Machine Learning Engineer Nanodegree

Project Report

Jean-Michel Taverne

August 03nd, 2019

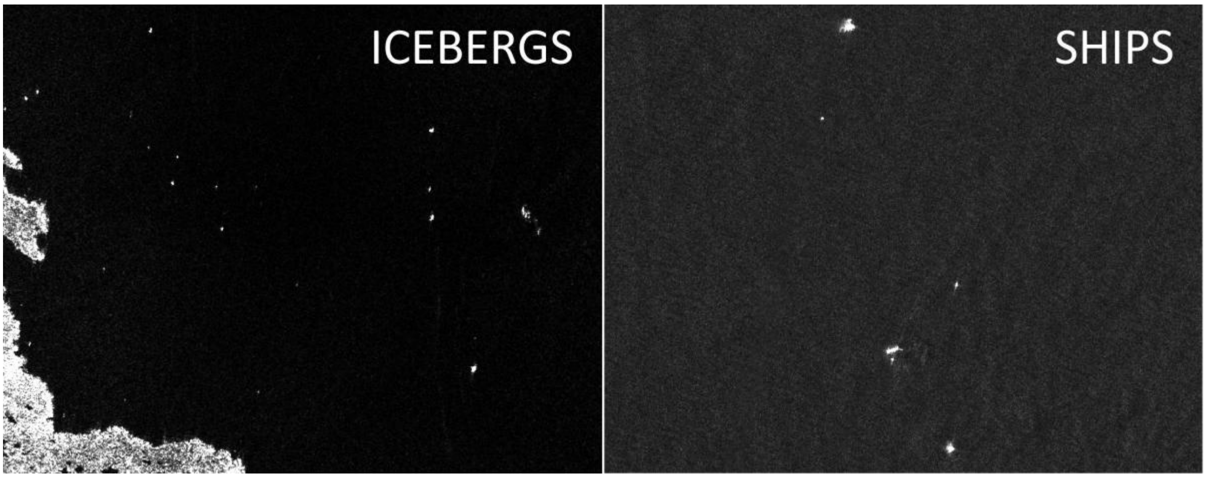
Content

I. Definition

Domain Background

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.



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 goal is to predict is_iceberg is '0' or '1'. If '1' it is an iceberg else '0', which is a ship.

There is already an academic paper where machine learning is applied to this problem. This paper can be found under following link:
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 strategy to solve this problem is as follows.

1 Data Preparation

- 1.1 Load the dataset and remove NAs
- 1.2 Create a third band by averaging the first and the second band. Now the images are of size (75x75x3).
- 1.3 Preprocess images using image augmentation
- 1.4 Split the data into train and test data,

2 Modelling and Validation

- 2.1 First model
  - 2.1.1 Create a CNN with 2 convolutional and one pooling layer with Keras to classify the images
  - 2.1.2 Validate the CNN with the validation set
- 2.2 Second model (improved first model)
  - 2.2.1 Improving the CNN by adding:
    - more convolutional layers and pooling layers,
    - dropout layers
    - using Batch Normalization
  - 2.2.2 Validate the CNN with the validation set
- 2.3 Third model using transfer learning
  - 2.3.1 Create a CNN with transfer learning by using the vgg16
  - 2.3.1 Validate the CNN with the validation set

3 Final prediction

- 3.1 Choose best model and predict on test data
- 3.2 Upload predictions to Kaggle to check public score.

Metrics

The evaluation metric which was used is the accuracy.

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

The simple accuracy metrics used on the training dataset has 53.05% "ships" and 46.94% "ice-bergs". The classes are close to balanced. Based on this I decided to use the accuracy instead of the f1-score. If the training data classes would be imbalanced, then f1-score metric would be better.

II. Analysis

The dataset is available over the corresponding Kaggle 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

The training data is containing following fields:

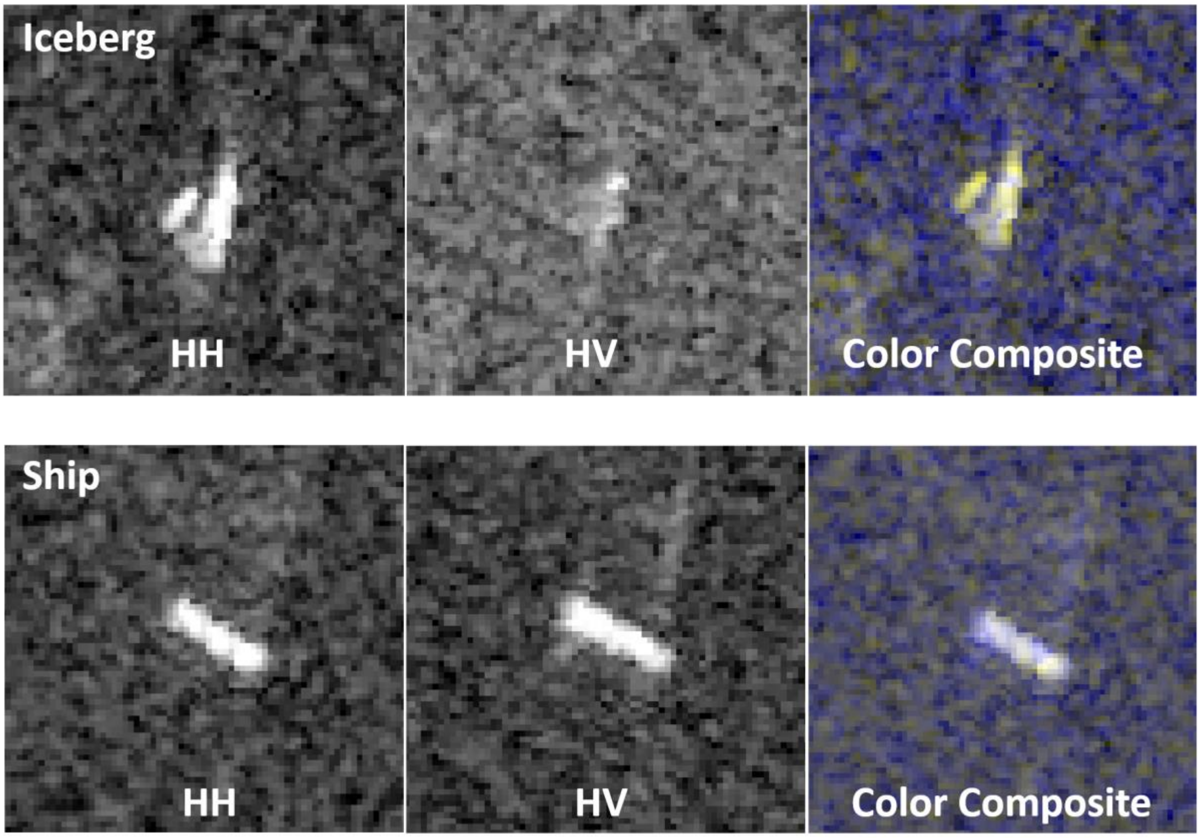
- Id: The ID for the image
- band_1, band_2: The flattened image data. Each band has 75x75 pixel values in the list, so the lost has 5625 elements
- inc_angle: the incidence angle of which the image was taken
- is_iceberg: the target variable, set to 1 if it is an iceberg and otherwise 0

