Machine Learning Engineer Nanodegree

Capstone Project

Satyanarayan Bhanja

November 28, 2017

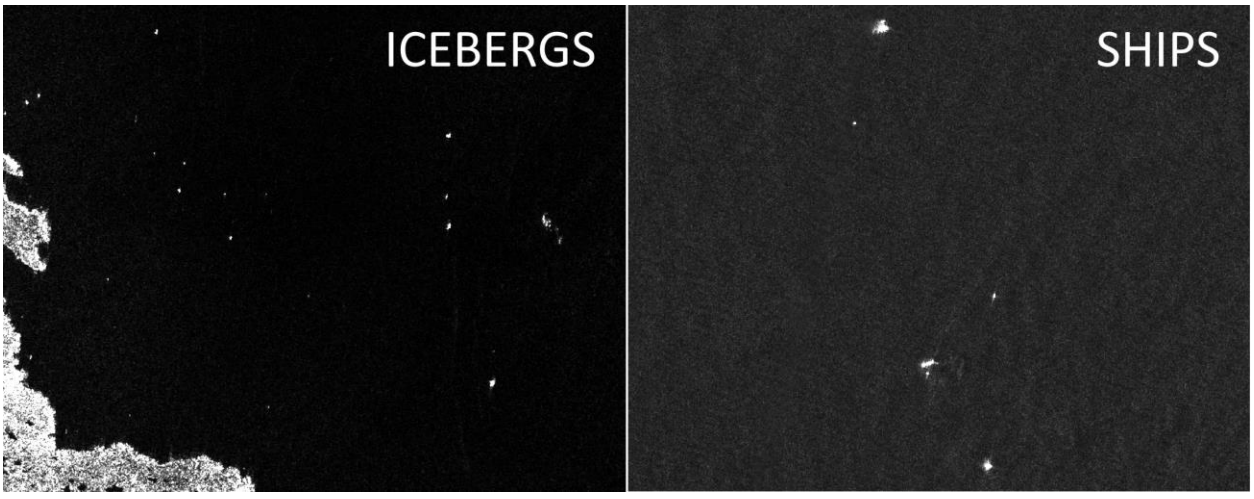
I. Definition

Project Overview

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 it's 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. 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 a 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.



Academic paper where machine learning is applied to this problem:
http://elib.dlr.de/99079/2/2016_BENTES_Frost_Velotto_Tings_EUSAR_FP.pdf
The researchers from German aerospace center(The authors of this paper) got an f1 score of 98% using a CNN model.

The datasets used in this project is from Kaggle competition "**Statoil/C-CORE Iceberg Classifier Challenge**".

Kaggle Datasets link:
https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/download/train.json.7z
https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/download/test.json.7z

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.

The problem is to predict whether an image contains a ship or an iceberg. This is a binary classification challenge.
The output is to predict the probability of an iceberg in the images.

The strategy to solve this problem is as follows.

1. Download the train, test data and preprocess it. The dataset contains image data with two bands for each image.
2. Use image augmentations and create one more band by averaging the values of two bands.
3. Make a CNN classifier and train the model.
4. Predict the probability of an iceberg in the test data images.

Metrics

The evaluation metric used in accuracy.

Accuracy = $\frac{(true\ positive + true\ negative)}{size\ of\ dataset}$

This is a binary classification challenge. So, this metric is used.

II. Analysis

Data Exploration

The dataset obtained from Kaggle competition. The links as below,

Kaggle Datasets link:
https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/download/train.json.7z
https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/download/test.json.7z

The fields of training data as below,

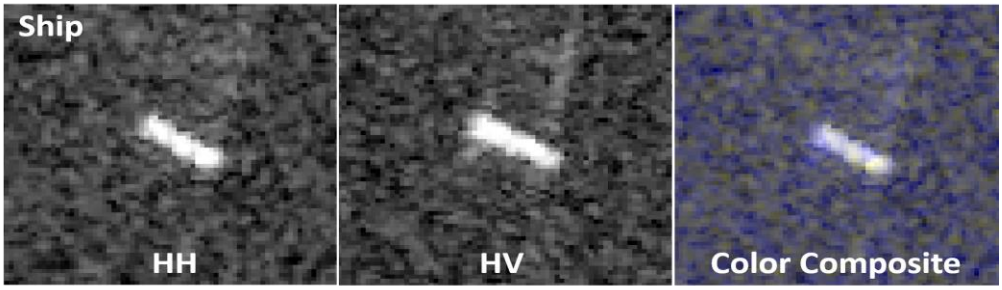
- **id** - the id of the image
- **band_1, band_2** - the flattened image data. Each band has 75x75 pixel values in the list, so the list has 5625 elements.
- **inc_angle** - the incidence angle of which the image was taken.
- **is_iceberg** - the target variable, set to 1 if it is an iceberg, and 0 if it is a ship.
- The testdata has all the fields except is_iceberg column.
- There are total **1604 training data** and **8424 test data.**
- The training dataset has **53.05% "ships" and 46.94% "ice-bergs".** The classes are close to balanced.
- For initial models, training dataset will be devided into 75% training and 25% testing data.
- For final model k-fold cross validation used with 3 folds.

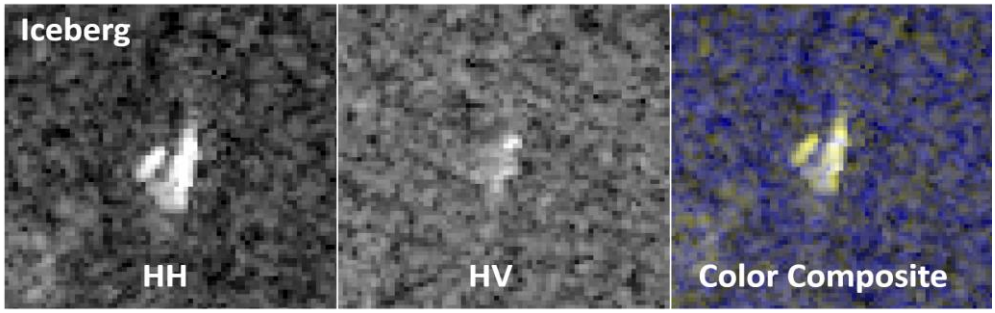
The data looks like as below,

	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

- The band_1 and band_2 are the HH and HV bands.
- HH (transmit/receive horizontally) and HV (transmit horizontally and receive vertically).

