Machine Learning Engineer Nanodegree

Capstone Project

Shehabul Hossain
November 25th, 2017

I. Definition

Project Overview

Image classification has been playing a vital role in Computer Vision. As computers are getting better and with more computational power, computer vision has also improved significantly. AI and Machine Learning has also played a vital role in computer vision. Identifying objects in an image, extracting information from text in an image, describe the whole image has become reality.

In this project, I created an iceberg classifier that can detect if an image that was taken from satellite has iceberg in it or not. This system takes an images as in input and returns the probability of the image containing an iceberg.

Problem Statement

Drifting icebergs are one kind of threat to navigation and activities in offshore areas. It can do serious damage to passing ships. Many companies use aerial reconnaissance and shore-based support to monitor environmental conditions and assess risks from icebergs. However, in harsh weather conditions, the only way is to monitor using satellite. Still, the data has to be processed manually in order to differentiate icebergs from other objects like a ship. It is a very tedious job to classify icebergs using satellite signals. To solve this problem machine learning can be used. The satellite collects data as an image and the objective is to create an image classifier that can find icebergs in images. A CNN can be used in this case as CNN's are very good at classifying images. The CNN model will take an image as an input and look for icebergs in that image. The output will be a number between 0 and 1 which will prepresent the probability that the image contains an iceberg.

Metrics

The results will be evaluated on the log loss between the predicted values and the ground truth. For each image in the test data there will be a predicted value from 0 to 1 which will represent the probability of the image containing an iceberg.

In multi-class version of the log loss metric at each observation is in one class and for each observation there is a output probability for each class. The metric is negative the log likelihood of the model that says each test observation is chosen independently from a distribution that places the submitted probability mass on the corresponding class, for each observation.

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

where N is the number of observations, M is the number of class labels, *log* is the natural logarithm, $y_{i,j}$ is 1 if observation i is in class j and 0 otherwise, and $p_{i,j}$ is the predicted probability that observation i is in class j.

Finally, we will get the validation accuracy that will tell us the percentage of images that are being predicted correctly. ## II. Analysis

Data Exploration

Statoil, an international energy company operating worldwide, has worked closely with companies like C-CORE. C-CORE has been using satellite data for over 30 years and has built a computer vision based surveillance system. To keep operations safe an efficient a more efficient system can be implemented using machine learning. The company released the data in a Kaggle competition to find an efficient solution using machine learning.

The satellites that are used to detect icebergs are 600 kilometers above the earth using a radar that bounces a signal off an object and records the echo, then the data is translated into an image. The C-Band radar operates at a frequency that can see through darkness, rain, cloud and even fog. Echos from different objects are recorded and then 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. Many things include winds affect the backscatter. High winds generate a brighter background and low winds generate darker. The Sentinel-1 satellite is a side-looking radar, which means it sees the image area at an angle (incidence angle). Generally, the ocean background will be darker at a higher incidence angle. You also need to consider the radar polarization, which is how the radar transmits and receives the energy. More advanced radars like Sentinel-1 can transmit and receive in the horizontal and vertical plane. Using this, you can get what is called a dual-polarization image.

Here, we have data with two channels: HH(transmit/received horizontally) and HV(transmit horizontally and received vertically). This can play an important role in classifying1 as different objects tend to reflect energy differently. All the images are 75x75 images with two bands and we also have inc_angel which incidence the angel of which the image was taken.

We have two data files (train.json, test.json). The files consist of a list of images and for each image, we have the following fields:

- 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).
- inc_angle - the incidence angle of which the image was taken. This field has some 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. The train.json has 1604 rows and the test.json has 8424 rows.

Exploratory Visualization

As we have described the data that every image in the dataset has 5625 pixels in each band, we will look more in those features and plot them as an image to get a better understanding of the dataset.

First of all, we take 18 random images from the train data where 9 of them are images of non-icebergs and rest 9 are images of icebergs. Then, we reshape both band_1 and band_2 array into 75x75 numpy array and then we plot them to get a better understanding.

Images of Iceberg

Let’s plot images of iceberg first. We took band_1 for those 9 iceberg images and then plotted to see how they look. The plot looks like following.

