EE-258
PROJECT II

REPORT BY
SAGAR SHAH 011426818
PROJECT BY
SAGAR SHAH 011426818
DHARMIK SHAH 011413103

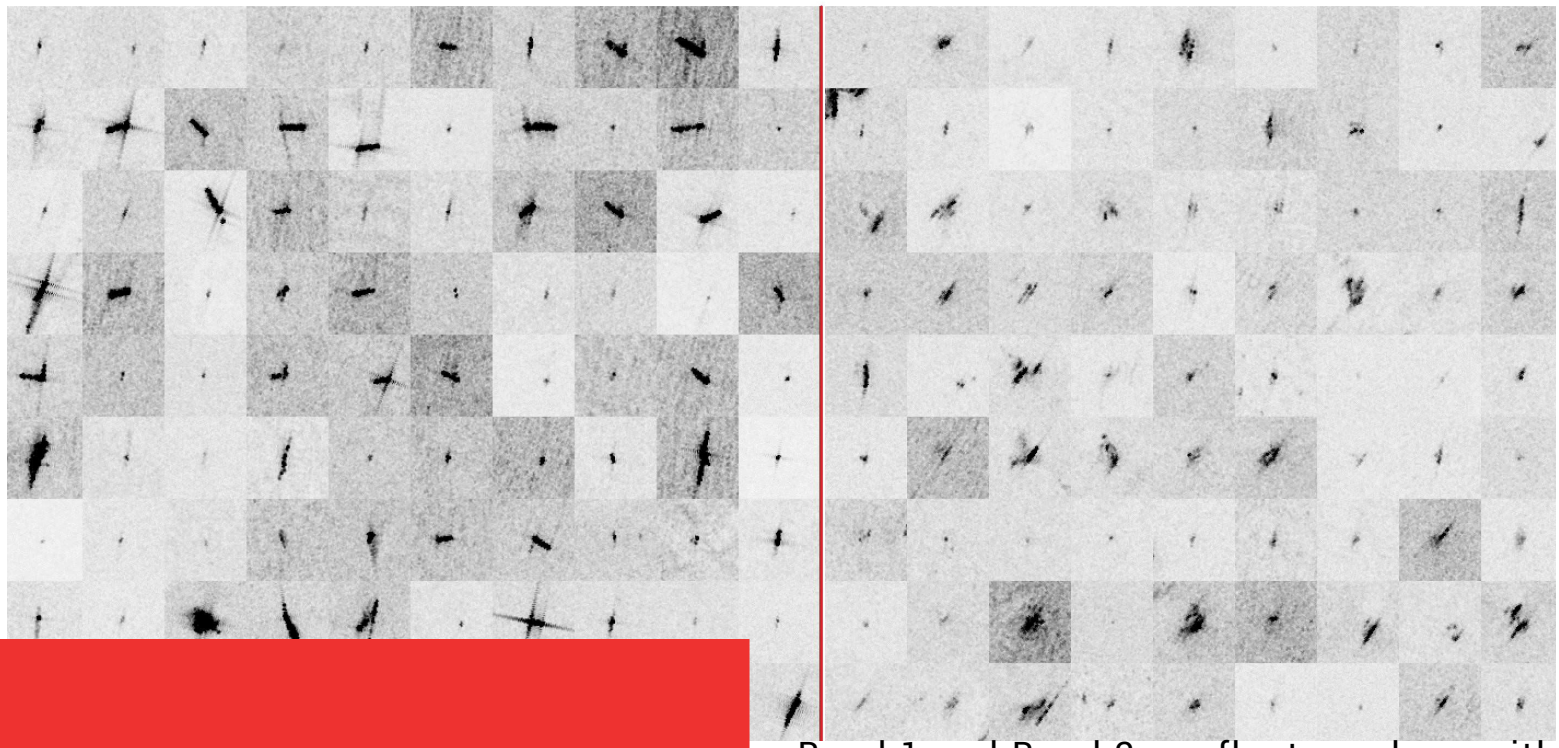
PROJECT GUIDE
DR. BIRSEN SIRKECI

kaggle Competitions

WHAT WE ARE DOING HERE

- We all know from TITANIC movie that drifting icebergs can cause threats to navigation for ships in areas like the Pacific Ocean.
- Many have researched in this field and most of them use aerial reconnaissance and shore-based support to monitor conditions and risks from icebergs.
- However, due to harsh weather conditions in remote areas, these methods are not very feasible and the only option is of a satellite.
- C-CORE is using satellite data for over 30 years and has built a computer vision based surveillance system.
- To keep operations safe and efficient, Statoil is interested in getting a unique perspective on the use of machine learning to detect more accurately and discriminate against threatening icebergs as early as possible.