Sample iceberg and ship images are as in kaggle below,

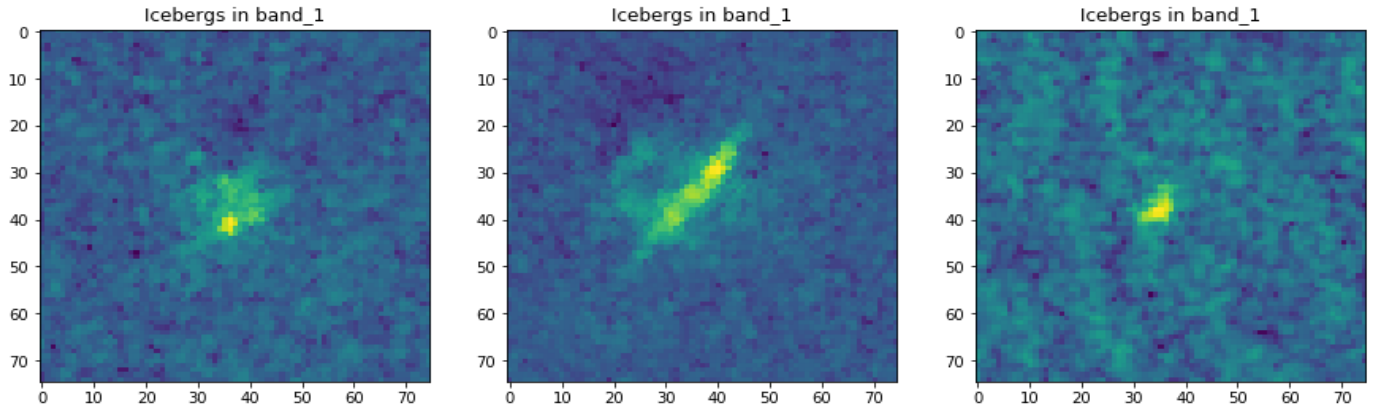




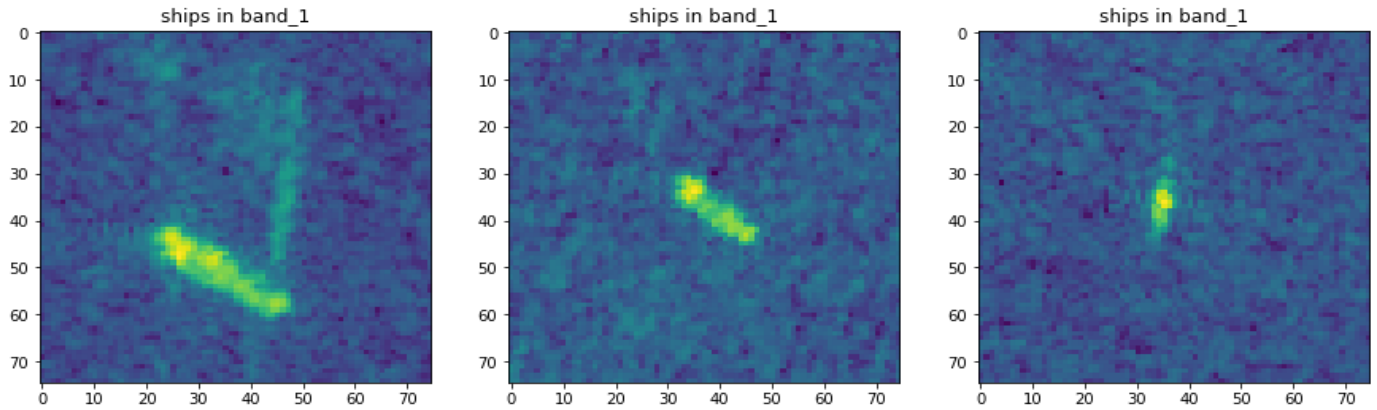
Exploratory Visualization

Lets see the images of icebergs and ships in band-1 and band-2,

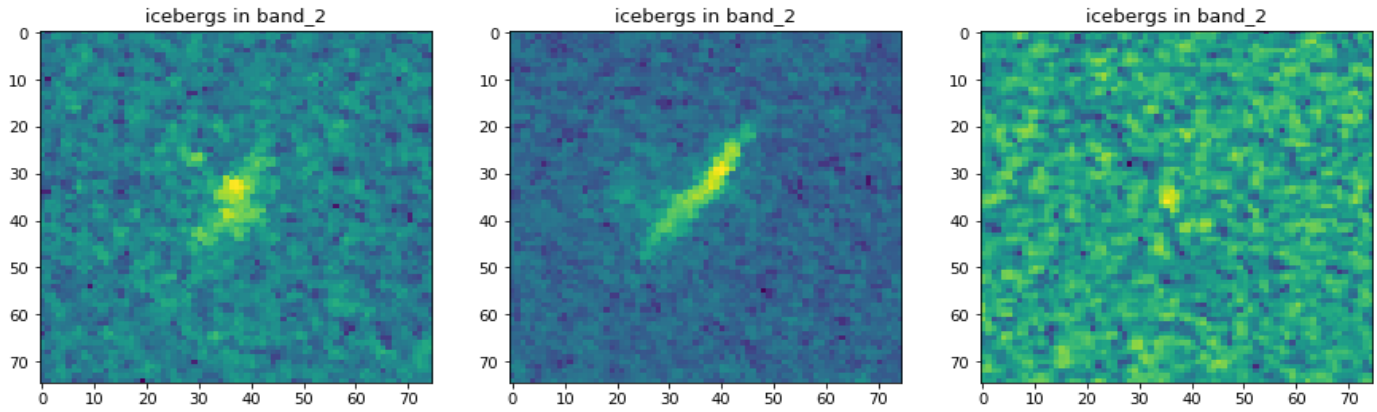
images of icebergs in band_1 :-



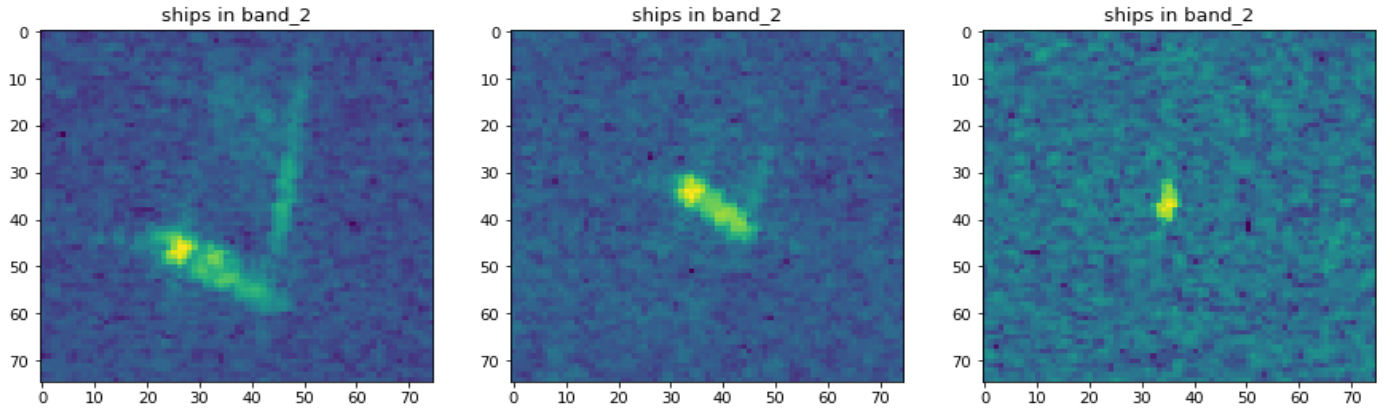
images of ships in band_1 :-