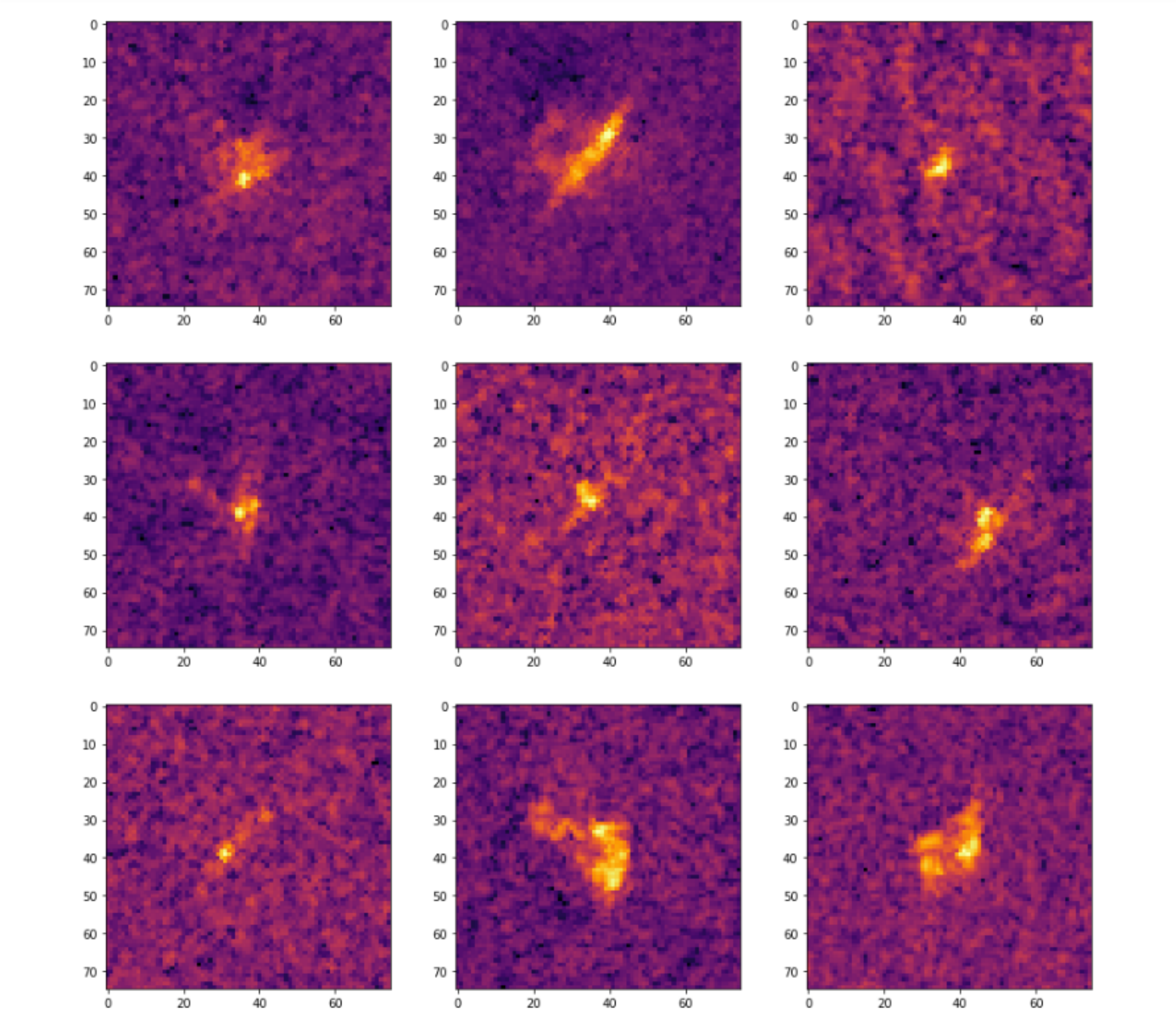


Image of Icebergs from Band_1

-

We can clearly see those images has a yellowish object in the center. As these images are labeled as icebergs, we can say those objects are actually iceberg.

Let’s plot the same images for band_2.