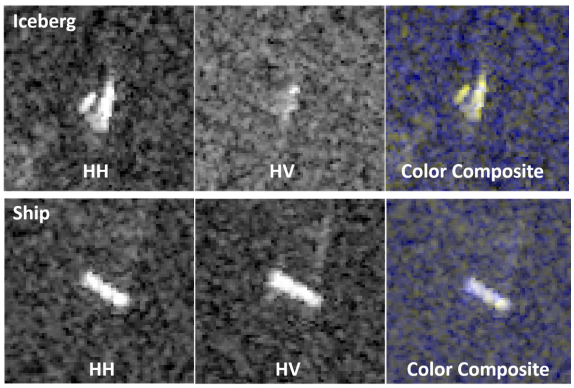
- Kaggle is a platform for predictive modeling and analytics competitions in which statisticians and data miners compete to produce the best models for predicting and describing the datasets uploaded by companies and users.
- In this competition, we're challenged to build an algorithm that can tell us if a remotely sensed target is a ship or iceberg.
- We are using remote sensing systems housed on satellites over 600 kilometers above the Earth to detect icebergs.
- The C-Band radar operates at a frequency that "sees" through darkness, rain, cloud and even fog and 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.
- We are using Sentinel-1 satellite which is a side-looking radar, which means it sees the image area at an angle (**incidence angle**).
- The Sentinel-1, can transmit and receive in the horizontal and vertical plane. Using this, you can get what is called a dual-polarization image.



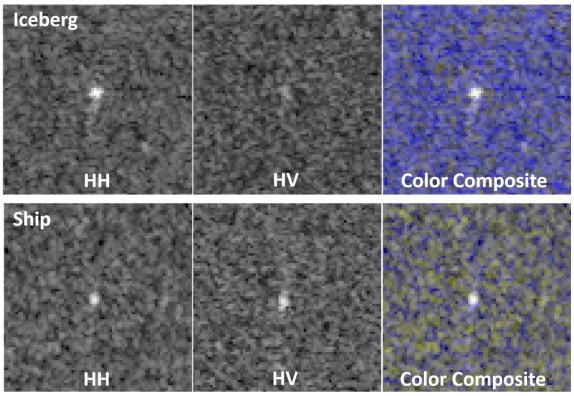
LET'S TALK ABOUT DATA

- Here, we will work to predict whether an image is a ship or an iceberg.
- All the images are 75x75 in size with two given bands.
- The data (train.json, test.json) is presented in json format. The files consist of a list of images.
- **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. 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.**

- Band 1 and Band 2 are float numbers with unit being dB. These 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).



- HH and HV channels can play an important role in the object characteristics.Easy classification examples are seen above. These objects can be visually classified.
- Here we see challenging objects to classify.



WHAT'S INSIDE THE DATA?

- If we see and observe train and test data, there are few things that we can tell from it.
- The train data contains 1604 rows and 5 columns.
- The test data contains 8424 rows and 4 columns.
- We have to add data augmentation for better results and for comparison we are putting 25% of train data as validation.