Same set of icebergs in band_2 :-

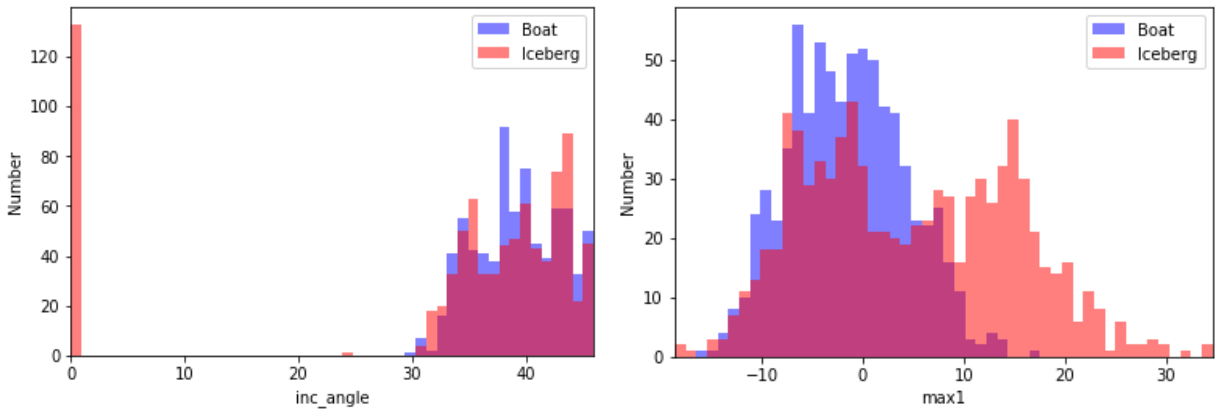


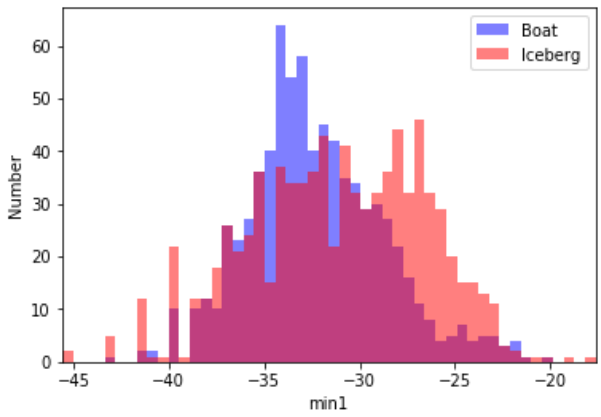
Same set of ships in band_2:-

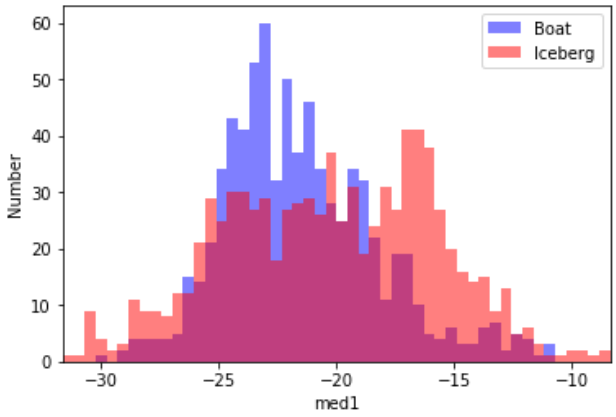


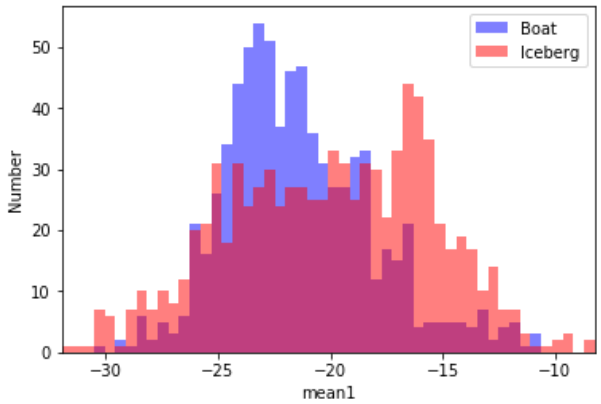
Histograms for band_1:-

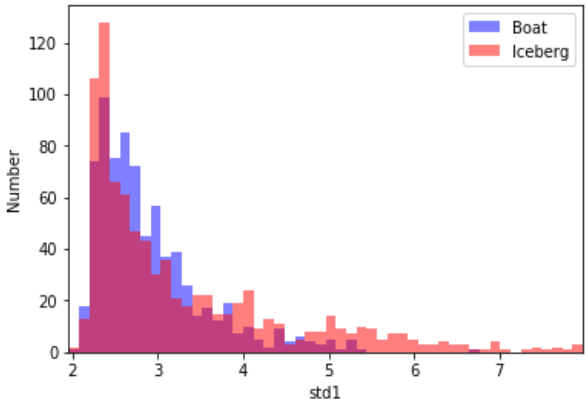
The histograms of inc_angle, max values, min values, median