Statoil/C-Core Iceberg Classifier Challenge (Kaggle)

Deepanshu Saini

Table of Contents

Data

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.

Statoil, an international energy company operating worldwide, has worked closely with companies like C-CORE. C-CORE have been using satellite data for over 30 years and have 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.

In this competition, you're challenged to build an algorithm that automatically identifies if a remotely sensed target is a ship or iceberg. Improvements made will help drive the costs down for maintaining safe working conditions.

The labels are provided by human experts and geographic knowledge on the target. All the images are 75x75 images with two bands. [1]

Data fields

train.json, test.json

The data (train.json, test.json) is presented in json format. The files consist of a list of images, and for each image, you can find 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). 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.[1]

Methodology

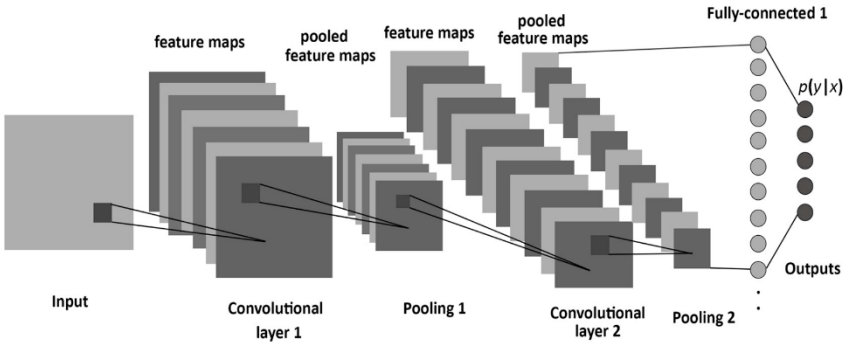
Pre-Processing

The Data Preprocessing involved the following steps:

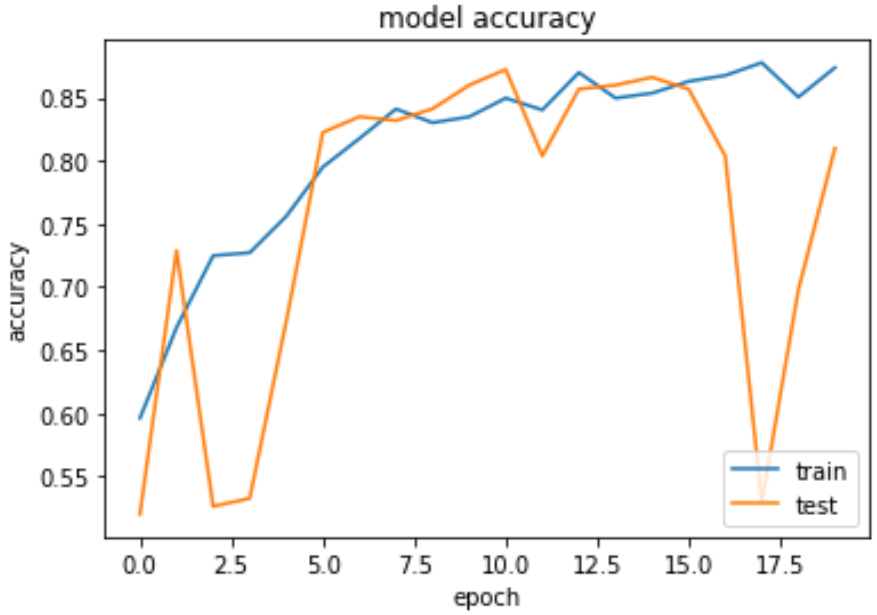
1. Loading the dataset (Both train and test)
2. Extracting the labels from the dataset
3. Extracting the 2 bands i.e.75x75 images set from the dataset
4. Adding a 3rd band to the training dataset
5. Aggregating the features by appending the 3 bands into a single array to create a RGB equivalent for the model.fit_generator method
6. Sampling the data using the train_test_split method
7. Extracting the normalization parameters i.e. min, max and mean values
8. Scaling the training and test datasets using the normalization parameters extracted in the previous step to prevent any form of data leak
9. The tools and methods used for data pre-processing are:
    - pandas for data structuring
    - numpy for linear algebra (mostly dealing with matrices)
    - sklearn for log loss metrics for this competition
    - keras for configuring the CNN model and pipelining the data augmentation process