	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...	e2538 8fd	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
5	[-20.769371, -20.769434, -25.906025, -25.90602...	[-29.288746, -29.712593, -28.884804, -28.88480...	b51d18b5	36.9034	1
6	[-26.673811, -23.666162, -27.622442, -28.31768...	[-24.557735, -26.97868, -27.622442, -29.073456...	31da1a04	34.4751	1
7	[-24.989119, -27.755224, -25.817074, -24.98927...	[-27.755173, -26.732174, -28.124943, -31.83772...	56929c16	41.1769	0
8	[-17.146641, -17.146572, -17.994583, -19.44553...	[-25.733608, -24.472507, -24.710424, -22.77215...	525ab75c	35.7829	0
9	[-24.020853, -23.551275, -27.18819, -29.126434...	[-28.702518, -33.563324, -29.571918, -29.12643...	192f56eb	43.3007	0
10	[-21.397552, -19.753859, -23.426783, -24.65221...	[-26.72291, -27.418192, -27.787899, -25.774536...	3aac67cd	44.624	1
1593	[-28.554342, -26.37149, -25.17157, -25.171608,...	[-26.055567, -27.394543, -28.55442, -30.389864...	d86deb2b	na	0
1594	[-25.950792, -26.255636, -23.834808, -25.09588...	[-30.093304, -28.75441, -27.240044, -29.178288...	cdee905a	na	0
1595	[-25.31617, -25.895485, -24.773212, -23.779497...	[-26.516129, -29.122847, -29.568417, -30.53455...	2539742b	na	0
1596	[-14.463013, -15.378237, -14.535848, -14.83290...	[-25.165276, -24.682676, -25.41721, -30.026262...	2ea3c9f1	na	0
1597	[-19.997332, -18.115284, -15.673027, -14.26133...	[-20.727669, -23.175728, -20.576469, -20.28193...	9cadda28	na	0
1598	[-14.200315, -15.292111, -15.60808, -18.324097...	[-24.02071, -24.455181, -24.020878, -23.607269...	8376a077	na	0
1599	[-30.999878, -29.976866, -28.233906, -29.50732...	[-27.847719, -28.233864, -24.712077999999998, ...	04e11240	na	0
1600	[-25.31155, -26.511555, -28.694487, -27.180115...	[-29.563713, -28.290375, -26.839405, -28.29046...	c7d6f6f8	na	0
1601	[-18.141895, -18.141844, -19.01737, -19.701599...	[-25.305355, -29.387701, -28.963863, -26.16023...	bba1a0f1	na	0
1602	[-22.455633, -25.794661, -26.954567, -22.83354...	[-26.070356, -22.093737, -21.577662, -24.53376...	7f66bb44	na	0
1603	[-19.909191, -20.678406, -20.208834, -18.42441...	[-24.44487, -24.956001, -27.722103, -26.078417...	9d8f326c	na	0

1604 rows × 5 columns

- The test data don't have the 5th column of is_iceberg and it's only for the training data.
- The training data have many values of incidence angle with 'na' in it. We have to change it to '0' in the dataset.
- If we compare train and test data, test data is of 1.41GB and train data is just of 187MB and it's very small in size than test data.