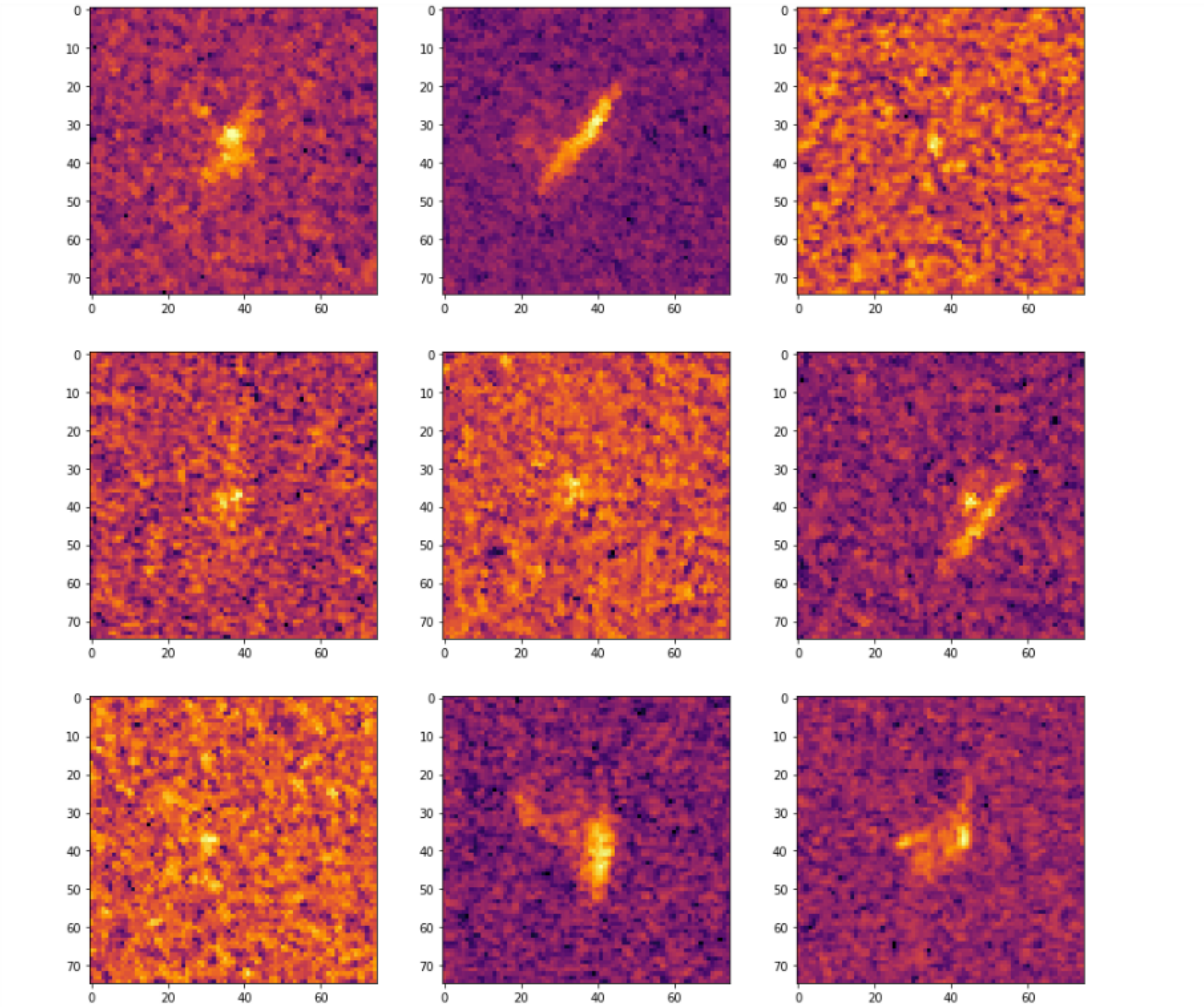


Image of Icebergs from Band_2

-

In this case, we see the same object in the center, however, in some cases, the noise in the background is too high.

Now, let's plot some images of **non-icebergs**. We take the same approach as before to take 9 random images of non-icebergs from train data and then plot them for both band_1 and band_2.

For band_1 the plot looks like following.