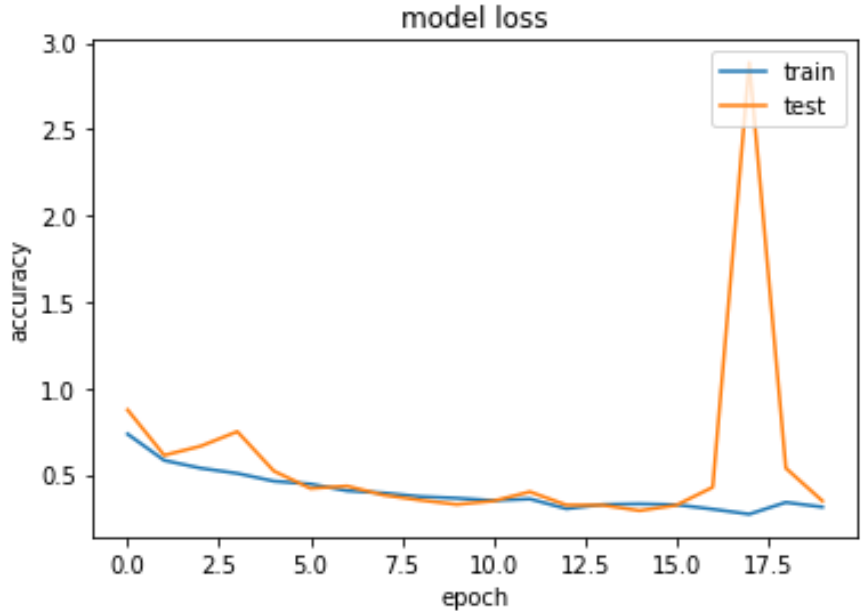
Baseline Model

1. The baseline model used is a CNN with 3 phases of Convolutional and Pooling layers and 2 Hidden layers of fully connected Neural Nets
2. Convolutional layers use a kernel size of 3*3 and Max Pooling layers use a pool size of 2*2 with strides of 2*2
3. The ‘Relu’ activation function is used for both convolutional and pooling layers & for the fully connected layer neurons
4. The output neuron uses the ‘Sigmoid’ activation function
5. Here we use ‘Adam’ optimization technique, with ‘binary_crossentropy’ as the loss model, for finding the optimized loss function



6. The model obtained a final training accuracy score of **acc: 0.8737** higher than the validation accuracy of **val_acc: 0.8100**, which reflects overfitting
7. The final validation loss calculated was **val_loss: 0.3473**
8. As we can see in the models plots below, the CNN model overfitted the data as the train accuracy is higher than that of the test (validation) accuracy.





Model Improvements

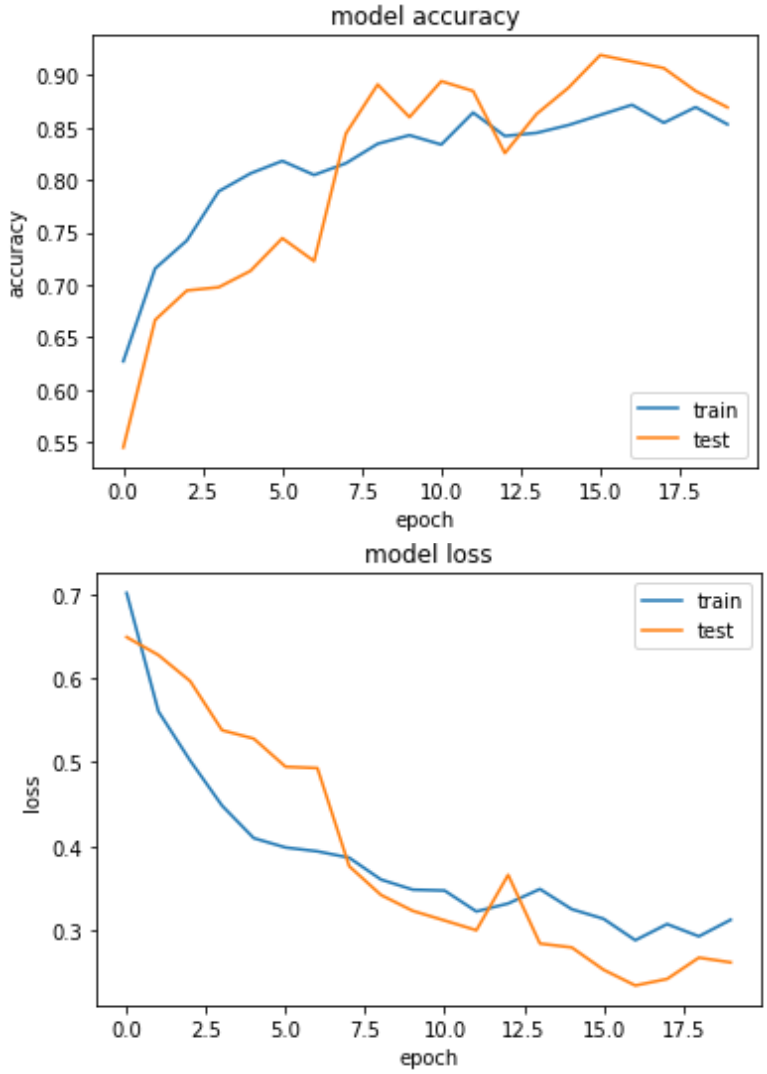
The baseline model described in the previous section was modified through various techniques (like data normalization, regularization etc.) as mentioned below:

1. CNN-1 model

   The CNN-1 model was the first major modification made to the base model through the following changes:

   1. Decreasing the Steps/Epoch to 42
   2. Increasing the number of channels to 3 using the method described in the Data pre-processing section
   3. Implementing Data Augmentation using ImageDataGenerator method
   4. Used the Annealer learning rate scheduler
   5. Increasing the number of CNN Layers to 4

   Consequently, the performance of the CNN-1 was better than the base model as the models Accuracy increased and the Loss dropped

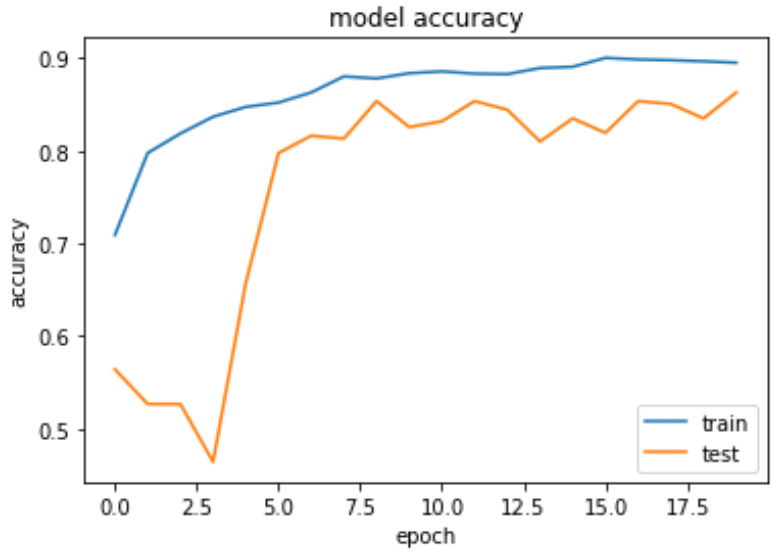


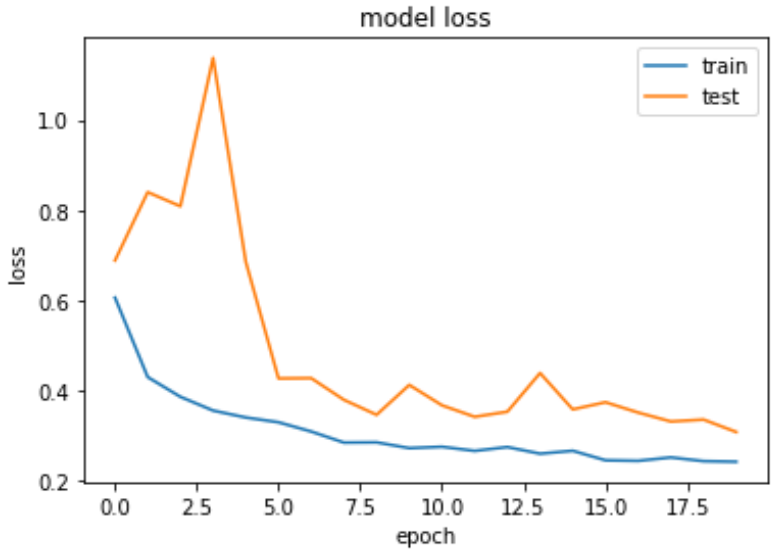
2. CNN-2 model

The CNN-2 model was generated by making these modifications to the CNN-1 model:

1. Applying Batch Normalization method to all the CNN Layers
2. Applying 20 to 30 % dropout in all layers of CNN and Fully connected MLP