	band_1	band_2	id	inc_angle
0	[-15.863251, -15.201077, -17.887735, -19.17248...	[-21.629612, -21.142353, -23.908337, -28.34524...	5941774d	34.966400
1	[-26.058969497680664, -26.058969497680664, -26...	[-25.754207611083984, -25.754207611083984, -25...	4023181e	32.615072
2	[-14.14109992980957, -15.064241409301758, -17....	[-14.74563980102539, -14.590410232543945, -14....	b20200e4	37.505433
3	[-12.167478, -13.706167, -16.54837, -13.572674...	[-24.32222, -26.375538, -24.096739, -23.8769, ...	e7f018bb	34.473900
4	[-23.37459373474121, -26.02718162536621, -28.1...	[-25.72234344482422, -27.011577606201172, -23....	4371c8c3	43.918874
5	[-18.13534927368164, -18.13534927368164, -18.1...	[-14.650325775146484, -14.650325775146484, -14...	a8d9b1fd	35.933020
6	[-24.565174, -26.567524, -25.946882, -26.89542...	[-32.915886, -35.194798, -32.272282, -28.7505,...	29e7727e	42.414200
7	[-20.162964, -18.16898, -17.023144, -17.023216...	[-30.905346, -28.406647, -25.381365, -28.77645...	92a51ffb	33.638300
8	[-21.920645, -21.737827, -21.383303, -21.73792...	[-25.53537, -28.514198, -27.75848, -31.280354,...	c769ac97	41.107600
9	[-14.811565, -16.318594, -17.082613, -16.41037...	[-22.617977, -21.065811, -27.914198, -25.24334...	aee0547d	34.966400
10	[-17.691271, -16.036381, -17.908293, -22.23594...	[-26.91781, -24.979542, -24.979473, -24.734716...	565b28ac	35.782900
11	[-28.337493896484375, -29.6109619140625, -30.5...	[-28.33753776550293, -30.080631256103516, -32....	e04e9775	35.566871
12	[-16.983395, -13.952508, -14.653442, -17.17666...	[-21.200462, -22.539478, -23.908703, -20.45786...	8e8161d1	33.185400
13	[-23.389873504638672, -26.042346954345703, -26...	[-24.104223251342773, -26.04238510131836, -26....	4cf4d256	40.111298
8408	[-13.77760124206543, -13.77760124206543, -13.7...	[-11.978379249572754, -11.978379249572754, -11...	c2834388	43.679004
8409	[-22.443909, -24.335094, -22.830139, -21.37918...	[-23.234081, -24.335094, -25.871956, -22.8302,...	146143c3	37.259800
8410	[-13.979014, -13.694148, -15.390521, -17.50066...	[-22.120684, -25.735348, -27.9583, -26.935192,...	d59aee00	38.153500
8411	[-24.32549476623535, -24.32549476623535, -21.4...	[-23.410398483276367, -23.410398483276367, -21...	cbc0b93b	46.326263
8412	[-23.779713, -22.732008, -22.345963, -21.44977...	[-32.678444, -30.992085, -25.773886, -26.05875...	088e2ff7	38.020000
8413	[-24.679191589355547, -24.679191589355547, -24.6...	[-24.419933319091797, -24.419933319091797, -24...	673d33cd	40.644460
8414	[-28.106094360351562, -28.106094360351562, -28...	[-27.04123306274414, -27.04123306274414, -27.0...	674b031e	39.234395
8415	[-20.00988006591797, -20.00988006591797, -20.1...	[-19.051467895507812, -19.051467895507812, -20...	43db4207	43.107674
8416	[-22.188007, -19.265495, -21.810396, -20.93233...	[-26.004297, -26.309145, -25.425119, -26.62512...	156855e1	41.858200
8417	[-16.408174514770508, -16.408174514770508, -16...	[-16.71302032470703, -16.71302032470703, -16.7...	ac96cfb0	45.963624
8418	[-23.416317, -25.469601, -28.567678, -27.40787...	[-24.908989, -24.641747, -28.163612, -32.73323...	fe45aef5	45.285300
8419	[-25.082357, -26.71583, -24.599827, -25.082571...	[-25.860718, -23.29442, -25.860861, -25.334354...	16ee9b50	34.795500
8420	[-21.031391143798828, -21.031391143798828, -21...	[-23.755836486816406, -23.755836486816406, -23...	5a599eb7	32.246683
8421	[-28.609278, -26.514626, -26.514679, -26.83061...	[-28.609278, -29.437183, -30.35239, -31.375494...	df30d6dd	39.503200
8422	[-27.068821, -27.068892, -23.970854, -22.38730...	[-29.991381, -29.163599, -24.886002, -27.71266...	18af95b1	33.638000
8423	[-25.438865661621094, -25.438865661621094, -25...	[-23.85527801513672, -23.85527801513672, -23.8...	27d788c8	36.758181

8424 rows × 4 columns

PRE-PROCESSING ON DATA

- I already mentioned earlier that we have to augment our data due to the large difference between the size of training and test data.
- Before augmenting our data we have to add one more band for better polarization.
- Polarization is a very important factor. By combining two channels we can have a better prediction of a ship or an iceberg.
- Here, we are first reshaping images to 75 x 75 in band1 and band2. After reshaping we are adding them for band3.
- We are doing addition because log(band1 x band2) = log(band1) + log(band2).
- The reshaping and rescaling of the images are done for equal and better processing inside the model.
- The training data have many unknown values of incidence angle with 'na' in it and we are changing it to '0'. After changing it we are finding the indices.
- Indices give us the ability to use either truncated data (take > 0) or the whole data.
- After this much of pre-processing, when we run our baseline model (which I will explain in next page), we were not getting the expected result.
- The training data is too small and we need data augmentation to increase the number of images.
- We are using keras and data augmentation can be done using OpenCV in it.
- We can easily download the OpenCV package from the official website and use it just by **import cv2**.
- I was having a trouble in running OpenCV on Jupyter notebook and I fixed it by updating Microsoft visual c++ packages on my computer.
- What we are doing is flipping horizontal and vertical data, after flipping we are adding them to the main training data.
- Other than flipping, there are other methods like rotation, translation, shearing, and stretching can be used.
- The selection of augmentation method depends on the data that we have.
- OpenCV is very helpful whenever there are images involved in any operation.

