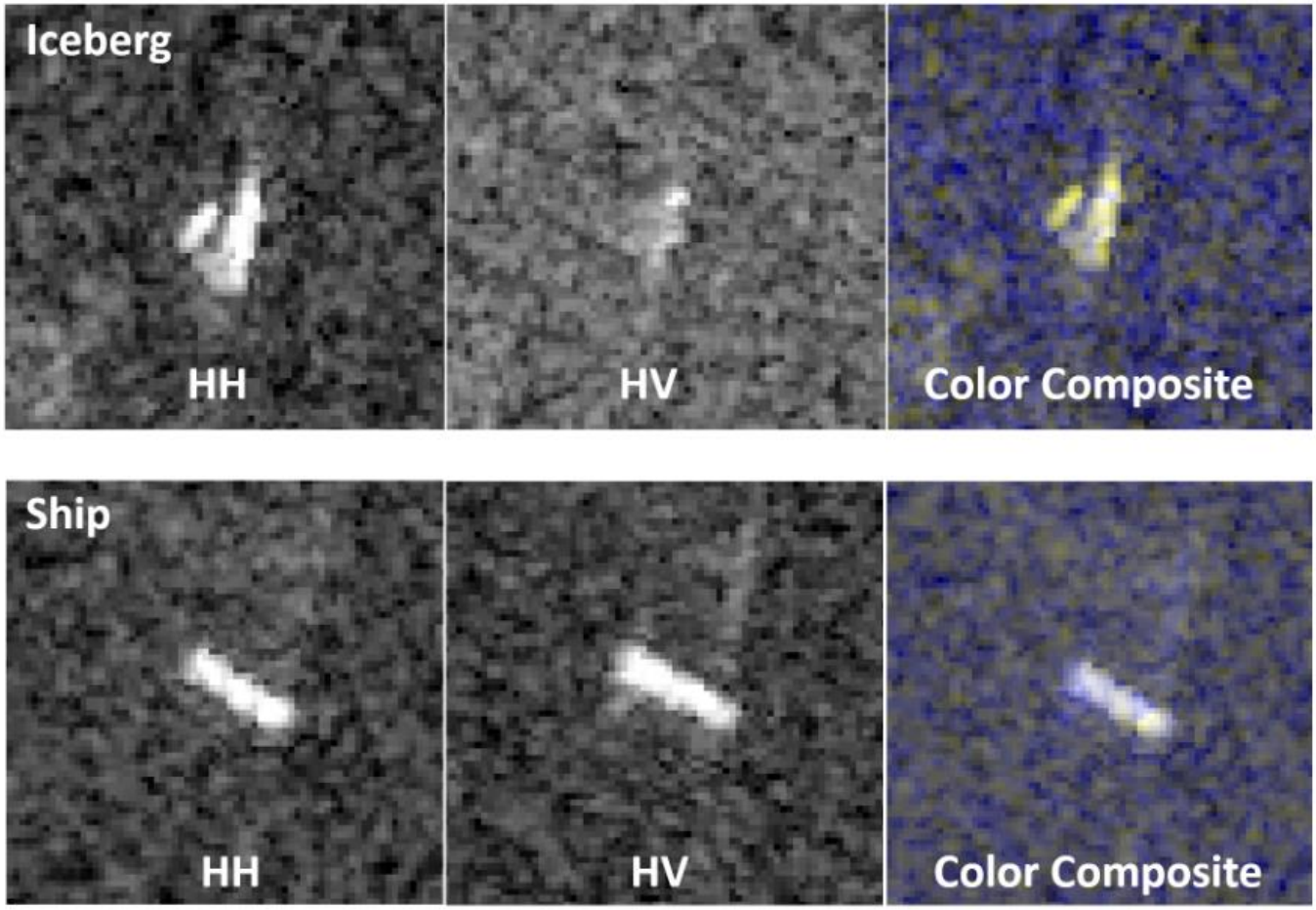
Free-Form Visualization

To get a wider view of the characteristics of this problem, several issues should be made clear; the 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. An object will appear as a bright spot because it reflects more radar energy than its surroundings, but strong echoes can come from anything solid - land, islands, sea ice, as well as icebergs and ships. The energy reflected back to the radar is referred to as backscatter.