values, mean and standard deviation as below for band_1 boat and ship images.





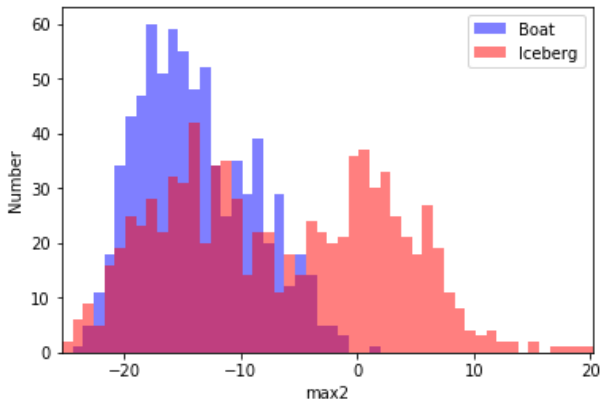


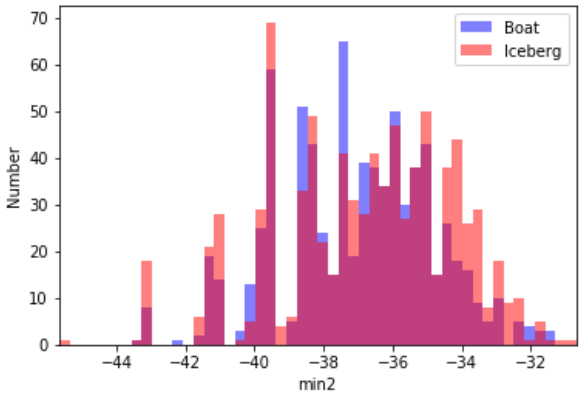


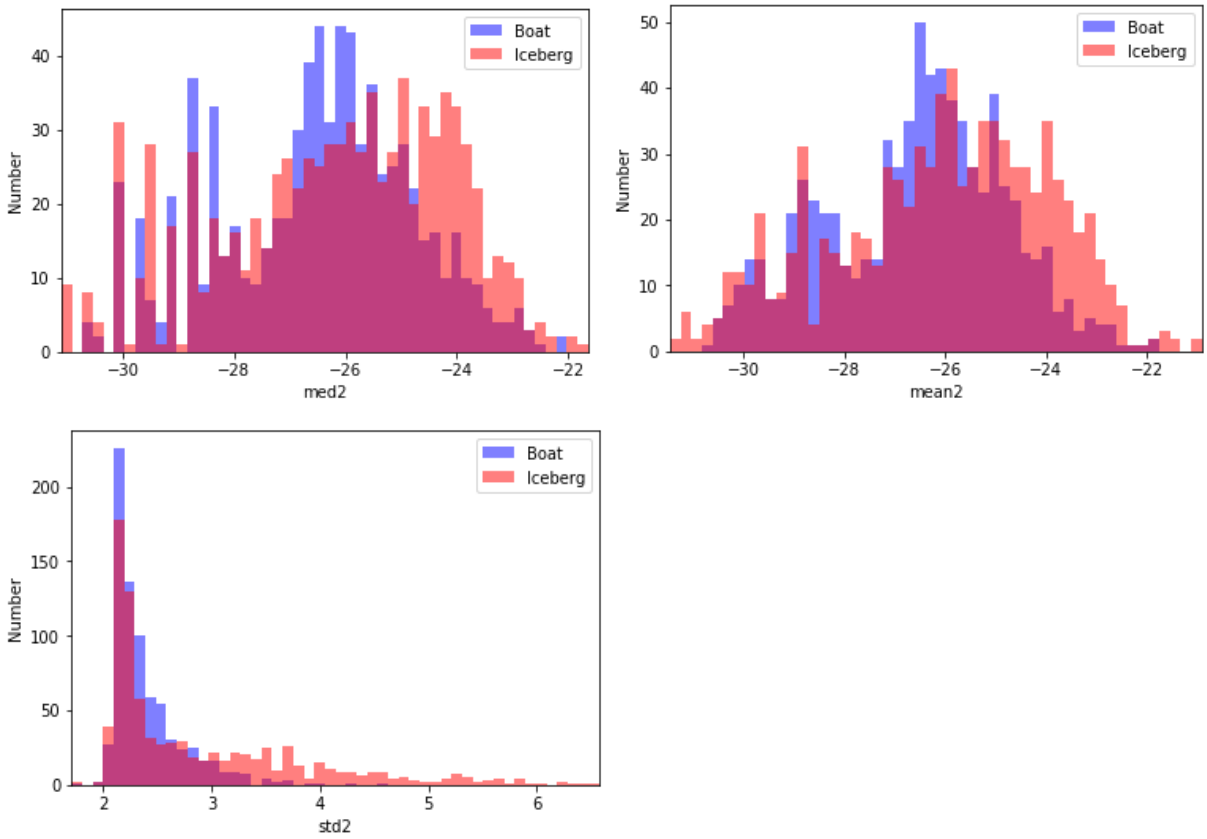


Histograms for band_2:-

The histograms of max values, min values, median values, mean and standard deviation as below for band_2 boat and ship images.