WHAT WE GOT AFTER PRE-PROCESSING

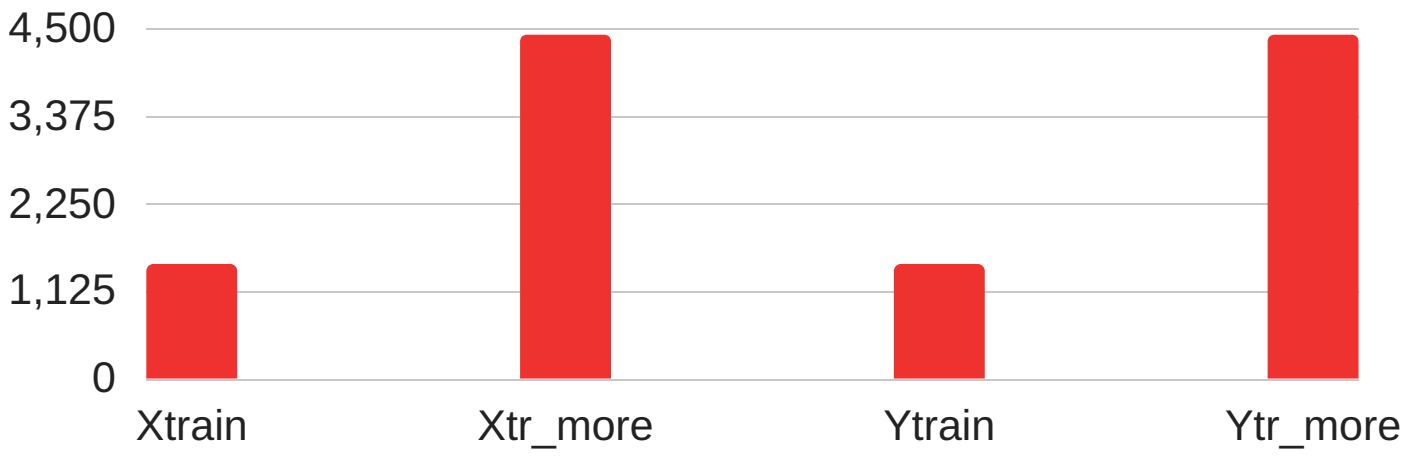
- As you can see in image and graph previously I had 1471 Xtrain and Ytrain images which increase after augmentation and becomes 4413 each.
- I have given two different names for original and extend data so that I get two options.
- An increased number have given the stability to my main model and I can be sure of my training and validation results.
- Data Augmentation is a very powerful tool and it creates more stable training and validation data.

```
In [22]: 1 Xtr_more.shape
Out[22]: (4413, 75, 75, 3)

In [23]: 1 Xtrain.shape
Out[23]: (1471, 75, 75, 3)

In [20]: 1 Ytr_more.shape
Out[20]: (4413,)

In [24]: 1 Ytrain.shape
Out[24]: (1471,)
```