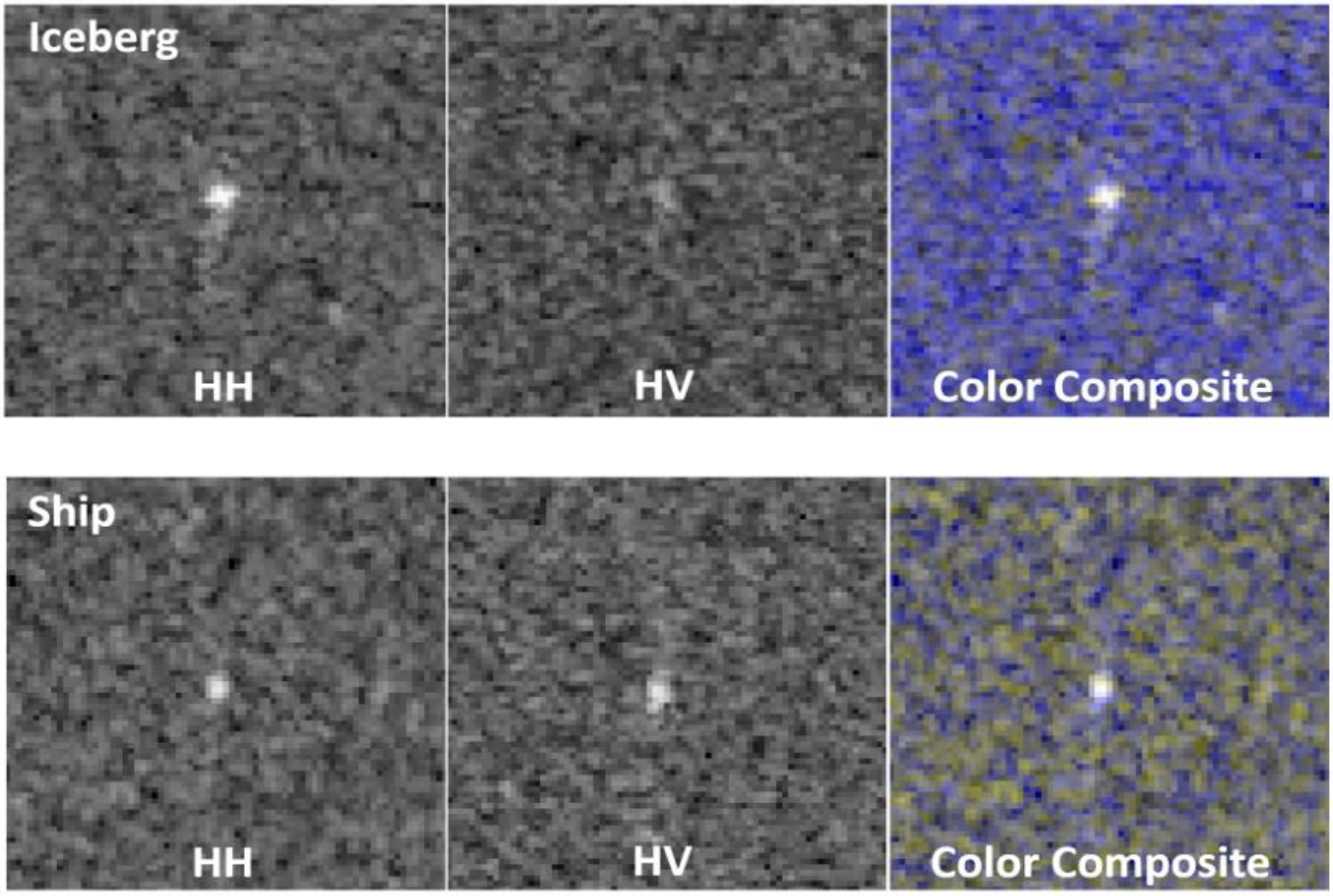


When the radar detects an object, it can't tell an iceberg from a ship or any other solid object. The object needs to be analyzed for certain characteristics - shape, size and brightness - to find that out