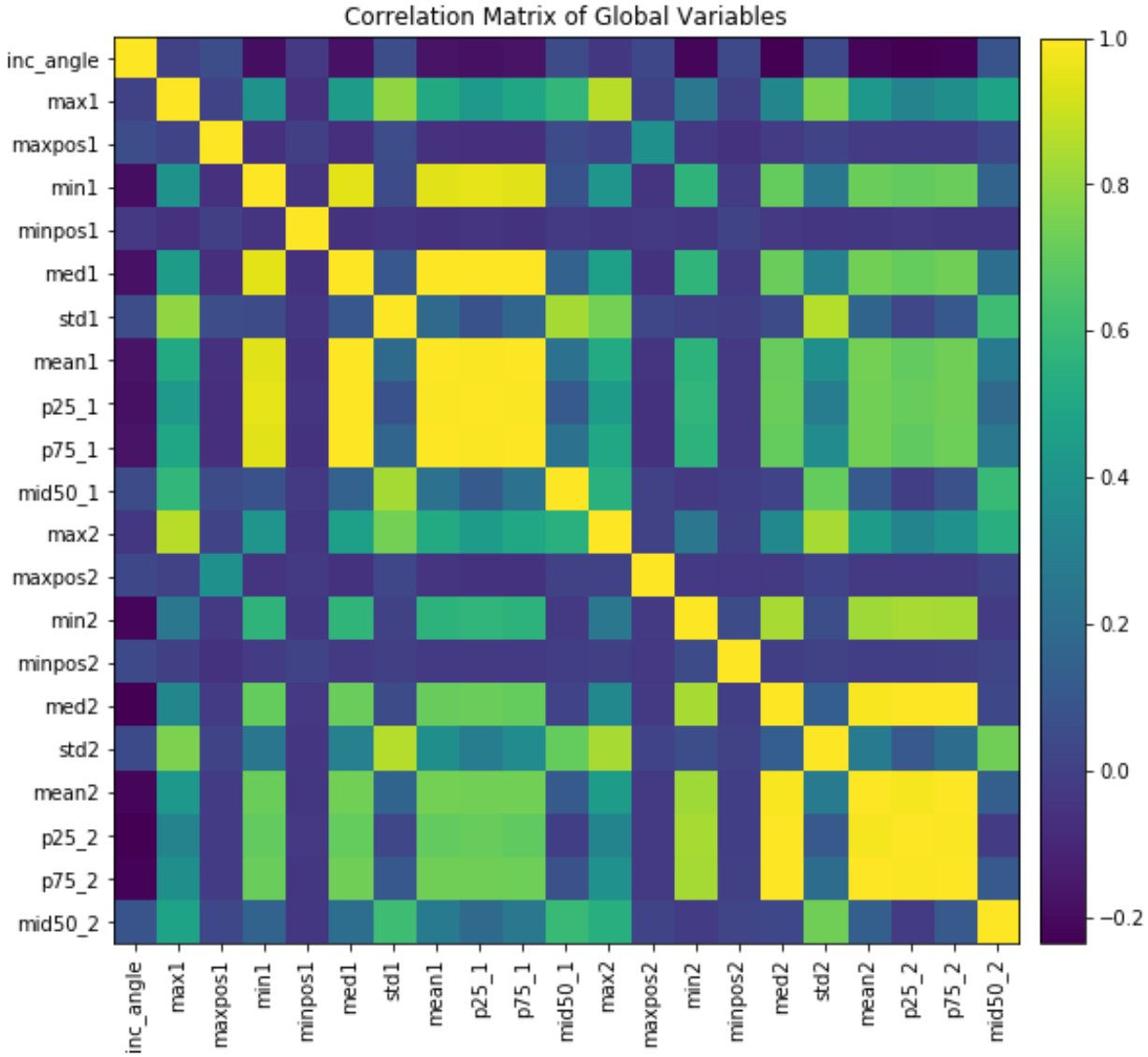






From, the histogram plots it is clear that, both band have similar results.

Correlations Between Features:-



We see that there are large correlations between some of the variables. In particular, the mean, median, 25% signal, and 75% signal are all closely related, with nearly 75% correlation. The min and max are also pretty highly correlated for band 1, as are the min and median for both bands, suggesting that the signals have maybe been scaled in some way to force this correlation. There are also some correlations between the two bands. Finally, we see an anticorrelation of around -0.5 between the mean of band 2 and the angle, with a weaker correlation for band 1.

Algorithms and Techniques

The classifier is a **Convolutional Neural Network**, which is the state-of-the-art algorithm for most image processing tasks, including classification. It needs a large amount of training data compared to other approaches. The algorithm outputs an assigned probability for each class.

The model will be trained using transfer learning, which uses the popular pretrained model vgg-16 architecture. The model validation done using k-fold cross validation.