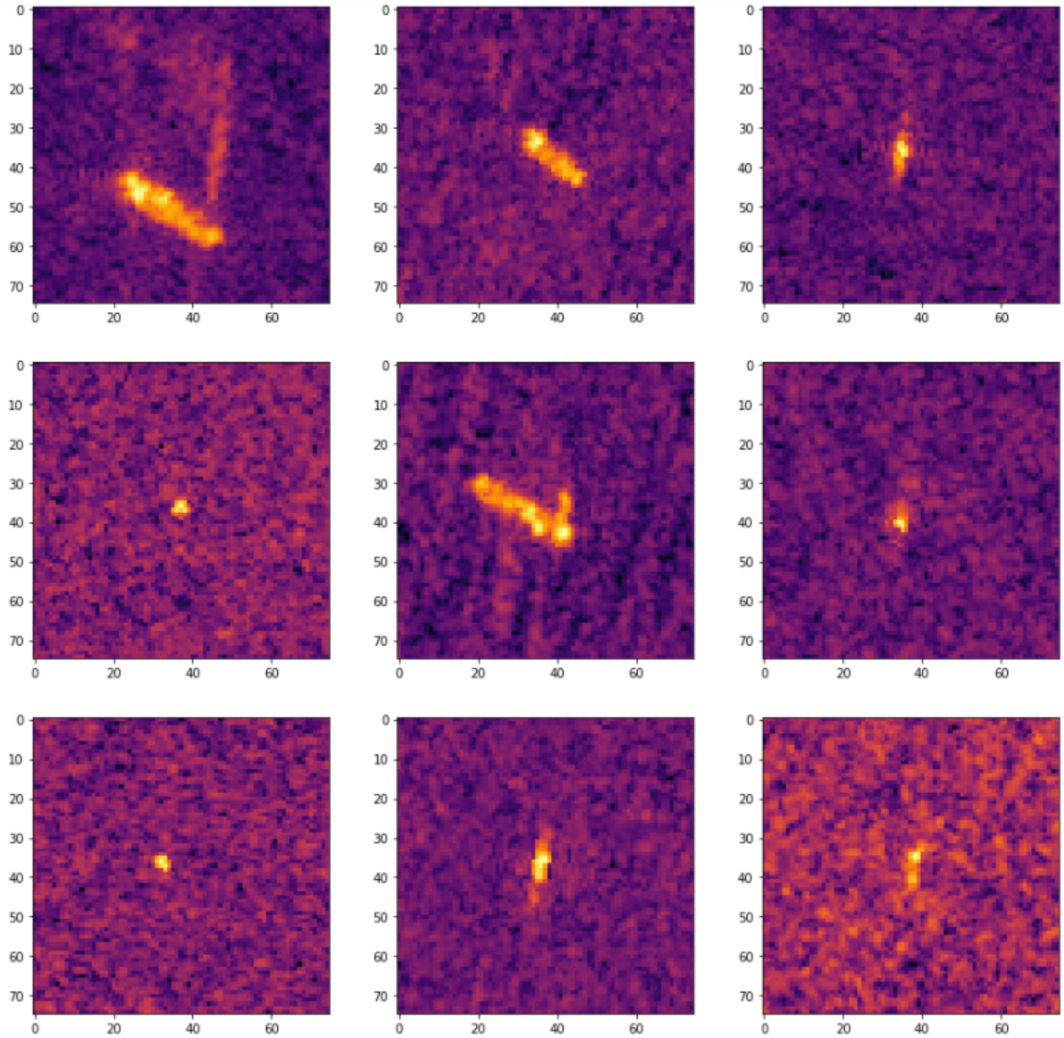


Image of non-icebergs from Band_1

-

In this case, we see 9 images of non-icebergs. We can identify most of them as non-icebergs however, some of them are hard to distinguish even for humans.

For band_2 the plot looks like following.