The test data has all the fields except is_iceberg column. There are total 1604 training data and 8424 test data samples. Within the training data there are around 53% of the samples marked as “ships” and 47% marked as “ice-bergs”. The training dataset will be divided into 75% training and 25% testing data.

An example of the data is provided in the following image.

	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

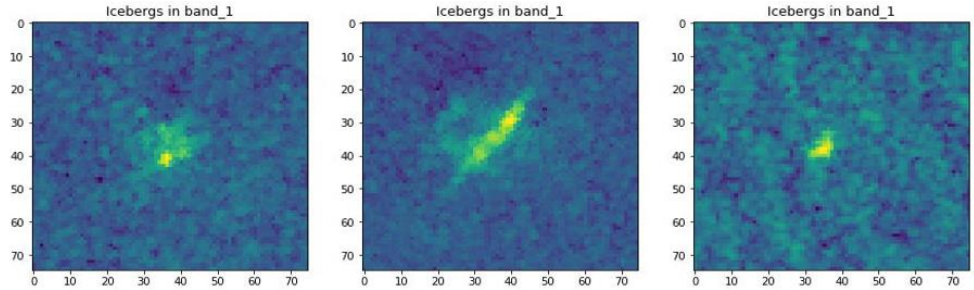
Sample iceberg and ship images are as in Kaggle below.



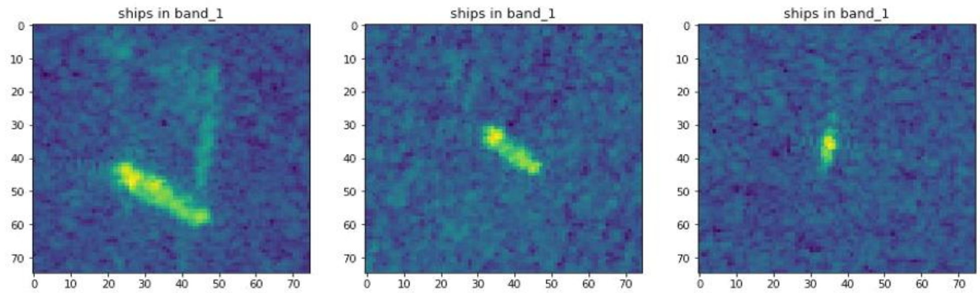
Exploratory Visualization

Let's take a look at the images of icebergs and ships in band-1 and band-2.

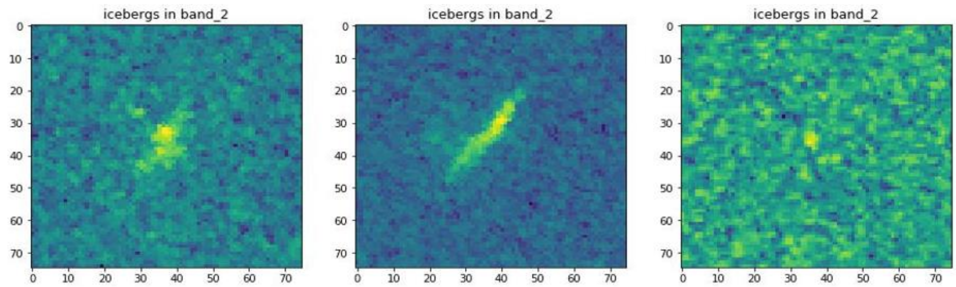
Images of icebergs in band_1:



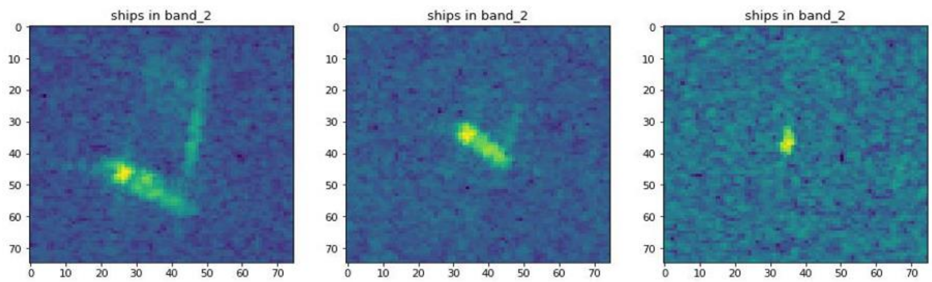
Images of ships in band_1:



Images of icebergs in band_2:

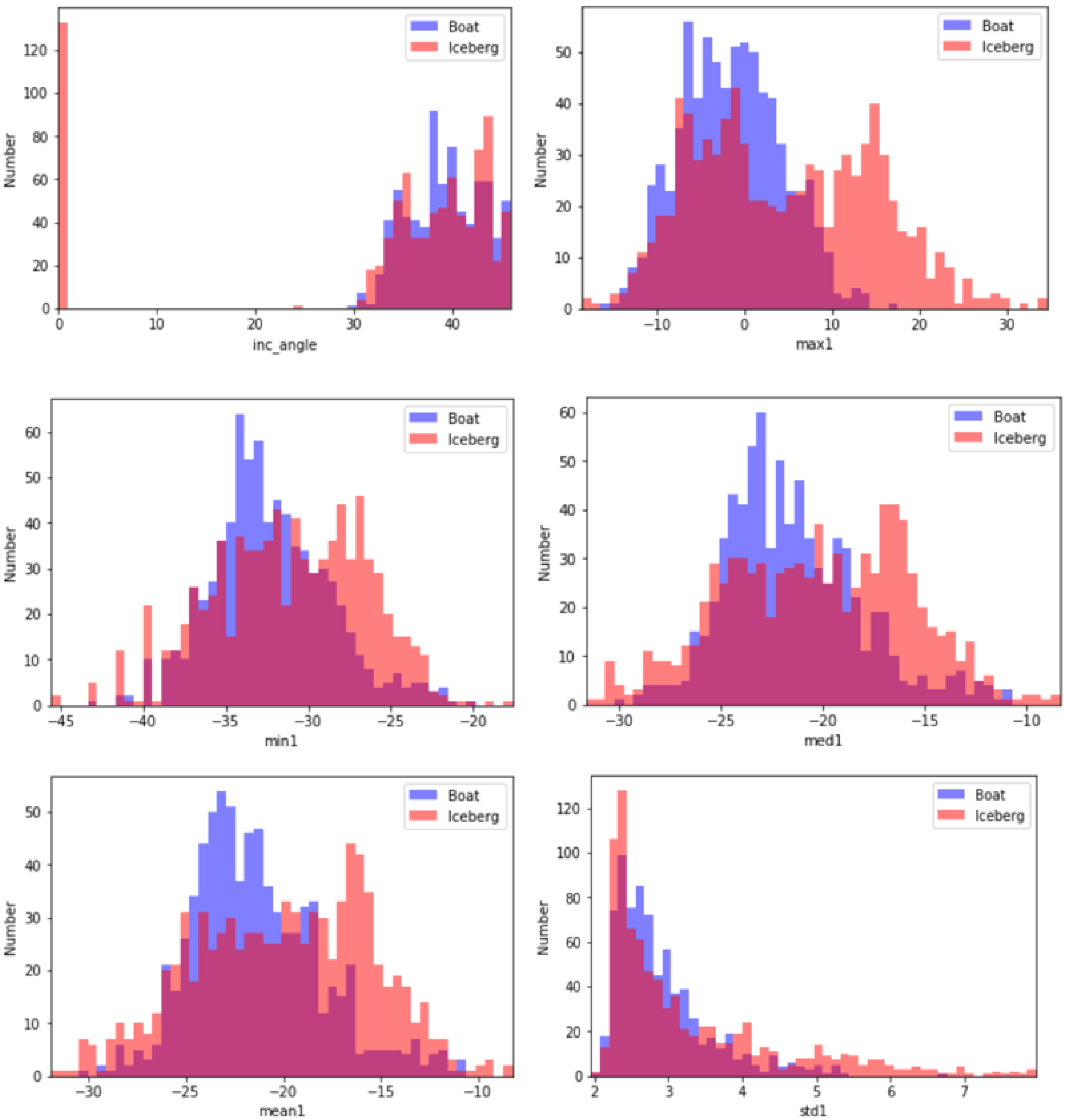


Images of ships in band_2:



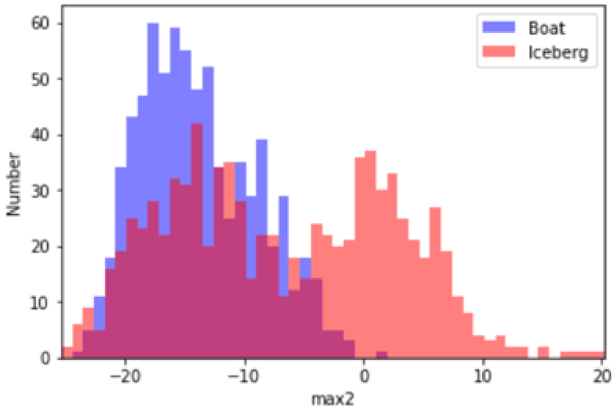
Histograms for band_1:

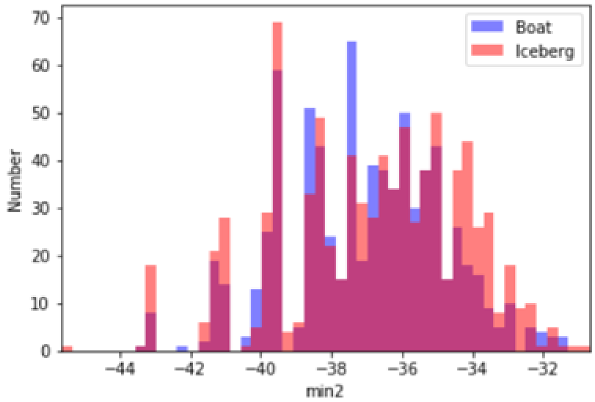
The histograms of inc_angle, max values, min values, median values, mean and standard deviation as below for band_1 boat and ship images.

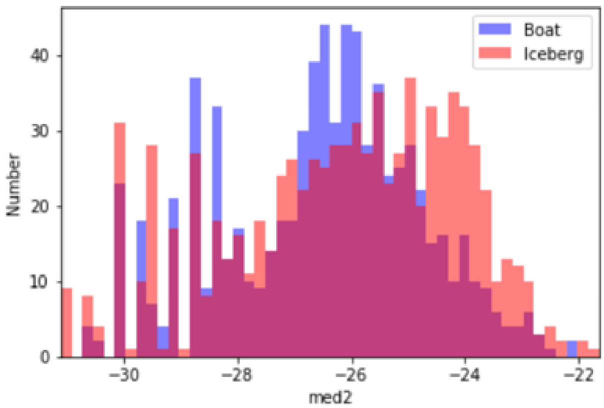


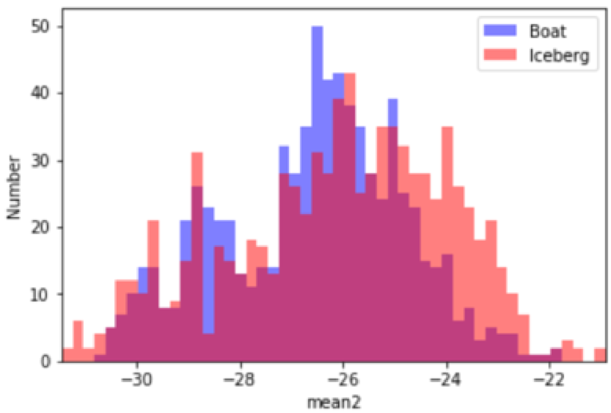
Histograms for band_2:

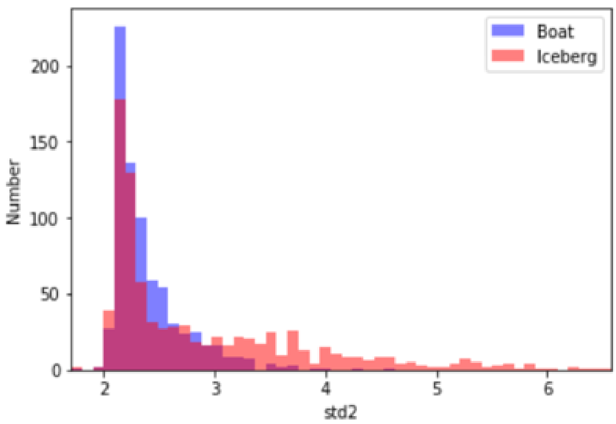
The histograms of max values, min values, median values, mean and standard deviation as below for band_2 boat and ship images.





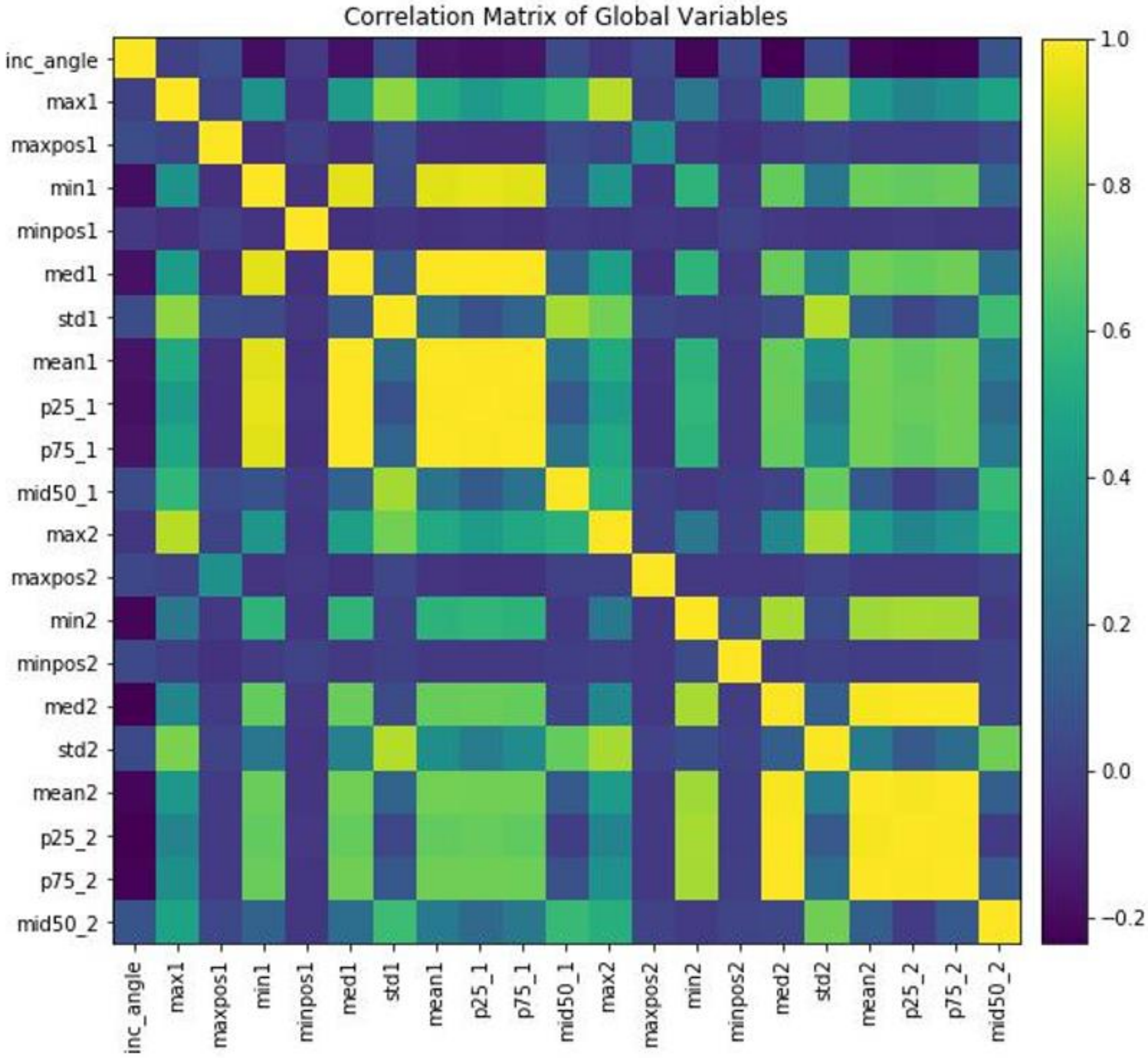






From, the histogram plots it is clear that, both bands have similar results.

Correlations between different 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.

Basic Architecture of CNN

The basics of a CNN architecture consist of 3 components. A **convolution**, **pooling**, and **fully connected layer**. These components work together to learn a dense feature representation of an input.

1. **Convolution:**

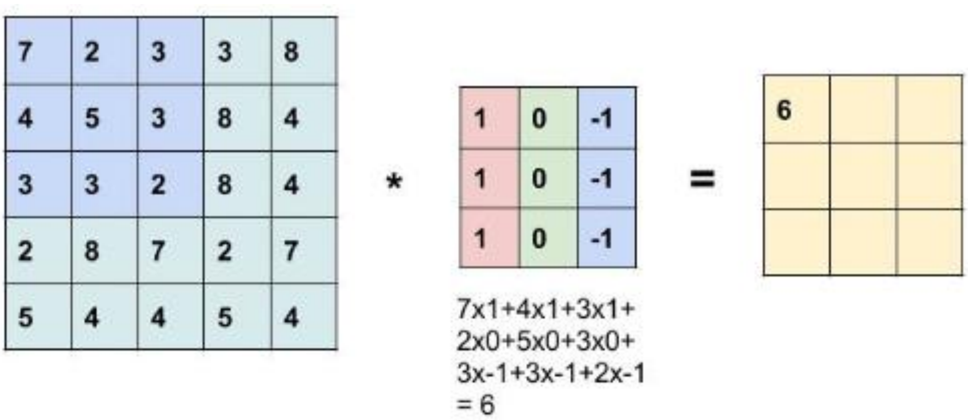
A convolution consists of a **kernel**, also called **filter**, that is applied in a sliding window fashion to extract features from the input. This filter is shifted after each operation across the input by an amount called **strides.** At each operation, a matrix multiply (dot product) of the kernel and current region of input is calculated. Filters can be stacked to create high-dimensional representations of the input.