The following parameters can be tuned to optimize the classifier:

1. **Training parameters:**
    - **Stochastic gradient descent parameters:**
        - lr: float >= 0. Learning rate.
        - momentum: float >= 0. Parameter updates momentum.
        - decay: float >= 0. Learning rate decay over each update.
        - nesterov: boolean. Whether to apply Nesterov momentum.
    - **Keras model compile parameters**:
        - optimizer: String (name of optimizer) or optimizer object. See optimizers.
        - loss: String (name of objective function) or objective function. See losses.
        - metrics: List of metrics to be evaluated by the model during training and testing.
    - **Keras fit_generator parameters:**
        - steps_per_epoch: Total number of steps (batches of samples) to yield from generator before declaring one epoch finished and starting the next epoch.
        - epochs: Integer, total number of iterations on the data.
        - shuffle: Whether to shuffle the order of the batches at the beginning of each epoch.
        - verbose: Verbosity mode, 0, 1, or 2.
        - validation_data: A generator for the validation data -> A tuple (inputs, targets)
        - callbacks: List of callbacks to be called during training.

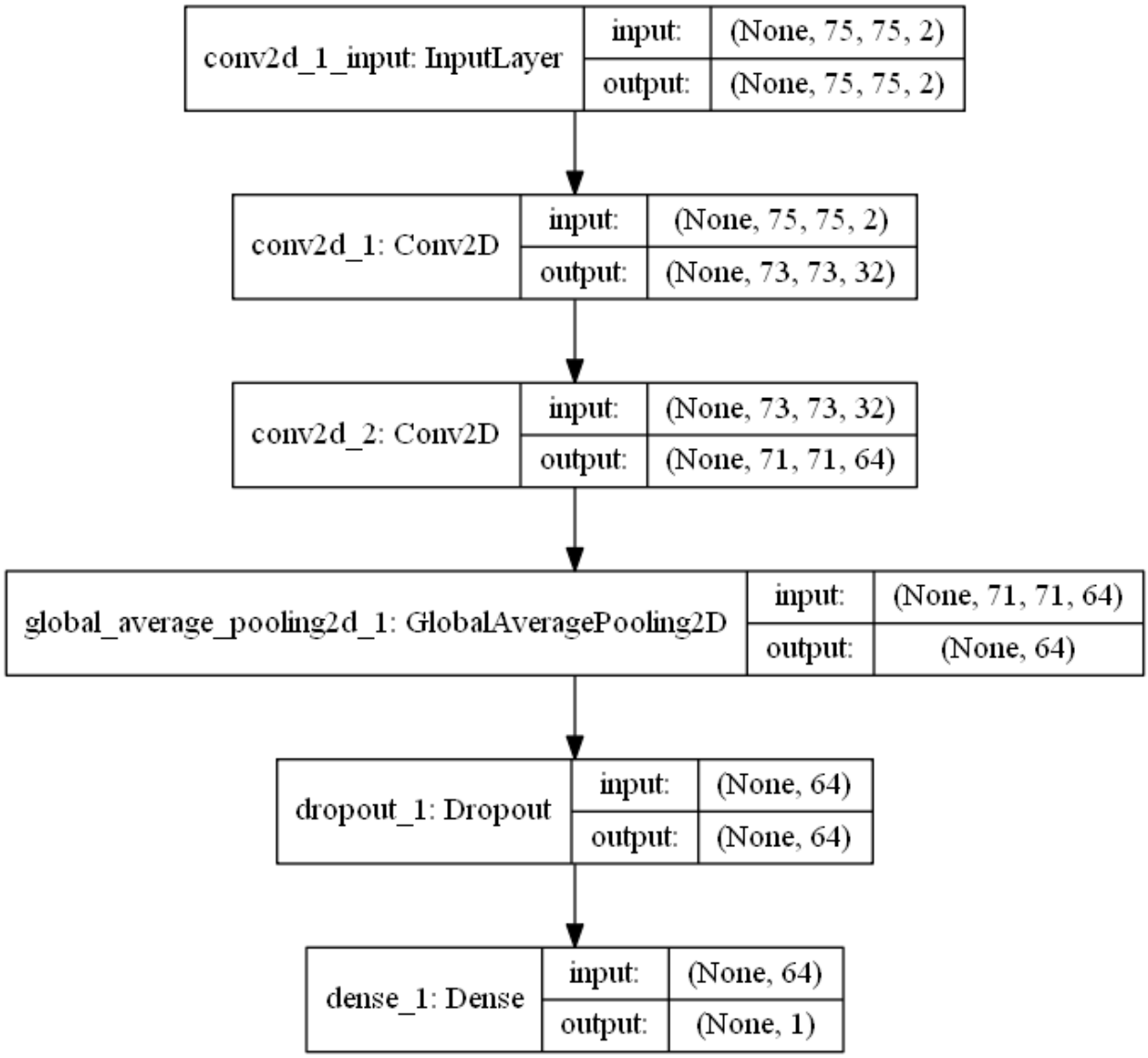
2. **Neural network architecture**
    - Number of layers
    - Layer types (convolutional, fully-connected or pooling, dense, dropout)
    - Layer parameters (see links above)

During training, both the training and the validation sets are loaded into the RAM. After that random batches are selected to be loaded into the GPU memory for processing. The training is done using the Stochastic gradient descent algorithm (with momentum).

Benchmark

The benchmark model is a simple CNN classifier trained on the train data.

The CNN architecture as below,



The first three layers are convolution layers, next is a global average pooling layer. Next is a dropout layer to prevent overfitting. Final layer is a densely connected NN layer.

This model got an validation accuracy of 63.42%.

III. Methodology

Data Preprocessing

The data pre-processing steps done in final model as below,

1. “Inc_angle” column data converted to numeric type in both train and test data.

2. In train data, “Inc_angle” column has 133 NAs. These NAs filled by pad method, which fill values forward.
3. Both band-1 and band_2 converted to numpy arrays and then 32-bit floats. This process done for both train and test data sets.
4. A third band created by averaging the two bands. This process also done for both train and test sets.
5. **Image augmentation**: The Keras ImageDataGenerator used to transform the images. The parameters are as below,
    I. **horizontal_flip**: Randomly flips images horizontally.
    II. **vertical_flip:** Randomly flip images vertically.
    III. **width_shift_range:** Range for random horizontal shifts.
    IV. **height_shift_range:** Range for random vertical shifts.
    V. **channel_shift_range:** Range for random channel shifts.
    VI. **zoom_range:** Range for random zoom.
    VII. **rotation_range:** Degree range for random rotations.