THOUGHT OF BASELINE MODEL AND IT'S IMPLEMENTATION

- This is the binary image classification problem and hence Convolutional Neural Network was the starting point for our baseline model.
- Images are a form of metrics and hence Tensorflow is the best option available for the implementation of CNN.
- We can use Tensorflow in two ways. One is a use of core Tensorflow and Second is a use of Keras API on Tensorflow.
- We are using Keras API because it's an official API for Tensorflow by Google and it gives results much faster due to inbuilt use of GPU.
- For the baseline model, we are not using the augmented data because we came to a conclusion of using data augmentation after using baseline model.
- For the baseline model, we have used two layers of CNN, one flattened layer, and two dense layers.
- Two CNN layers are of 16 and 32 respectively. We have used 3x3 kernel and 2x2 of max-pooling in each layer.
- The first dropout before the Flatten layer is of 0.25 and second dropout after first Dense layer is of 0.5.
- We have used 'relu' activation function for two CNN and one Dense layer. For the final Dense layer, we have used 'sigmoid' activation function.
- The final Dense layer is of 48 and we have provided batch normalization after every layer.
- Since it is very basic CNN model, we didn't want to change the size of a layer to take the final decision of selecting the parameter.
- We have used two optimizers for comparison, one is RMSprop and second is Adam.
- For the first run of baseline model, we have used 12 epochs and batch size is 10.
- After running both the parameters on Adam and RMSprop, results are not so exciting.
- For the first run, we got 75% and 79% accuracy respectively for RMSprop and Adam.
- We were not getting good results with RMSprop and it was time consuming so we took a decision of shifting to Adam optimizer permanently.
- For the second run of baseline model, we have used 50 epochs and batch size of 10.
- After increasing epochs to 50 in the second run, we got around 95% of accuracy using Adam optimizer.
- We changed the batch size from 10 to 32 and got almost the same result.
- We also changed kernel size to 2x2 and used a stride of 2, but got almost the same result.
- After these many changes, we had fixed the model and went ahead for more intensive CNN.

75%

Accuracy percentage of the baseline model with RMSProp optimizer.

79%

Accuracy percentage of the baseline model with ADAM optimizer.

95%

Accuracy percentage of the baseline model with ADAM optimizer for 50 epochs.

ROAD AHEAD WITH CNN AND MODEL IMPROVMENTS

- After so many trials and errors with our baseline model, we fixed one structure and went deeper inside the CNN.
- From baseline model, we were not getting the expected results and Log loss score was also not that great so we tried a different approach to the given training data.
- At this point, we were above 1500 in the competition and it was not at all what we wanted ahead.
- We applied Data Augmentation on training data and increased the size of the data for training model.
- In the final baseline model, we had around 100 non-trainable parameters and our aim was to make it zero anyhow.
- Now we were focusing more on the size of CNN and parameters attached to it.
- We started changing parameters one by one and observed the result.
- The road till 95% was easy but from 95% to 100% was very long and we are still trying for 100%.
- Due to the restriction of our system and lack of GPU use our experiments were taking so much of time to execute and produce results.
- Once we produce the results, the next step was to compare it with the previous one and what improvements we have got in this one or what we are missing.
- The performance is increased or decreased at the same time how much time and resources this model takes, we were observing all of this one by one.
- CNN have many ready models from ImageNet competition like VGG16, VGG19, Xception, ResNet50, InceptionV3, InceptionResNetV2 and MobileNet. We have used some of them after creating our own model at the start.
- After Baseline model we created CNN with a minimum of four layers before we flatten them and two Dense layers before the final output produced.
- We started with a combination of 16-32-32-16 for CNN and Dense layers were of 128-64-1 respectively.
- Apart from the final Dense layer which has a sigmoid function, all other layers have 'relu' as the activation function.
- The kernel and Maxpool size are of 3x3 with the stride of 2x2 for all the Convolution layers.
- We have used the dropout of 0.2 after every layer before the final layer.
- As mentioned earlier, we are using Adam optimizer from here onwards for every model we design. The learning rate for Adam was set to 0.001 and with the decay of 0.0.
- We got 96% accuracy and 0.3982 public score.

96%

Accuracy percentage of the Updated CNN model.

0.3982

Kaggle public score after this CNN model.

1024

Kaggle rank after running this CNN model.

1. THIS WAS OUR SITUATION AT THE START OF THE BASELINE MODEL WITH RMSPROP.

2. THE FACE BECAME A LITTLE BIT BETTER WITH BASELINE MODEL AND ADAM OPTIMIZER.

3. THIS IS US AFTER MODEL IMPROVEMENTS AND ACHIEVING 96% ACCURACY.

4. WE GOT HAPPY WHEN GOT 768 RANK WITH 98% ACCURACY. THE FIRST TIME WE WERE BELOW 1000.

WE WILL BE THIS IF WE REACH TO BELOW 50.