2. **Strides:**

After we choose the filter size, we also choose the **stride.**

Stride controls how the filter convolves around the input volume. The filter convolves around the input volume by shifting one unit at a time. The amount by which the filter shifts is the **stride**.

Convolution example: The below example is a convolution layer, where a 3x3 filter is applied to a 5x5 image and stride 1.



(https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/)

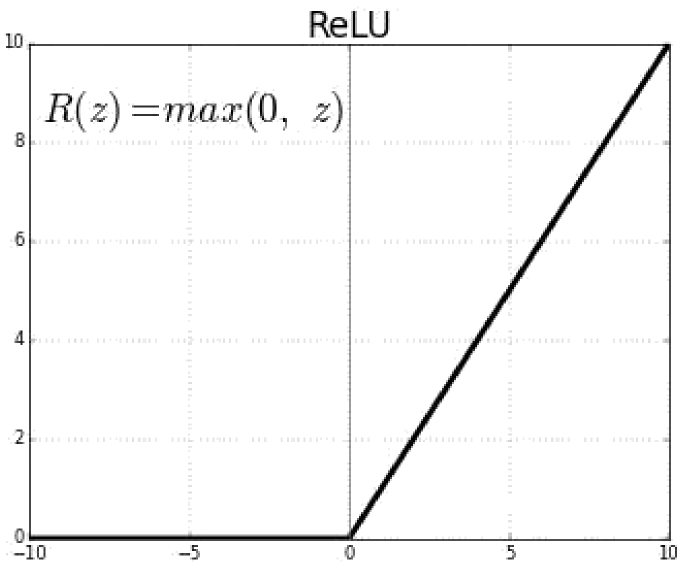
3. **ReLU layer:**

The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers.

In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that **ReLU layers** work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy.

It also helps to remove **the vanishing gradient problem**, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers.

The ReLU layer applies the function **R(z) = max(0, z)** to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0.



(https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec)

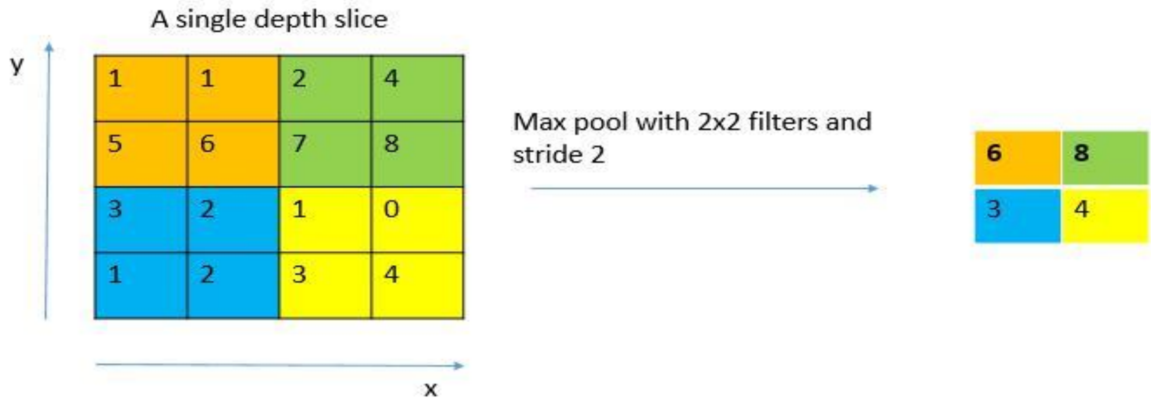
4. **Pooling:**

Pooling is an operation to reduce dimensionality. It applies a function summarizing neighboring information. Two common functions are max pooling and average pooling. By calculating the max of an input region, the output summarizes intensity of surrounding values.

Pooling layers also have a kernel, padding and are moved in strides. To calculate the output size of a pooling operation, you can use the formula,

(Input Width - kernel width + 2 * padding) / strides + 1.

Example: Max pooling applied to below 4x4 image with stride 2 and 2x2 filters. So, (4 – 2 + 2 * 0) / 2 + 1 = 2



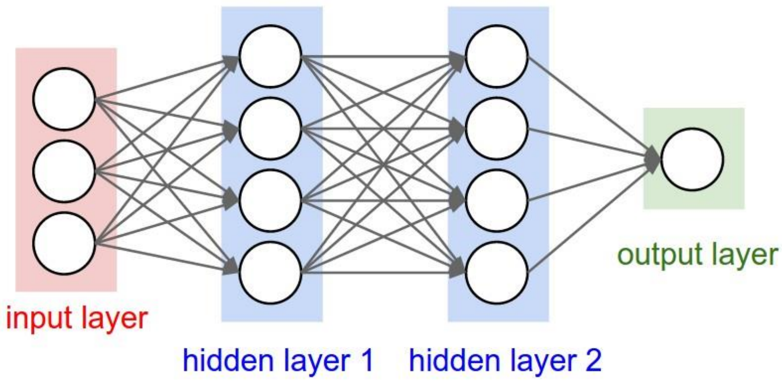
(https://de.wikipedia.org/wiki/Convolutional_Neural_Network)

5. **Fully Connected layer:**

   Fully connected layers are neural networks, where Each neuron in the input is connected to each neuron in the output; fully-connected. Due to this connectivity, each neuron in the output will be used at most one time.

   In a CNN, the input is fed from the pooling layer into the fully connected layer.

   In the below figure each neuron is fully connected the neurons in the next layers, except the output layer.