Implementation

Transfer learning method used for the final model.

1. The VGG16 model, with weights pre-trained on ImageNet used as the base model.
2. The parameters of VGG16 as below,
    - include_top: false i.e. do not include the 3 fully-connected layers at the top of the network.
    - input_shape: the input shape.
3. Pass a GlobalMaxPooling2D laye and merge with inc_angle layer.
4. Then add a Dense layer with RELU activation to the merged layers.
5. Then add Dropout, RELU DENSE and dropout layers.
6. Final layer is a Dense layer with sigmoid activation.
7. The Keras compile step:
    - Loss: binary_crossentropy, the loss function.
    - Optimizer: SGD. The Stochastic gradient descent optimizer.
        - **lr:** Learning rate.
        - **Momentum:** Parameter updates momentum.
        - **Decay:** Learning rate decay over each update.
        - **Nesterov:** applied Nesterov momentum.
8. StratifiedKFold used for K-fold cross validation with 3 folds.
9. In each fold,
    - Get the index of cv set and holdout set.
    - Similarly, get the index of inc_angle for cv and holdout.
    - Use callbacks for early stopping of model when a ‘val_loss’ has stopped improving.
    - Use the image data generator and load the weights from VGG16 model and train the model.
    - Get the validation loss and validation accuracy for each fold.
    - The model ran for 100 epochs with batch size 64.
    - Get the best model for prediction.

Refinement

I train three CNN models.

1st: A basic CNN model

2nd: A CNN with moe layers.

3rd: Transfer learning with VGG16 model and image augmentation.

Each of the CNN architecture and their accuracies as below,

Basic CNN model:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 73, 73, 32)        608
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 71, 71, 64)        18496
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 19,169
Trainable params: 19,169
Non-trainable params: 0
_________________________________________________________________
```

Validation accuracy: 60.60%.

2nd CNN Model:

The model has more convolution and max_pooling layers.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_1 (Batch (None, 75, 75, 2)         8
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 73, 73, 8)         152
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 8)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 34, 34, 16)        1168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 15, 15, 32)        4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 5, 5, 64)          18496
```

```
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
global_max_pooling2d_1 (Glob (None, 64)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 30,737
Trainable params: 30,733
Non-trainable params: 4
_________________________________________________________________
```

The validation accuracy for this model is about 83.5%.

This is a significant improvement from the basic CNN model.

3rd Model: VGG16 and image augmentation:

The third model uses image augmentation which is then used along with VGG16 imagenet weights to train the model.

K-fold cross validation used with 3 folds.

The results for each fold as below,

Fold 0:

```
Train loss: 0.15563088912
Train accuracy: 0.933582787708
Test loss: 0.200516237026
Test accuracy: 0.923364486873
```

Fold 1:

```
Train loss: 0.117495991987
Train accuracy: 0.953227315248
Test loss: 0.171188797795
Test accuracy: 0.938317755784
```

Fold 2:

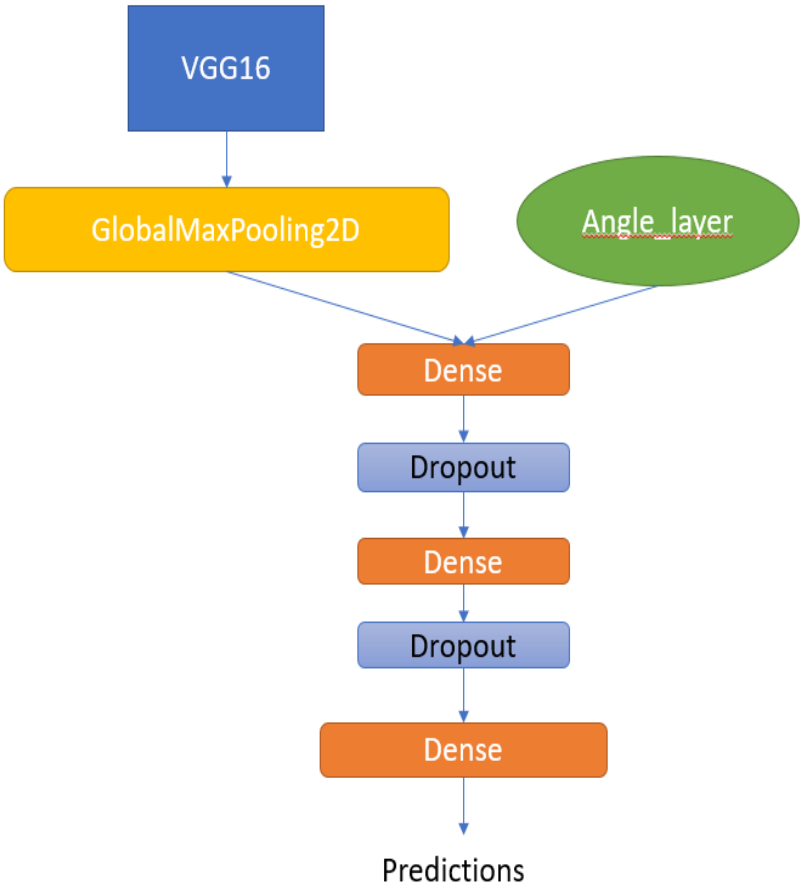
```
Train loss: 0.165511704188
Train accuracy: 0.930841121161
Test loss: 0.26552314928
Test accuracy: 0.887640449215
```

We got test accuracy of 0.93 in Fold1, which is an improvement from the 2nd model.

IV. Results

Model Evaluation and Validation

The final model as below,



[The final model]

The final model built upon VGG16 along with incidence angle.