The performance of the CNN-2 dropped a bit in comparison to CNN-1



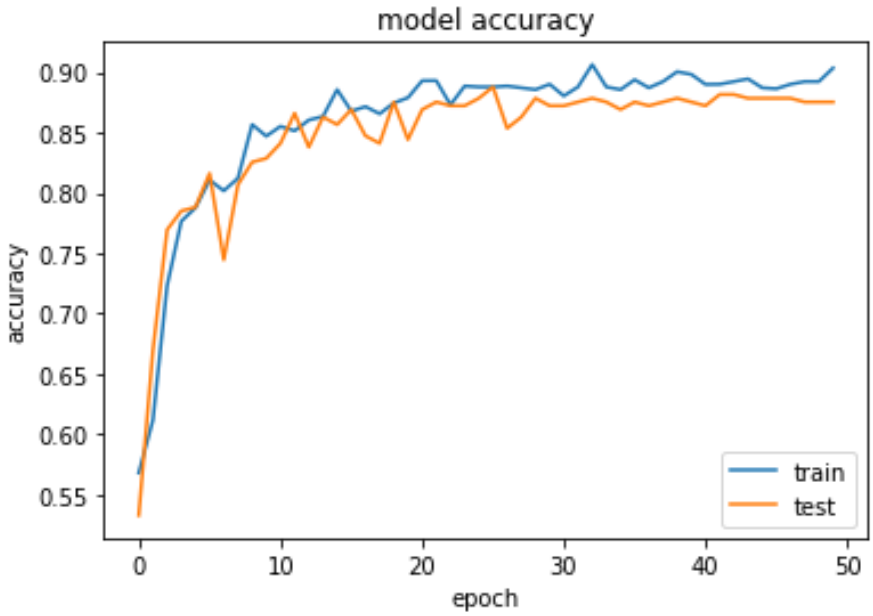


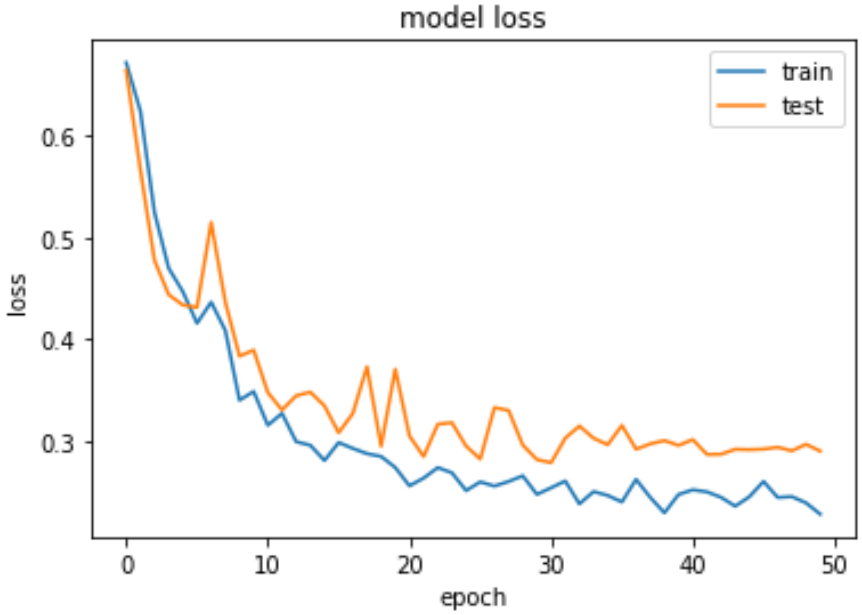
3. CNN-3 model

The CNN-3 model was generated by making these modifications to the CNN-2 model:

1. Increasing the number of Epochs to 50
2. Removing Batch Normalization method from all the CNN Layers

Increasing the Epochs improved the performance of the CNN model



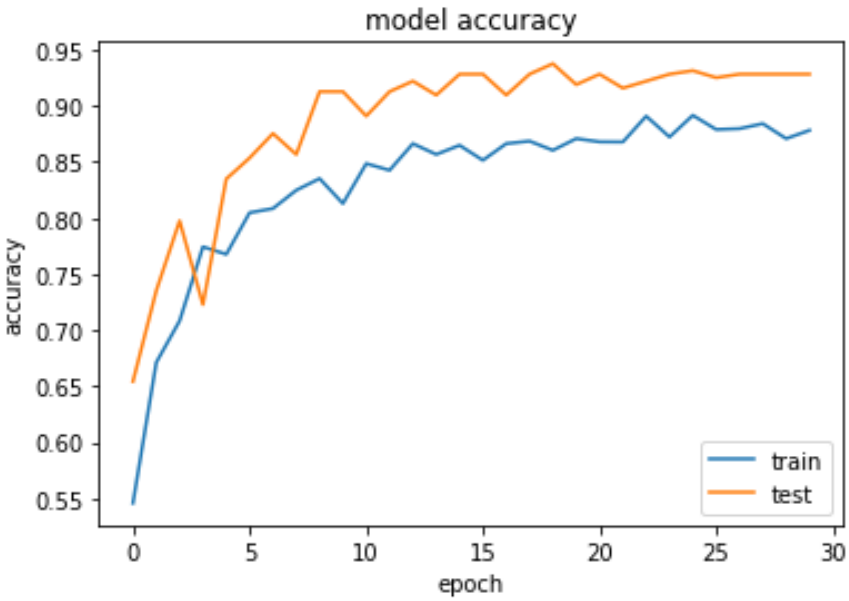


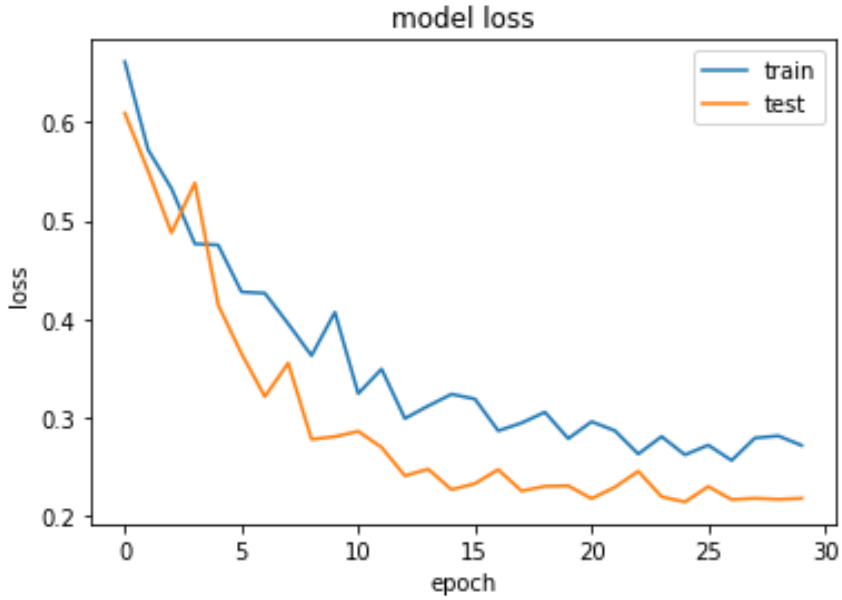
4. CNN-4 model

The CNN-4 model was generated by making these modifications to the CNN-3 model:

1. Decreasing the number of Epochs to 30

Surprisingly, 30 Epochs gave a much better performance for the CNN model as the accuracy and Loss improved by a good margin