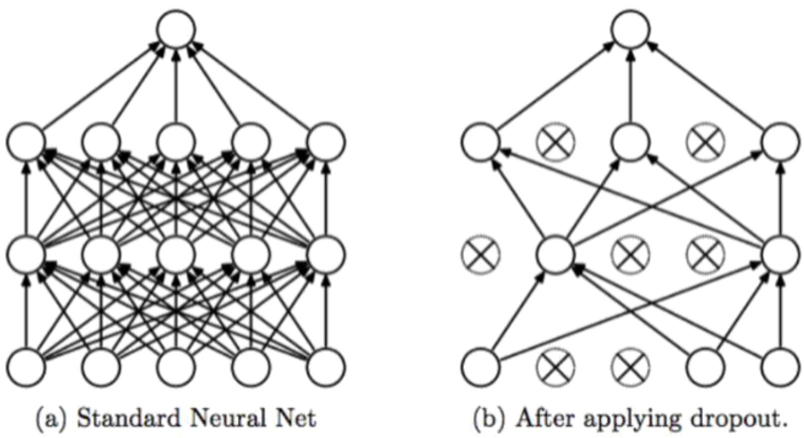


(http://cs231n.github.io/convolutional-networks/)

6. **Dropout layer:**

Dropout layer used to **reduce overfitting**.

Dropout prevents complex co-adaptation on the training data.
Dropout is a technique where **randomly selected neurons are ignored during training**. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the



neuron on the backward pass.

(https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/deep_learning.html)

Transfer Learning:

Transfer learning is the process of taking a pre-trained model (the weights and parameters of a network that has been trained on a large dataset by somebody else) and "fine-tuning" the model with your own dataset.

The idea is that this pre-trained model will act as a feature extractor. Remove the last layer of the network and replace it with your own classifier (depending on what your problem space is). You then freeze the weights of all the other layers and train the network normally (Freezing the layers means not changing the weights during gradient descent/optimization).

The VGG16 pre-trained model used for transfer learning.

Data Augmentation Techniques:

when a computer takes an image as an input, it will take in an array of pixel values.

Let's say that the whole image is shifted left by 1 pixel. To you and me, this change is imperceptible. However, to a computer, this shift can be significant as the classification or label of the image doesn't change, while the array does.

Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as **data augmentation** techniques.

Stochastic gradient descent:

SGD computes the gradient of the parameters using only a single or a few training examples.

Generally, each parameter update in SGD is computed w.r.t a few training examples or a minibatch as opposed to a single example.

The reason for this is twofold:

- First this reduces the variance in the parameter update and can lead to more stable convergence,
- Second this allows the computation to take advantage of highly optimized matrix operations that should be used in a well vectorized computation of the cost and gradient.

The SGD parameters as below,

Learning rate: In SGD the **learning rate α** is typically much smaller than a corresponding learning rate in batch gradient descent because there is much more variance in the update.

Momentum: If the objective has the form of a long shallow valley leading to the optimum and steep walls on the sides, standard SGD will tend to oscillate across the narrow valley since the negative gradient will point down one of the steep sides rather than along the valley towards the optimum. The objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains. **Momentum** is one method for pushing the objective more quickly along the shallow valley.

Decay: The earning rate decay over each update.

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:

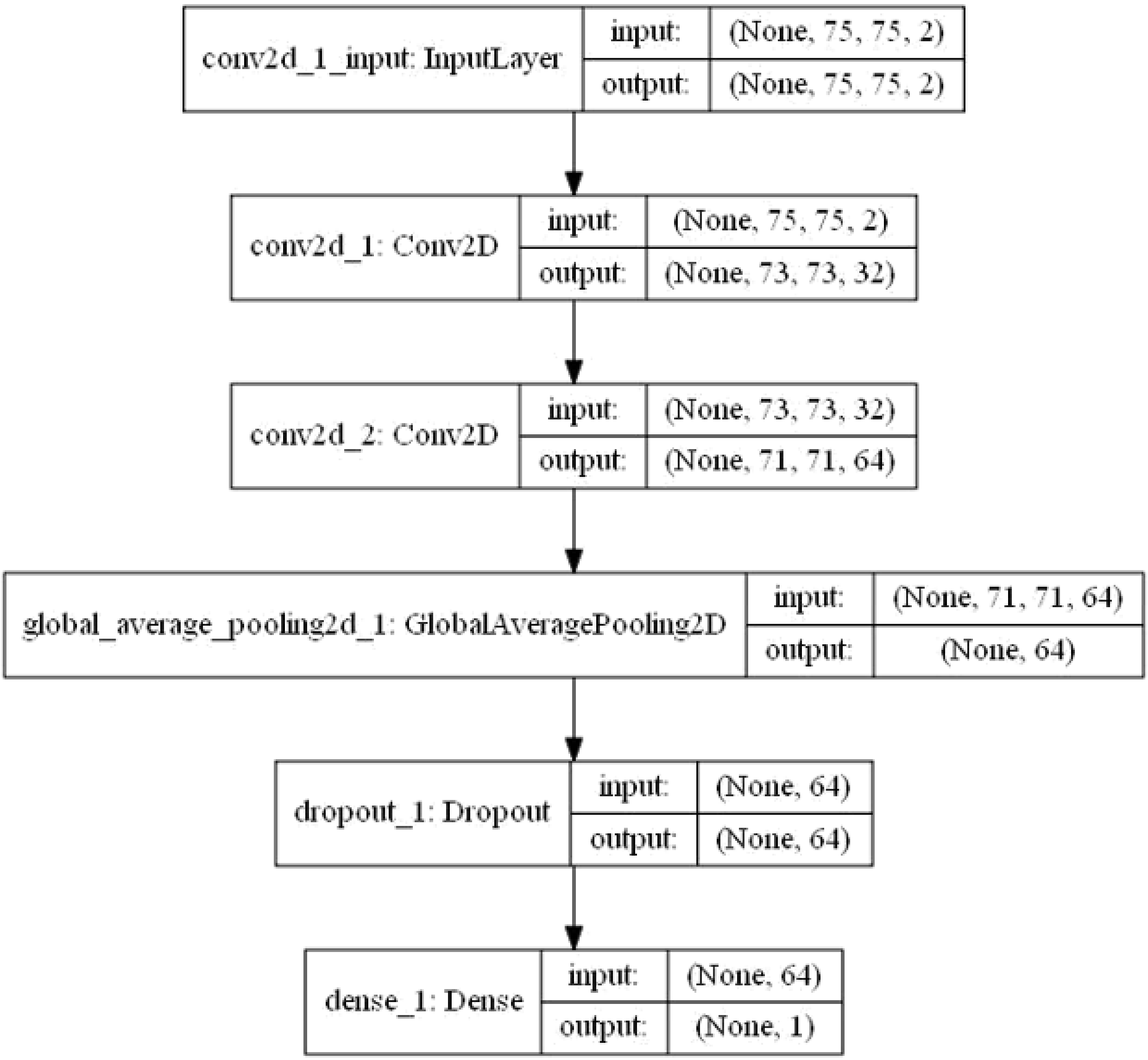
1. **Training parameters:**
    - **Stochastic gradient descent parameters:** o lr:
      float >= 0. Learning rate.
      - o momentum: float >= 0. Parameter updates momentum.
      - o decay: float >= 0. Learning rate decay over each update.
      - o nesterov: boolean. Whether to apply Nesterov momentum.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