From VGG16 model I used the pretrained Imagenet weights for training on the data.

After that dropout layers used to prevent overfitting.

This model performed well with test set accuracy of 93%.

Also, I think image augmentation played a good role for better accuracy. The transformed images used for training, so that overfitting can be eliminated.

Justification

The Benchmark model(1st model) has a test accuracy of 16.60% , while the final model has test accuracy of 93%.

So, clearly final model is stronger model.

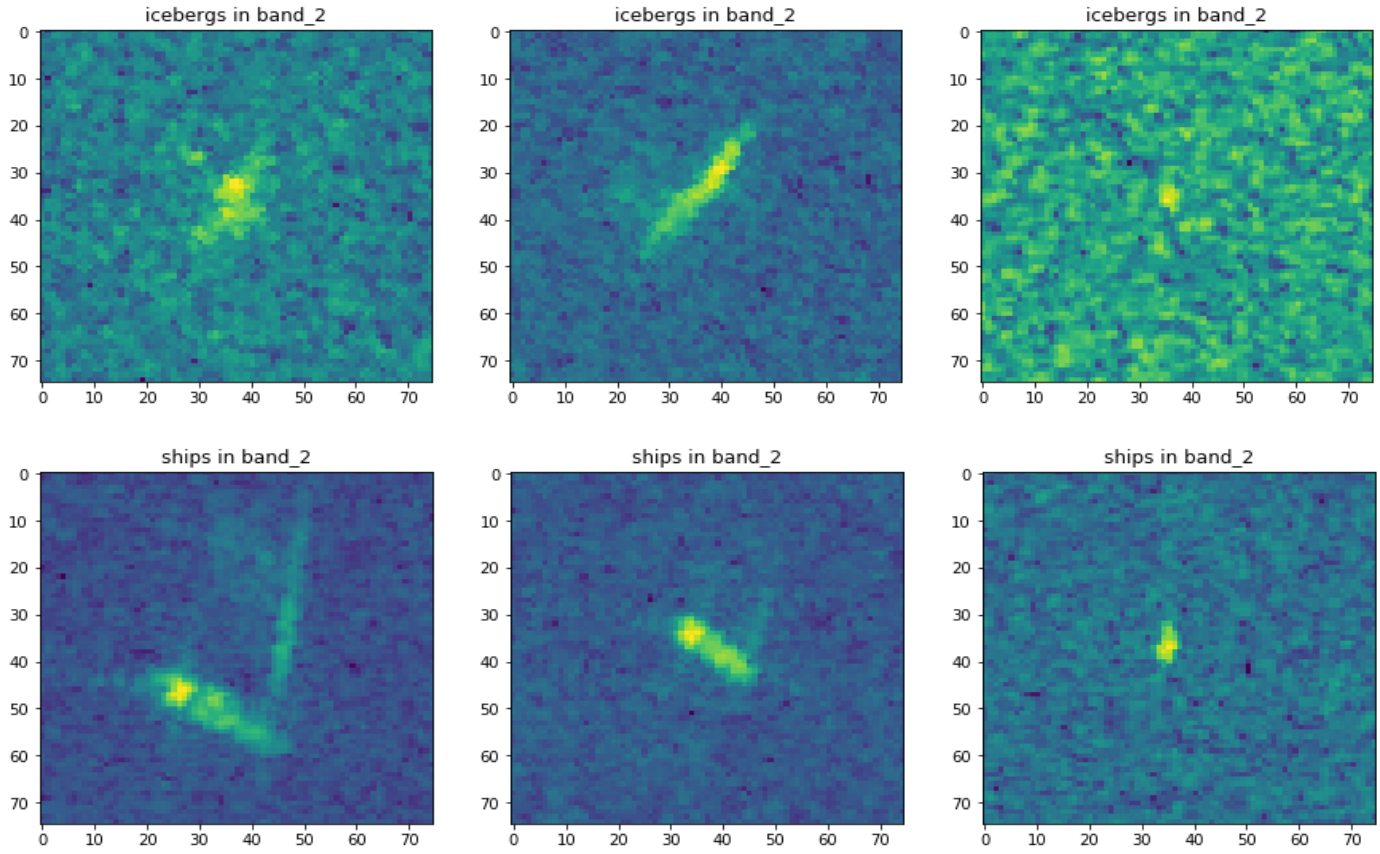
I think the model is significant enough for this problem's solution.

The image augmentation step is the feature generation step and VGG16 is a robust model for image classification. Together they make the best model for this problem solution.

V. Conclusion

Free-Form Visualization

If you see the below images, it is difficult to classify from naked eye, whether it is ship or iceberg.



So, for this type of problem deep learning models will help in classification.

Reflection

The process used for this project can be summarized using the following steps:

1. Get data from Kaggle. The dataset is in json format so, convert it to a dataframe.
2. Fill NAs in Inc_angle column.
3. Convert band_1 and band_2 to numpy arrays with 32bit float data types.
4. Create a third band by averaging the two bands.
5. Use image augmentation to transform the image and prepare for training.
6. Use VGG16 architecture with incidence angle for training.
7. Use cross validation with 3 folds.
8. Finally predict for unseen test data.

I found the image augmentation step challenging. Because, the image needs to be generated in batches and passed to the model.

The interesting aspect of this project is that, this is a very good dataset for image classification. These images are not from camera but from radar reflections. That's why it is very interesting to work on.

I think my solution fit the expectation of the problem to classify whether it is ship or iceberg and can be used in general setting to solve these type of problem, though some improvement might be required.

Improvement

I might consider following improvements in the future,

1. The 3rd band created from averaging the two bands. May be some different method to generate the 3rd method.
2. Applying different image augmentation techniques.
3. Using a different pretrained model apart from VGG16.
4. Also, more CNN layers on top of pre-trained model.

I want to try with caffe2, pytorch and CNTK libraries and see how they perform.

These are the libraries used by computer vision researchers extensively.