We saw data with two channels: HH (transmit/receive horizontally) and HV (transmit horizontally and receive vertically). This can play an important role in the object characteristics, since objects tend to reflect energy differently. Easy classification examples are seen below. These objects can be visually classified. But in an image with hundreds of objects, this is very time consuming.



Here we see challenging objects to classify:



Although the data exhibit hard to detect difference in many cases, the final model did a good job in the classification problem with about 92% accuracy. Here is an exploration of how it performs with different sample images:

```
Predictions on sample images of icebergs
```

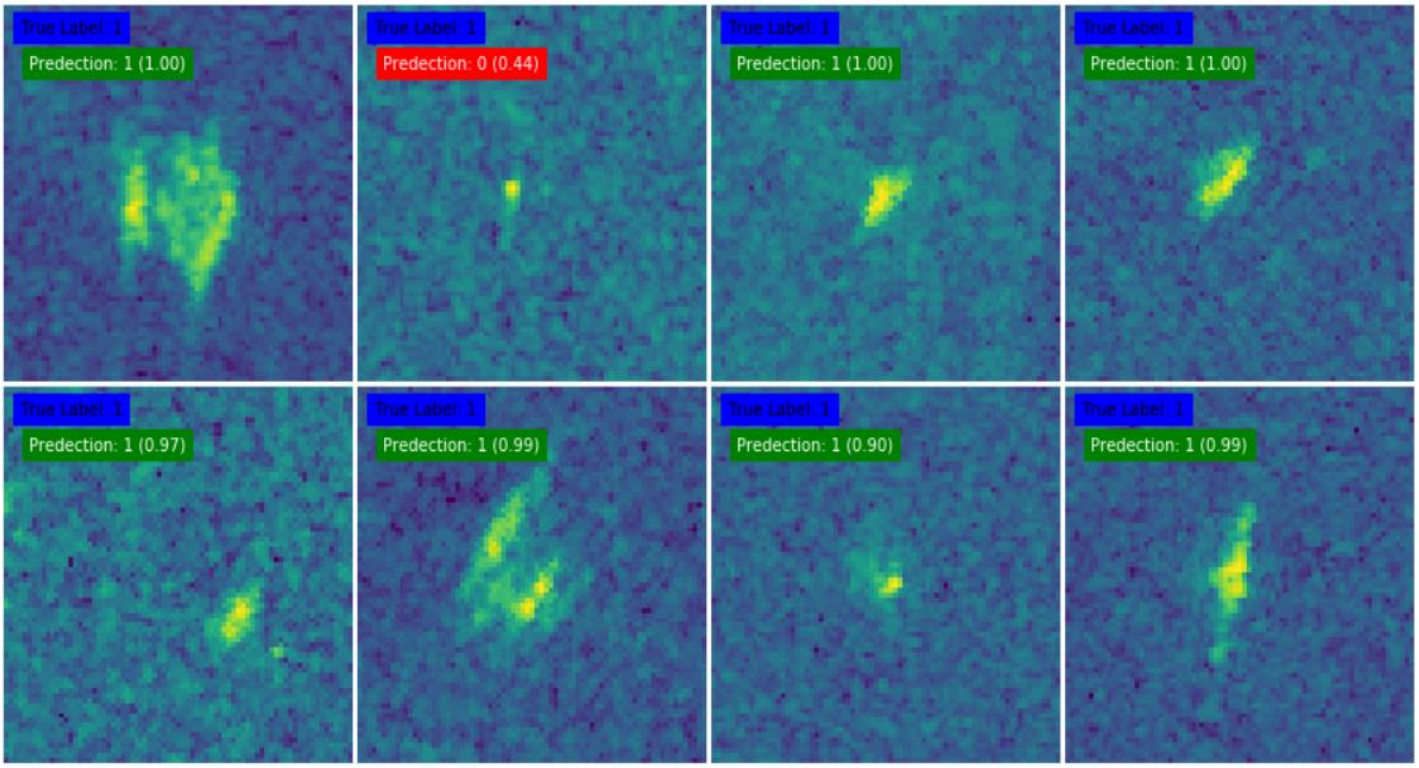


Figure 5: sample icebergs predictions

As we can see the model did a pretty good job; it predicts the correct label and with very high probability. Although the misclassified image is hard to predict it's only 0.06 away of the probability needed to properly classify it as an iceberg.

```
Predictions on sample images of ships
```