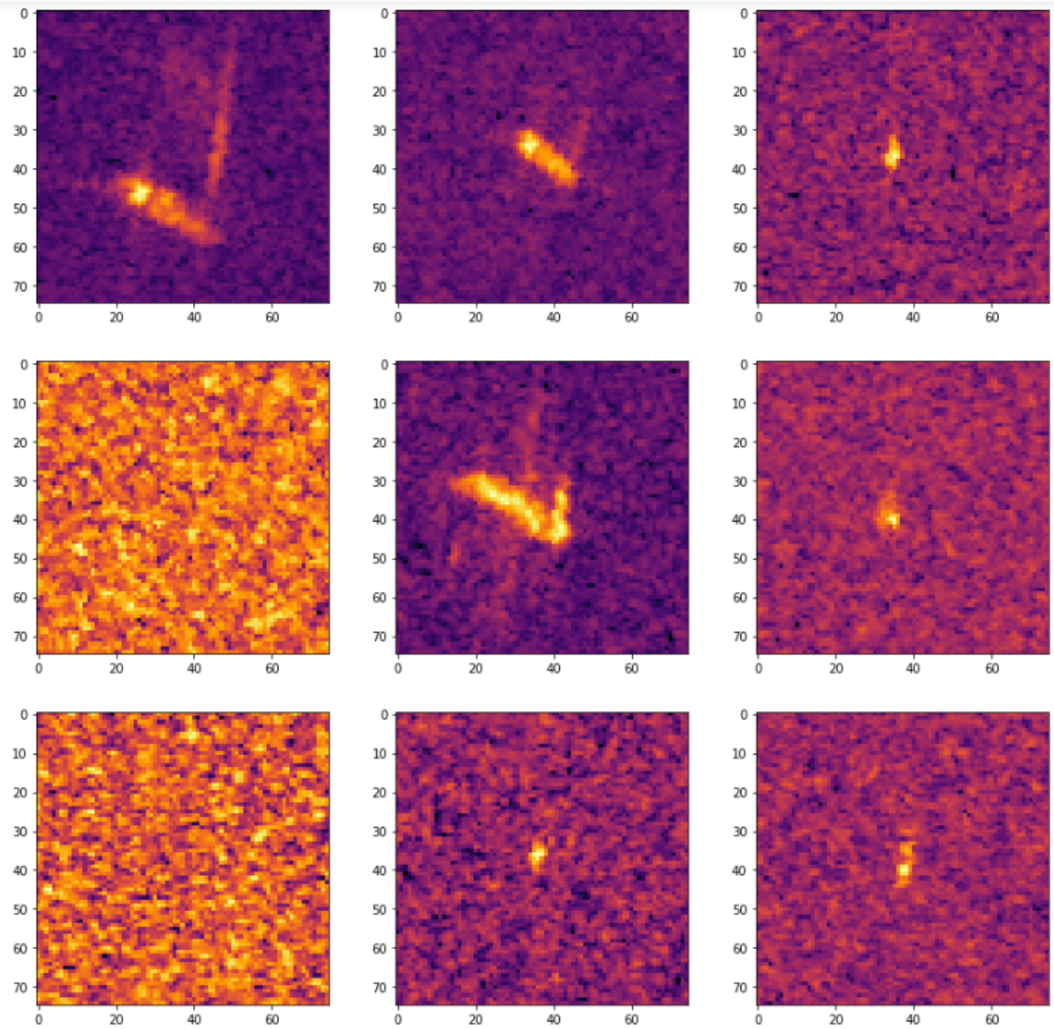


Image of non-icebergs from Band_2

-

Again, some of these images have higher background noise compared to band_1.

In order to minimize overfitting, we combine both band_1, band_2 and their average to generate a 3 channel image of the data.

Algorithms and Techniques

I used Convolutional Neural Networks, which is the state-of-the-art algorithm for most of the image processing tasks out there. This algorithm takes images as input and then the data is fed through some layers of neurons including convolutional layers to generate output. In our case, we used a sigmoid layer at the end to generate a value from 0 to 1 which represents the probability. CNN has some parameters that can be tuned. These are following:

- number of epochs/number of training iterations
- karnel_size of each convolutional layer
- activation function of each convolutional layer
- pool_size of each pooling layer
- strides of each pooling layer

- optimizer function
- loss function
- metric function (used to judge the performance of the model)

Benchmark

For benchmark model, I created a Convolutional Neural Network with only 1 convolutional layer, 1 dense layer. Finally, a sigmoid layer is added to generate output. This is a minimalistic CNN that can be used as a basic image classifier. With this model and 10 epochs, I have got minimum loss of 8.4681 with a validation accuracy of 0.4688.

III. Methodology

Data Preprocessing

First of all, We checked for any missing data especially in band_1 and band_2 as they are the features that we want to use. Fortunately, there was no missing data for both bands. In each image, the band_1 and band_2 has been flattened. So, we need to create 75x75 array from each band to represent them as images. With numpy we have created 75x75 array from each band. Then, we want to create an image that contains data from both bands. So, we created a 3 channel image with band 1 and band 2 as first two channels and their average as the third channel. Finally, I have created an array with dimension of 75x75x3 that represents each image in the dataset.

Implementation

The implementation process is divided into two parts. - Training the classifier - Prediction

In order to train the classifier, a CNN model was trained on the preprocessed data. This is done in the Jupiter notebook titled "Iceberg Classifier". Let's get into further details.