MORE MODEL IMPROVMENTS

- After previous results, we tried the same model with RMSprop and ADAgrad optimizer to have a final check about model performance and the result was better with ADAM optimizer only.
- Now with more surety on our model, we tried the model with more higher parameters that can cause a problem to our software system in laptops.
- First, we tried a model with a combination of 32-64-64-32 for CNN and Dense layers were of 256-128-1 respectively.
- Use of activation function was same as earlier.
- The kernel and Maxpool size are of 3x3 with the stride of 2x2 for all the Convolution layers.
- We have used the dropout of 0.2 after every layer before the final layer.
- The whole purpose of just changing the CNN layers with a number of feature maps was to gain more accuracy and reduce the logloss function on training data.
- We got a result of 97% and a public score of 0.2428 with the rank of 982 on Kaggle.
- What was happening behind the model after changing a number of feature maps? We were increasing the number of parameters after each trial and processing was becoming more and more intense.
- One big achievement we got after this implementation was, there were 'zero' non-trainable parameters in our model.
- The execution time was increasing after every change in our CNN and we were felt that we will need a GPU at a later stage of the project.
- The next change was making a combination of 64-128-128-64 for CNN and Dense layers were of 256-128-1 with other parameters remained unchanged.
- The process became too slow and my laptop was not able to produce faster results from this model implementation.
- We got a somewhat same result as the previous model and I was wondering about how my laptop will handle the further expansion of feature maps in CNN model.
- There enter's a wildcard entry from Cisco. One of our friends helped us to get one PC with 64GB of Ram and way more advanced features than what we have in our laptops.
- We created a model with a combination of 256-512-512-256 for CNN and Dense layers were of 512-256-1 with a dropout rate of 0.15 and other parameters remain unchanged.
- BANG!!! look at the results we have got, we got the accuracy of 99% with a public score of 0.1869 and rank of 768 on Kaggle.
- It was a big jump for us and that was made possible just because of the more computational power and a number of parameters.

99%

Accuracy percentage of the Updated CNN model.

0.1869

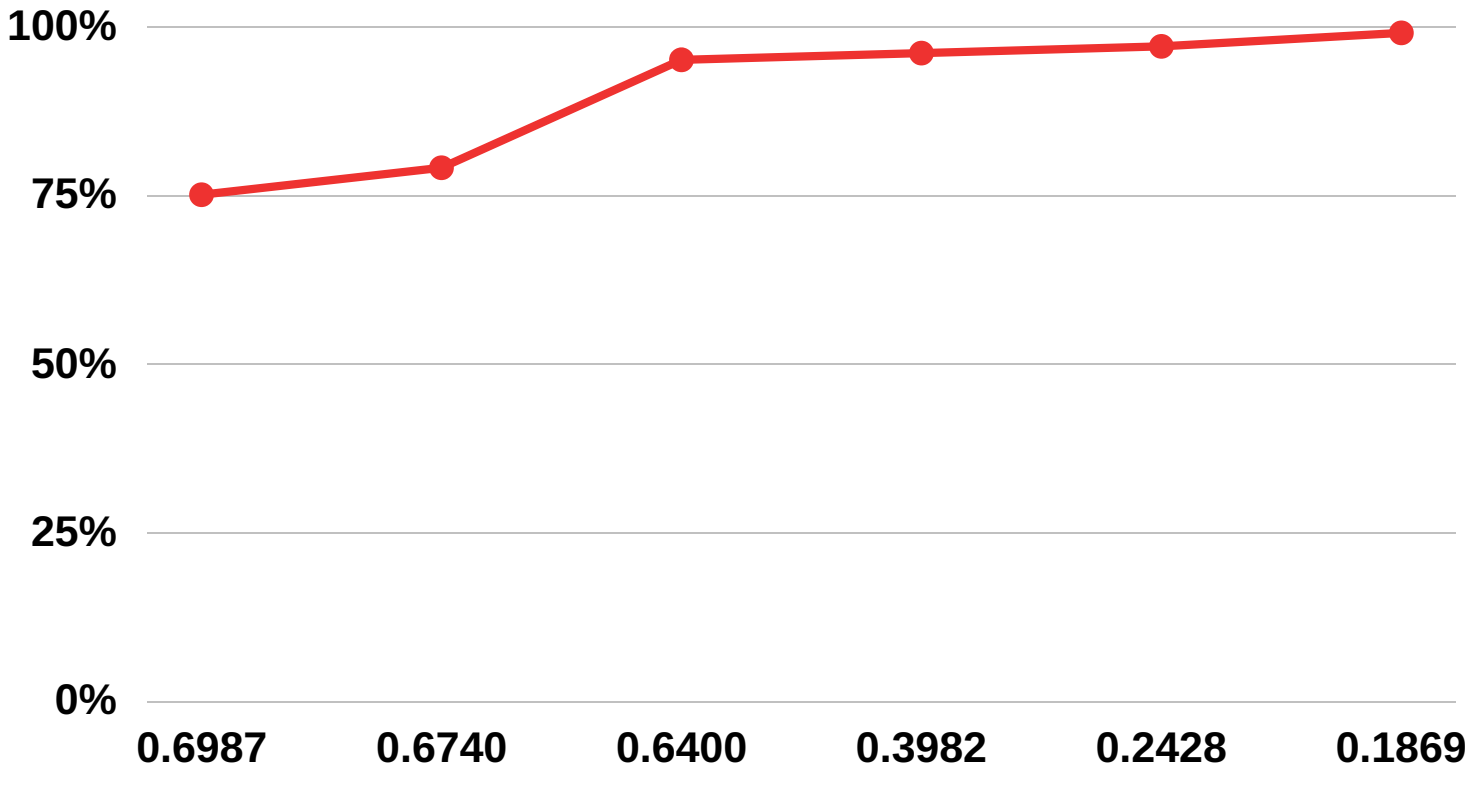
Kaggle public score after this CNN model.