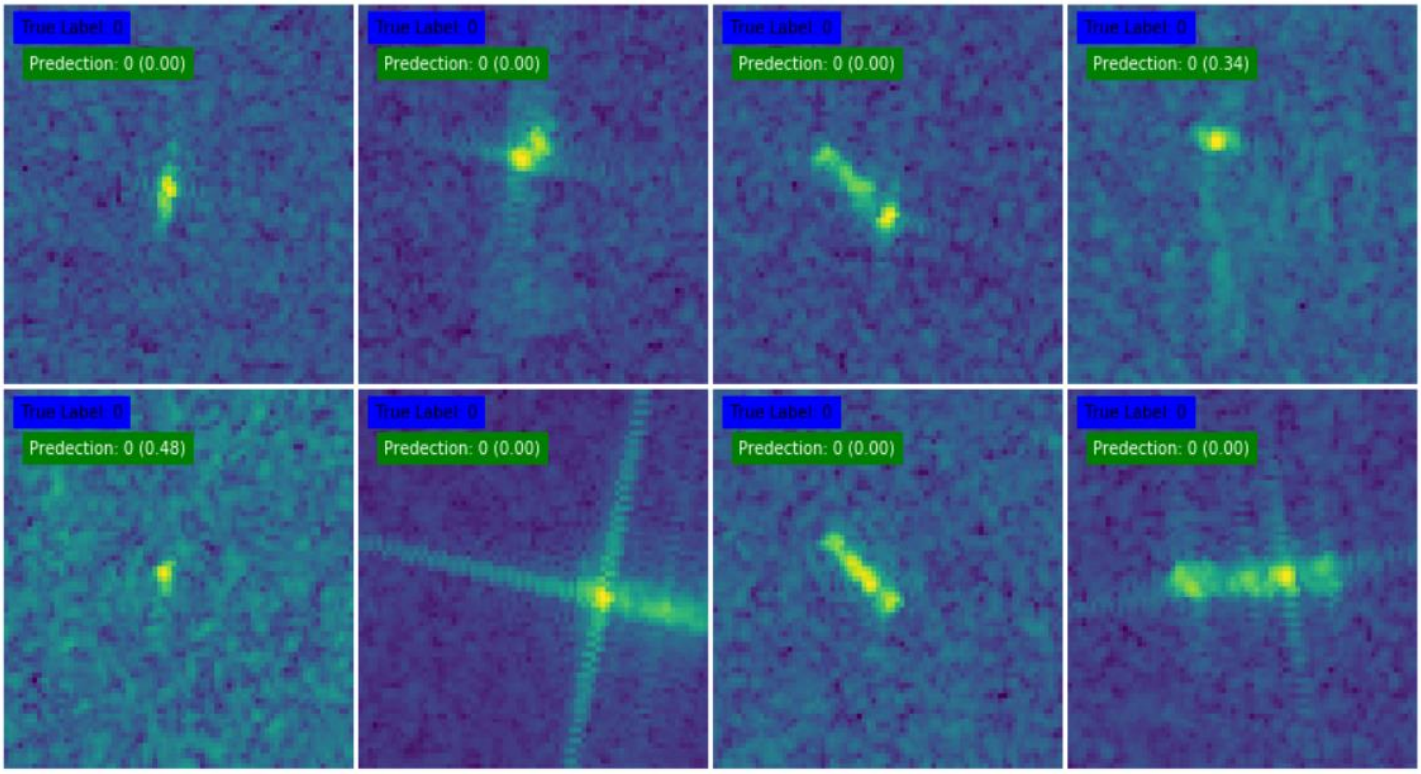


Figure 6: sample ships predictions

Here, the model perfectly classified ship images with full certainty, except for two examples that have subtle differences with the iceberg images.

Reflection

At first, we preprocessed the data; as the 'inc_angle' column has many missing values we decided to and exclude it. We combined the two bands to reconstruct each data point as an image with two channels with shape 75X75X2.

We then implemented different models and applied improvements along the way. To summarize:

- By dimensionality reduction and using classifiers like Random Forest, K-NN and Logistic Regression we didn't get good results and the accuracy didn't exceed 79%.
- Using CNNs has led to better results where a simple CNN benchmark model outperformed the previous methods with ~83% accuracy and up to 90% accuracy after tuning, 0.26 loss and f1-scores of 0.90 and 091 for ship and iceberg classes, respectively.
- By increasing the complexity of the CNN model and introducing more layers we got a better model with ~91.8% accuracy, 0.34 loss, and higher f1-scores of 0.92 and 0.91 for ship and iceberg classes, respectively.

- By making use of transfer learning with VGG16 together with data augmentation, we got 90.5% accuracy, 0.247 loss, and f1-score of 0.91.
- With pseudo-labeling, and by retraining the best obtained model on the increased training data we got a better model with:
    - 95.48% accuracy and 0.1214 loss on the test data (after pseudo-labeling: 752 samples), and f1-scores of 0.95 and 0.96 for ship and iceberg classes, respectively.
    - 92.02% accuracy and 0.32 loss on the test data (before pseudo-labeling: 401 samples), and f1-score of 0.92 for both ship and iceberg classes.