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 a validation accuracy of 60.60%.

III. Methodology

Data Preprocessing

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,
   - horizontal_flip: Randomly flips images horizontally.
   - vertical_flip: Randomly flip images vertically.
   - width_shift_range: Range for random horizontal shifts.
   - height_shift_range: Range for random vertical shifts.
   - channel_shift_range: Range for random channel shifts.
   - VI.zoom_range: Range for random zoom.
   - 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 accurancy: 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 83.80%.

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: 0.141631368983
Train accuracy: 0.941066417268
Test loss: 0.181382532042
Test accuracy: 0.923364484756
```

Fold 1:

```
Train loss: 0.139042855828
Train accuracy: 0.942937324602
Test loss: 0.175567350711
Test accuracy: 0.936448599022
```

Fold 2:

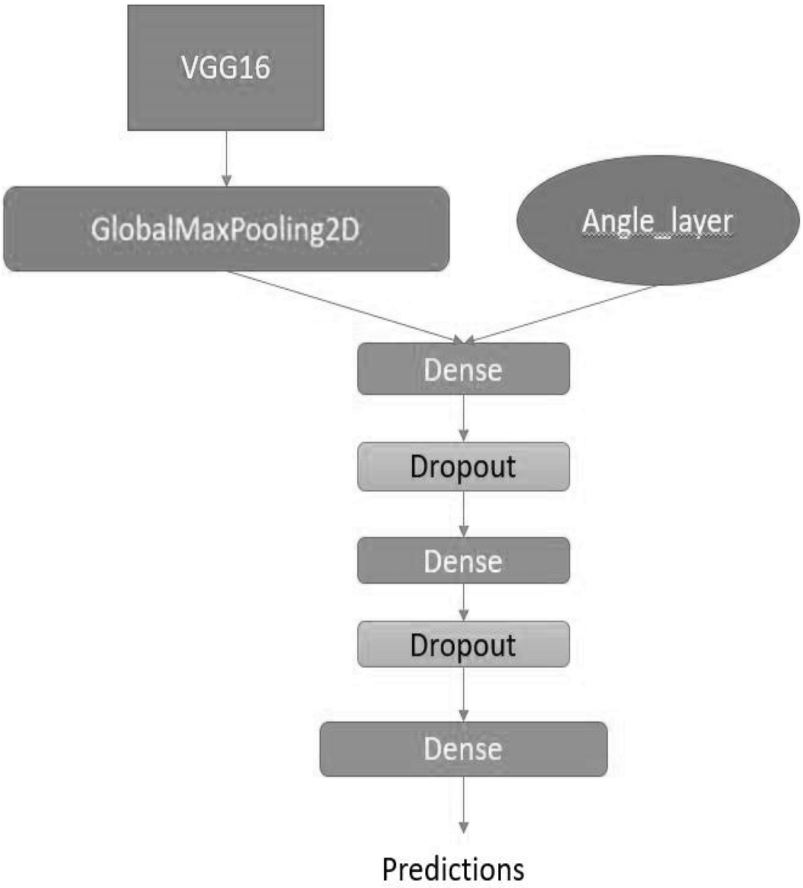
```
Train loss: 0.169280481227
Train accuracy: 0.928971962617
Test loss: 0.266447290946
Test accuracy: 0.897003745765
```

Taking average of the test accuracy of the 3 folds, we got 91.89% accuracy, which is better than the second model.

IV. Results

Model Evaluation and Validation

The final models structure looks like below,



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 91.89%.

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 60.60% , while the final model has test accuracy of 91.89%.

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.

Why this is a robust image classification model?

1. Previously, many research papers cited that, image augmentation gives better accuracy than, with no augmentation. Refer: http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf

So, the final model uses image augmentation to improve the accuracy of the model.