1. Load both train and test data in the memory. Look for missing data.
2. To get a better understanding of the data some random data from the dataset are taken and plotted them for visualization.
3. Now, I defined the architecture of a CNN model that looks like following.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 73, 73, 64)        1792
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
dropout_19 (Dropout)         (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 16, 16, 128)       0
_________________________________________________________________
dropout_20 (Dropout)         (None, 16, 16, 128)       0
_________________________________________________________________
flatten_5 (Flatten)          (None, 32768)             0
_________________________________________________________________
dense_11 (Dense)             (None, 256)               8388864
_________________________________________________________________
dropout_21 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 257
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0
=================================================================
Total params: 8,464,769
Trainable params: 8,464,769
Non-trainable params: 0
_________________________________________________________________
```

CNN model for iceberg classification

This architecture which is a sequential model has 2 convolutional layers each has kernel size of 3, a 'relu' activation function followed by a max pooling layer that has pool size of 2 and strides of 2. And then a dropout layer is added to drop 20% of the incoming flow. The first convolutional layer has 64 nodes and the second one has 128 nodes. After that, a flatten layer is added to flatten the data followed by a fully connected dense layer of 256 nodes. Again, a dropout layer is added to overcome overfitting. Finally, all the nodes are connected to a single dense layer and a sigmoid activation function.

4. A loss function is defined, in this case, "binary_crossentropy" loss function is used.
5. We used an Adam optimizer as the optimizer function with a learning rate of 0.001, beta_1 of 0.9, beta_2 of 0.999, epsilon value of 1e-08 and decay value of 0.
6. Finally, we compiled the model with a metric of 'accuracy'.
7. For training and validation we used a train_test_split function from sklearn to split the preprocessed data with a ratio of 0.75:0.25 for training and validation dataset.
8. Now we fit the train data with the training label into the fit function with a batch size of 24 and epochs value of 10 and validate the data with the validation set in each epoch. Finally, we save the best performance into an hdf5 file.

Refinement

In order to improve accuracy, I modified the CNN architecture. Previously we had 2 convolutional layers. We added 2 more convolutional layers and instead of 1 fully connected layers, we used 2 fully connected dense layers. The architecture looks following.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 73, 73, 64)        1792
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
dropout_22 (Dropout)         (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 16, 16, 128)       0
_________________________________________________________________
dropout_23 (Dropout)         (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 14, 14, 256)       295168
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 6, 6, 256)         0
_________________________________________________________________
dropout_24 (Dropout)         (None, 6, 6, 256)         0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 4, 4, 64)          147520
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 1, 1, 64)          0
_________________________________________________________________
dropout_25 (Dropout)         (None, 1, 1, 64)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_13 (Dense)             (None, 256)               16640
_________________________________________________________________
dropout_26 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_14 (Dense)             (None, 512)               131584
_________________________________________________________________
dropout_27 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 513
_________________________________________________________________
activation_6 (Activation)    (None, 1)                 0
=================================================================
Total params: 667,073
Trainable params: 667,073
Non-trainable params: 0
_________________________________________________________________
```

Modified CNN model for iceberg classification

As we added 2 more convolutional layers, the third one has 256 nodes with the same kernel size of 3, 'relu' activation function as before. The forth convolutional layer, however, has 64 nodes and all the other configure are same as others. Both of these new layers each have one max pooling layer with a pool size of 3 and strides of value 2 and a dropout layer with 20% dropout added to them. after these 3 convolutional layers, we flattened them and added two fully connected dense layers each with 'relu' activation function and a 20% dropout layer added next to them. The first dense layer has 256 nodes and the second one has 512 nodes. Finally, we added a dense layer with 1 node and added a 'sigmoid' function added to it. Also, during the architecture, we didn't change any configuration from before however we increased epochs from 10 to 50.

IV. Results

Model Evaluation and Validation

During the development process, a validation set was used from the train data. During fitting the model we used a loss function of 'binary_crossentropy' and 'accuracy' metrics that help us to evaluate the model. Let's analyze both the architecture that we have defined.

The first model that we created has 2 convolutional layers and a fully connected dense layer with dropout layers attached to them. As the test data is not labeled and we can not really anything about the test data so we relied on the validation data for evaluation. In this case, the lowest loss score we got is 8.3858 with a validation accuracy of 0.4681. Is this is a binary classification 0.4688 is not even a decent score as complete random prediction will give us roughly accuracy of 0.5. The result looks like following.

```
Train on 1203 samples, validate on 401 samples
Epoch 1/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.3802 - acc: 0.4658Epoch 00000: val_loss improved from inf to 8.4
6815, saving model to saved_models/weights.best.from_scratch.hdf5
1203/1203 [==============================] - 57s - loss: 8.3858 - acc: 0.4655 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 2/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4495 - acc: 0.4700Epoch 00001: val_loss did not improve
1203/1203 [==============================] - 52s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 3/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4495 - acc: 0.4700Epoch 00002: val_loss did not improve
1203/1203 [==============================] - 52s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 4/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4495 - acc: 0.4700Epoch 00003: val_loss did not improve
1203/1203 [==============================] - 53s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 5/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4627 - acc: 0.4692Epoch 00004: val_loss did not improve
1203/1203 [==============================] - 54s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 6/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4362 - acc: 0.4708Epoch 00005: val_loss did not improve
1203/1203 [==============================] - 57s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 7/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4627 - acc: 0.4692Epoch 00006: val_loss did not improve
1203/1203 [==============================] - 55s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 8/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4362 - acc: 0.4708Epoch 00007: val_loss did not improve
1203/1203 [==============================] - 56s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 9/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4627 - acc: 0.4692Epoch 00008: val_loss did not improve
1203/1203 [==============================] - 59s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
Epoch 10/10
1200/1203 [============================>.] - ETA: 0s - loss: 8.4627 - acc: 0.4692Epoch 00009: val_loss did not improve
1203/1203 [==============================] - 62s - loss: 8.4549 - acc: 0.4697 - val_loss: 8.4681 - val_acc: 0.4688
```