Results (Best in **bold**):

Model	Accuracy	Loss	Precession (ship-iceberg)	Recall (ship-iceberg)	F1-score (ship-iceberg)	LB score
Benchmark Model	83%	0.35	0.88	0.88	0.88	N/A
Refined Benchmark	90.77%	0.264	0.91	0.91-0.90	0.91-0.90	N/A
Improved Model	91.8%	0.34	**0.92**	0.92-0.91	0.92-0.91	0.2719
Transfer Learning + Data augmentation	90.5%	**0.247**	0.91-0.90	0.90-0.91	0.91-0.90	N/A
Pseudo-labeling	**92.02%**	0.32	**0.92**	**0.92**	**0.92**	**0.2677**

Table 5: Comparison between model results

One of the interesting aspects of this project is that the satellite data imagery came from radars not cameras, which makes it a good practice on real-world situations where obtaining regular high resolution images isn't feasible or adequate. The hard to spot differences between the images makes it a challenging yet interesting problem that offered a good experience on how to approach classification problems where objects can't be easily distinguished, a challenge that can be overcome by building complex models that captures subtle differences and underlying patterns in the images. It also shows how powerful deep learning techniques are, and how CNNs can provide significant results when compared to other methods.

Improvement

Even if we set the final model as a benchmark, there will always be a better solution. So, in the future, different steps can be made to enhance the model performance:

* Including `*inc_angle*` into consideration and filling the missing values.

* Getting more training data or trying different data augmentation techniques

* Increasing the complexity of the model together with pre-trained models.

* Applying transfer learning with different pre-trained models other than VGG16.