In real world scenarios, the images can in different rotation angle, different brightness, different contrast etc.

Image augmentation introduces some variations to the input data and model built upon this data performs better in real world scenarios.

2. The VGG16 model is a 16-layer network, which is used for transfer learning in final model.

The research paper for VGG16 is "Very Deep Convolutional Networks for Large-Scale Image Recognition, by K. Simonyan, A. Zisserman". Link: - https://arxiv.org/abs/1409.1556

The VGG model is trained on large dataset and have very less error. So, instead of building and train the CNN network from scratch, use a pre-trained model's like VGG16's weights, which will give performance boost and better accuracy.

3. I used dropout layers in last layers to prevent overfitting of the model and it prevents complex co-adaptation on the training data. Two dropout layers added after the VGG16 model to prevent overfitting.

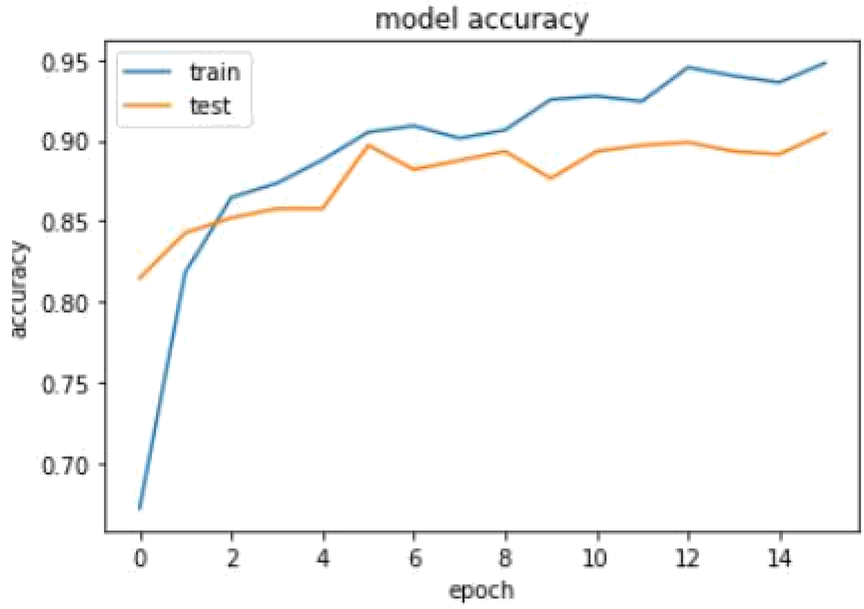
If we look at the log loss of train and test data,

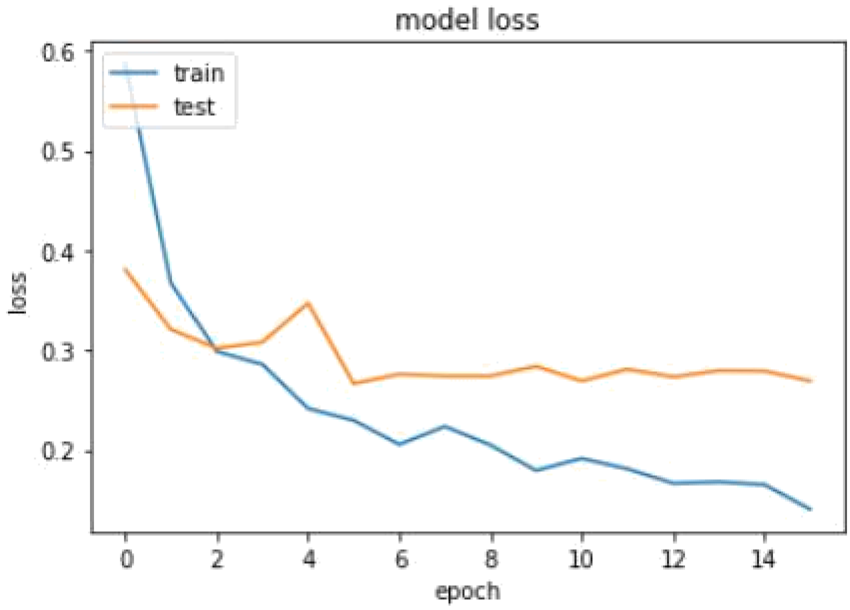
```
Train Log Loss Validation= 0.1548120401
Test Log Loss Validation= 0.207762497157
```

The less the value of log loss, the better model.

Also, final model achieved 91.65% accuracy.

The plot of model accuracy and model loss for final model as below,

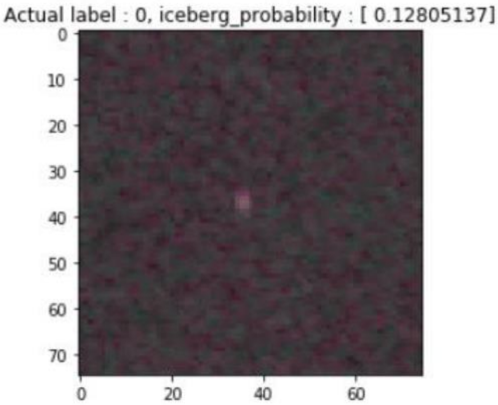




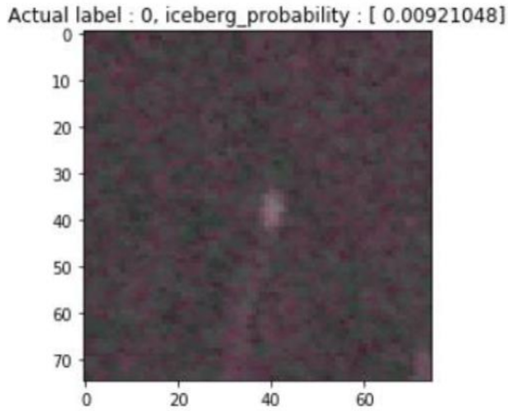
V. Conclusion

Free-Form Visualization

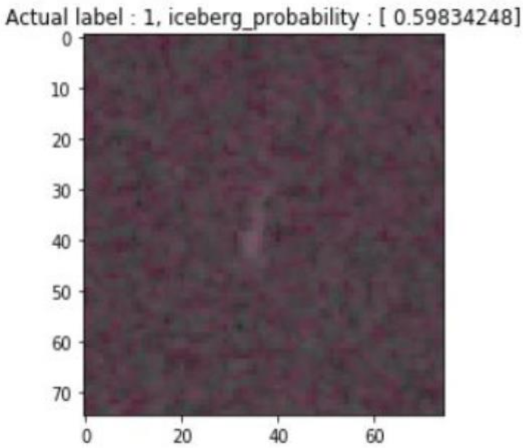
Let’s check some images and its actual label and predicted probability as below,



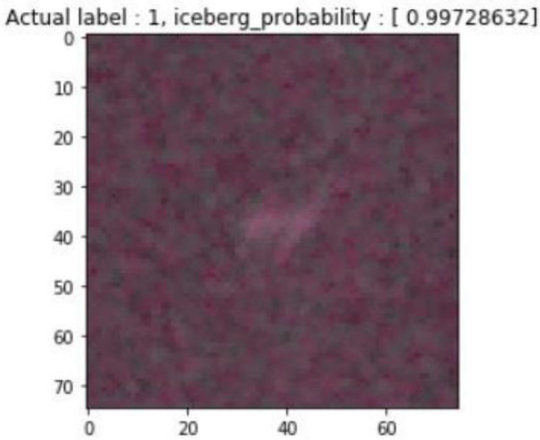
Correct classification



Correct Classification



Correct classififcation.



Correct Classififcation

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 types of problems, 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.