Result of CNN model for iceberg classification

-

For the **improved** architecture where we increased the number of layers and epochs, the result looks different. This model has 4 convolutional layers and 2 dense layers with a 1 node dense layer at the end with a sigmoid function. As usual, we fit the data and validate with the 25% validation set from the train data. Part of the 50 epochs result looks like following.

```
Epoch 40/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2518 - acc: 0.8850Epoch 00039: val_loss did not improve
1203/1203 [==============================] - 67s - loss: 0.2512 - acc: 0.8853 - val_loss: 0.3347 - val_acc: 0.8254
Epoch 41/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2650 - acc: 0.8775Epoch 00040: val_loss did not improve
1203/1203 [==============================] - 65s - loss: 0.2644 - acc: 0.8778 - val_loss: 0.3660 - val_acc: 0.8454
Epoch 42/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2353 - acc: 0.8908Epoch 00041: val_loss did not improve
1203/1203 [==============================] - 65s - loss: 0.2351 - acc: 0.8911 - val_loss: 0.2759 - val_acc: 0.8703
Epoch 43/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2412 - acc: 0.8833Epoch 00042: val_loss improved from 0.26515 to
0.24999, saving model to saved_models/weights.best.improved.hdf5
1203/1203 [==============================] - 61s - loss: 0.2407 - acc: 0.8836 - val_loss: 0.2500 - val_acc: 0.8903
Epoch 44/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2031 - acc: 0.9075Epoch 00043: val_loss did not improve
1203/1203 [==============================] - 63s - loss: 0.2096 - acc: 0.9069 - val_loss: 0.2671 - val_acc: 0.8953
Epoch 45/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2780 - acc: 0.8733Epoch 00044: val_loss did not improve
1203/1203 [==============================] - 64s - loss: 0.2777 - acc: 0.8736 - val_loss: 0.3020 - val_acc: 0.8753
Epoch 46/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2328 - acc: 0.8958Epoch 00045: val_loss did not improve
1203/1203 [==============================] - 63s - loss: 0.2325 - acc: 0.8961 - val_loss: 0.2880 - val_acc: 0.8803
Epoch 47/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2042 - acc: 0.9117Epoch 00046: val_loss did not improve
1203/1203 [==============================] - 62s - loss: 0.2046 - acc: 0.9111 - val_loss: 0.2648 - val_acc: 0.8853
Epoch 48/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.2396 - acc: 0.8917Epoch 00047: val_loss did not improve
1203/1203 [==============================] - 60s - loss: 0.2402 - acc: 0.8911 - val_loss: 0.3293 - val_acc: 0.8828
Epoch 49/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.1993 - acc: 0.9158Epoch 00048: val_loss did not improve
1203/1203 [==============================] - 61s - loss: 0.1990 - acc: 0.9160 - val_loss: 0.2913 - val_acc: 0.8828
Epoch 50/50
1200/1203 [============================>.] - ETA: 0s - loss: 0.1933 - acc: 0.9183Epoch 00049: val_loss did not improve
1203/1203 [==============================] - 62s - loss: 0.1935 - acc: 0.9185 - val_loss: 0.3063 - val_acc: 0.8778
```

Result of modified CNN model for iceberg classification

-

In this case, we can see that the result improved significantly. We got the lowest validation loss score of 0.2499 and validation accuracy of 0.8953. So we can say that this model predicts the right answer almost 90% of the time on data that it has never seen before.