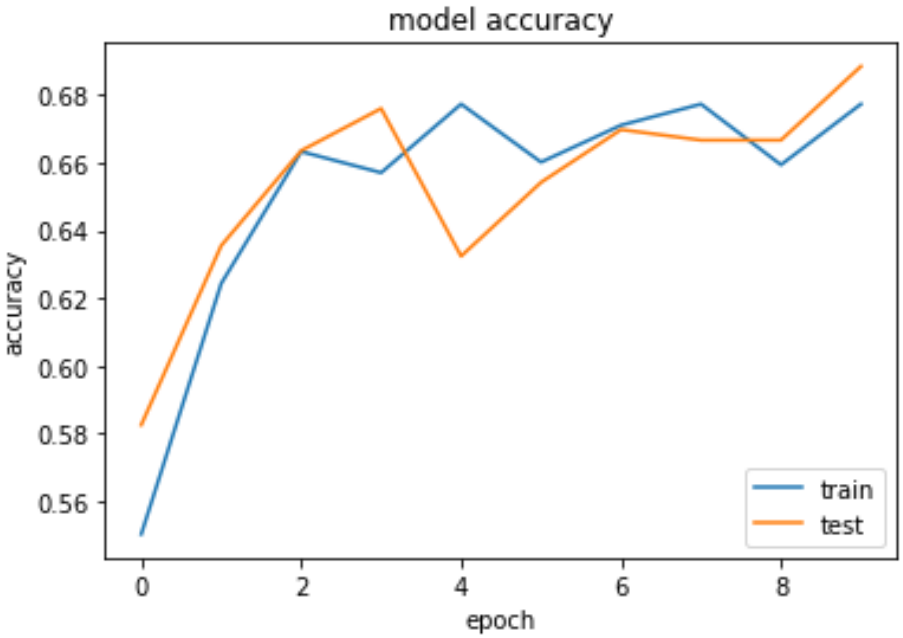


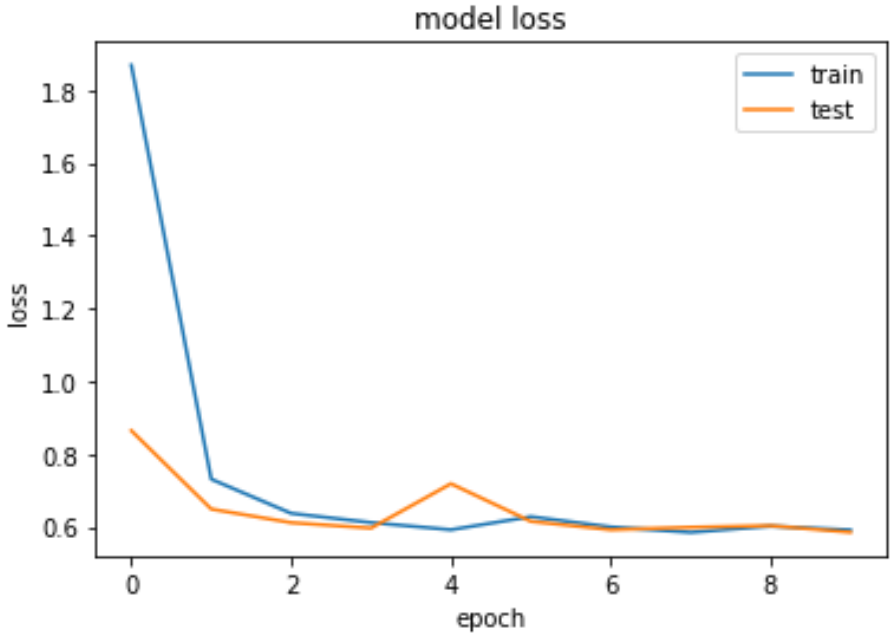
5. CNN-5 model

The CNN-5 model was generated by making these modifications to the CNN-4 model:

1. Decreasing the number of Epochs to 10
2. Implementing *L-2* Regularization for all the layers with alpha=0.01

The CNN model gave quite a poor performance with L-2 Regularization



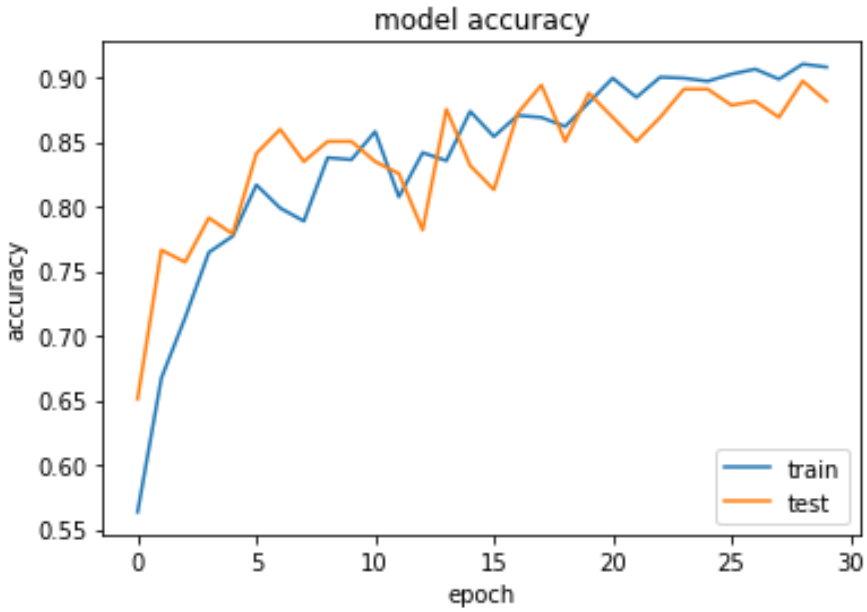


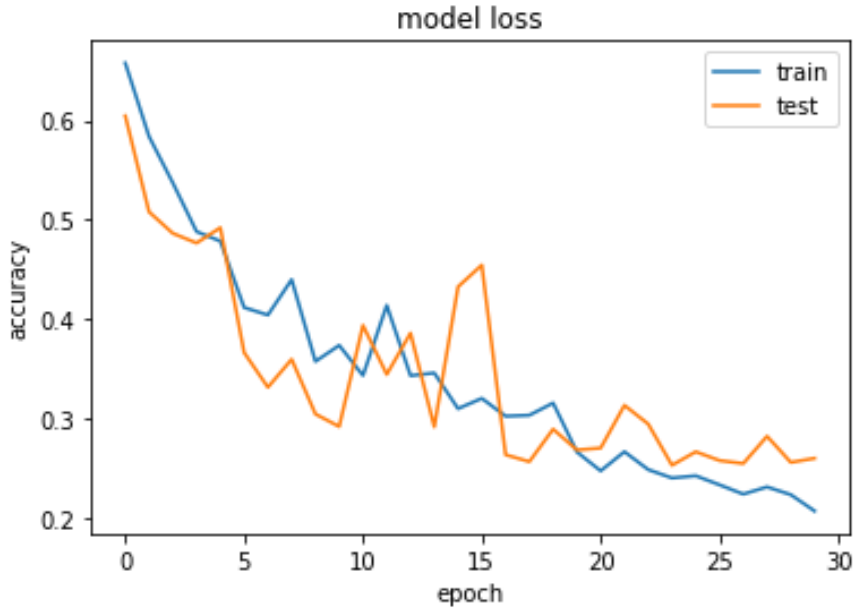
6. CNN-6 model

The CNN-6 model was generated by making these modifications to the CNN-5 model:

1. Increasing the number of Epochs to 30
2. Removing L-2 Regularization
3. Implementing the *ReduceLROnPlateau* method which reduces the learning rate when a metric has stopped improving

Consequently, the performance of the CNN-6 model improved through a huge margin as compared to CNN-5



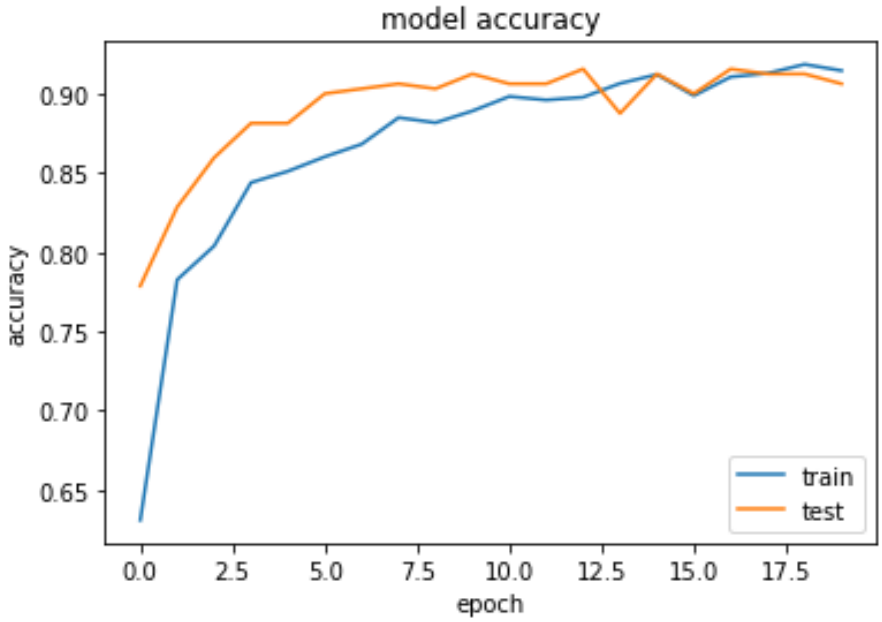


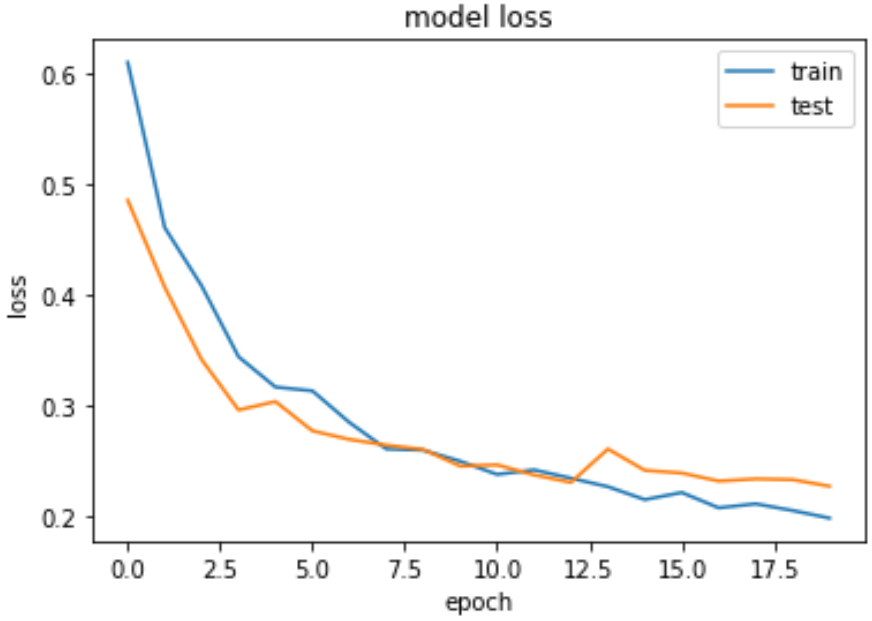
7. CNN-7 model

The CNN-7 model was generated by making these modifications to the CNN-6 model:

1. Decreasing the number of Epochs to 20
2. Increasing the Steps/Epoch parameter to 128
3. Changing the Callback to Annealer Learning rate back from *ReduceLROnPlateau*

The CNN-7 model gave a comparatively better performance with the increased number of steps/epoch and decreased Epochs





A comparative analysis of all the 7 Models, based on the hyperparameters used, is provided in the table below