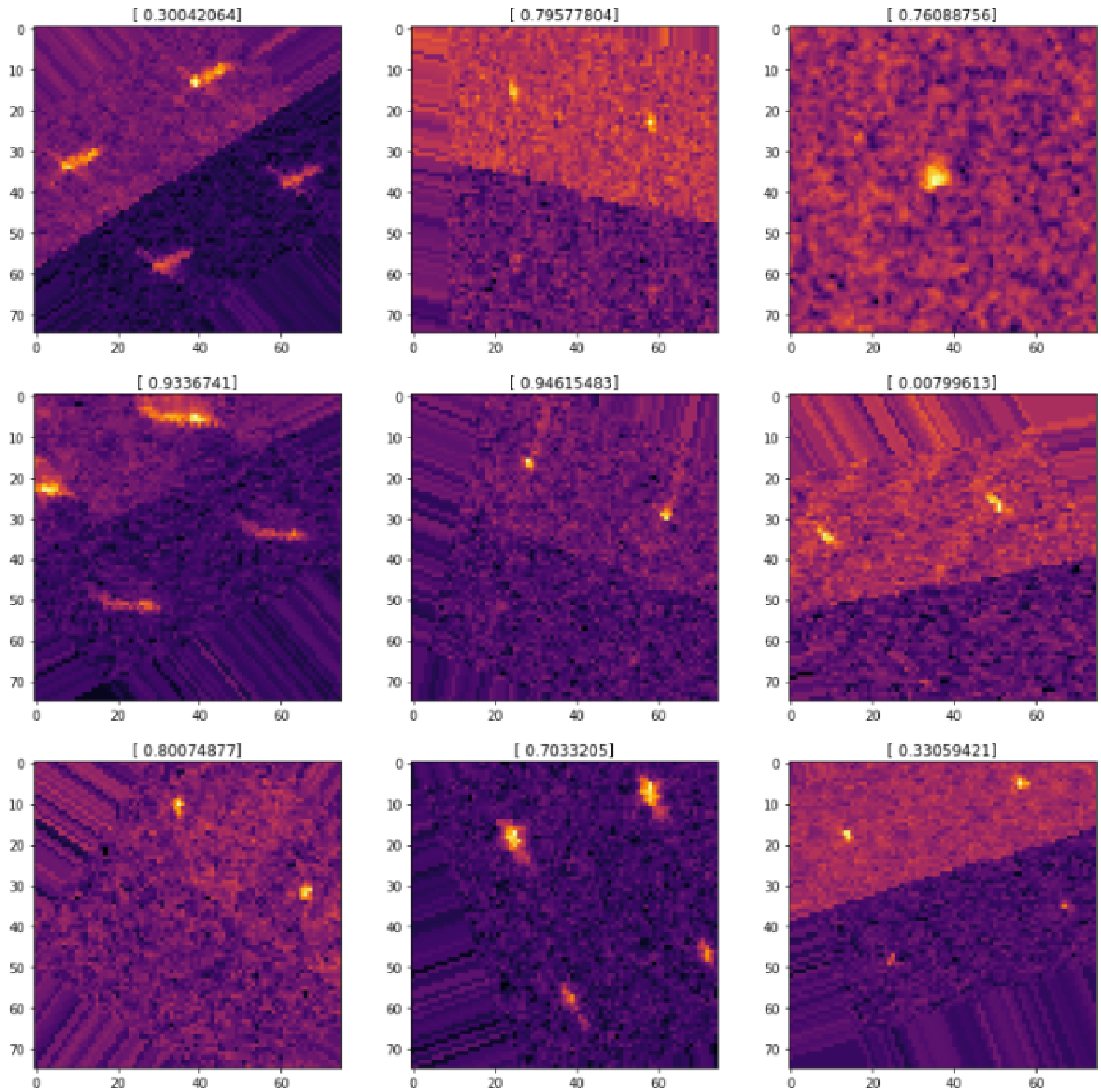
Justification

In our benchmark model, we got a minimum loss of 8.4681 and a validation accuracy of 0.4688 which is very poor given that this is a binary classification problem and even with absolute random prediction we should get an accuracy of 0.5 on a large dataset. However, our benchmark did not do better than that. If we compare this to out improved CNN model with 4 convolutional layers then we can see that out improved model has a loss score of 0.2499 and validation accuracy of 0.8953. This is very impressive as almost 90% of the data is being predicted correctly from images that are sometimes difficult for humans to classify too. I think this solution is significant enough to solve the problem as almost 90% of the unseen data is predicted correctly.

V. Conclusion

Free-Form Visualization

Let's look at some of the images from the test dataset and their predicted values.



Random images from test data with the corresponding prediction

Each image in this plot is created with 3 channels containing band 1, band 2 and their average. And the corresponding value is the predicted probability of the image containing an iceberg. We can see most the images in this plot looks like an iceberg and their corresponding value is close to 1. The first image is predicted as not an iceberg and it looks more like a ship than iceberg.

Reflection

The object of this project was to successfully create a binary image classifier that can classify icebergs from other objects like ships or anything else. However, the interesting part was that although CNN is very effective in image classification, however, in this case, the input data was not really an image. It was actually the reflection of radar signals bounced off the surface and other objects like icebergs and ships. The input was preprocessed from a 75x75 pixel radar image to a flattened array. We reshaped them back into a 75x75 pixel image and then used CNN with multiple convolutional layers and dense layers to successfully classify icebergs. Even with only 1203 images used for training and 401 images to validate, the model did good achieving almost 90% accuracy.

Improvement

Although 90% is good enough with such a small train dataset however it could have been further improved. The number of convolutional layers could be increased to check if that gives a better performance. Moreover, the test data has been totally unused in this case. We could make a semi unsupervised model that could use the test dataset as well. For example, now that we have a model that classify with an accuracy of 90%, we could classify batch of images from test dataset and then put them in the training dataset to further improve the prediction accuracy of our model. I think that would make the accuracy even better.