Model	Batch Size	Epoch	Steps/Epoch	Channels	Augmentation	Callbacks	CNN Layers	MLP hidden layers	L2 Regularization
Base CNN model	32	20	1283	2	No	No	3	2	No
CNN-1 model	32	20	42	3	Yes	Yes	4	2	No
CNN-2 model	32	20	42	3	Yes	Yes	4	2	No
CNN-3 model	32	50	42	3	Yes	Yes	4	2	No
CNN-4 model	32	30	42	3	Yes	Yes	4	2	No
CNN-5 model	32	10	42	3	Yes	Yes	4	2	Yes
CNN-6 model	32	30	42	3	Yes	Yes	4	2	No
CNN-7 model	32	20	128	3	Yes	Yes	4	2	No

Discussions & Conclusions

This project is used to demonstrate the use of a convolutional neural network in tensorflow on the Statoil/C-CORE Iceberg Classifier Challenge dataset found on Kaggles' website

(https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/data).

The purpose of this is to demonstrate the use of tensorflow/keras to create a convolutional neural net that classifies an iceberg from a boat.

Based on the results of all the 7 CNN models, it can be concluded that CNN-4 gave the best performance out of all as shown in the table below:

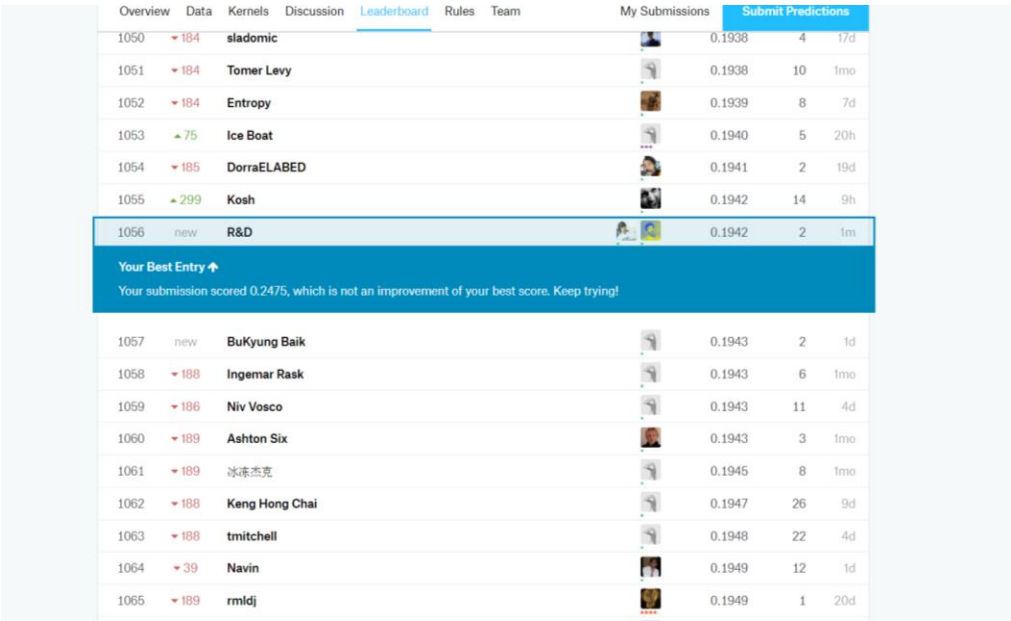
MODEL	LOSS	ACCURACY
Base CNN model	0.3473	0.8100
CNN-1 model	0.2614	0.8692
CNN-2 model	0.3086	0.8629
CNN-3 model	0.2910	0.8754
CNN-4 model	0.2179	0.9283
CNN-5 model	0.5855	0.6885
CNN-6 model	0.2596	0.8816
CNN-7 model	0.2264	0.9065

Future Scope

1. From my limited observations, this ConvNet was able to get a *Log Loss* score of around 0.194~0.205+ and has the potential to go deep by further regularization and optimizing such as to extend the training epochs, adjust early stopping, tweak optimizer, etc
2. We would also try refining the model through fine tuning it, using L-2 Regularization.
3. We would also like to figure out the use/effect of the *inc_angle* feature on the dataset and use it to better refine our CNN model
4. Finally, I would like to thank *shivam207* for his solution implemented for the Iceberg Classifier Challenge (https://github.com/shivam207/iceberg_challenge). I learnt a lot for how to reshape and normalize the dataset features from his solution [2]

Kaggle Performance

Using the submission files generated from the **CNN-4** and **CNN-7** model we were able to achieve a rank of **1056** in the [competition]



References

[1] "Statoil/C-CORE Iceberg Classifier Challenge | Kaggle." *Kaggle.com*. N. p., 2017. Web. 16 Dec. 2017.

[2] "Shivam207/Iceberg_Challenge." GitHub. N. p., 2017. Web. 17 Dec. 2017.