768

Kaggle rank after running this CNN model.

GRAPHICAL MEASURE OF PERFORMANCE IMPROVMENT

- After the last result of 99% accuracy and 768 Kaggle rank, we were at least somewhere from the starting point.
- We increased the size with a combination of 256-512-512-512 for CNN and Dense layers were of 512-256-1 with a dropout rate of 0.15 and other parameters remain unchanged.
- At the start of the baseline model we were observing overfitting due to the small size of the training data, we had introduced data augmentation from the start of the model improvements to avoid overfitting.
- After implementing above model, we got the overfitting and for safe operation, we introduced early stopping in our model at the time of training.
- Early stopping is the best way to avoid overfitting in any model.
- We got the downgraded result from this model and thought of changing parameters came to my mind again after baseline model.
- We changed kernel size from 3x3 to 2x2 and Maxpool size from 2x2 to 3x3 with a stride of 1, all one at a time.
- The result was not good and accuracy downgraded from 99% to 96% with this changed parameters.
- We again went to our last best model of 99%.
- The performance was downgraded due to excessive layers and slow operation.
- Due to a stride of one, there was no reduction of data after each convolution layer and parameters went up in numbers but performance decreased.
- Small kernel size was also not helping due to size and condition of images from the input.
- It was complicating the performance at the backend rather than improving it.
- Below figure gives the graphical representation of the performance improvement in % model by a model which we explained till now.
- The graph is of Public Score vs the Accuracy and from it, you can see that after 99% of accuracy also we were at 768 in the competition.
- We were stuck with this model and not finding an answer for further improvements.
- We thought of data augmentation again after this stage and created training data with flip and rotation.



TIME-LINE OF CNN LAYERS TILL NOW

- Here is the Time-Line representation of changes in CNN layers sizes from start to till now.
- As you can see from this Time-Line, we have changed the size of the layers and came this far.
- With the change of every size, we have changed the parameters and hyperparameters also.
- As I told, we have changed the training data again and added the flipped as well as rotation to the data, we have run the final model on it and the result is downgraded.
- We tried and changed the parameters like kernel size, pool size, and stride as well as learning rate but the 99% accuracy result was nowhere near with all these trial and error methods.
- With rotation, we got the power of more data but that was of no use.

16-32

BASELINE MODEL

16-32-32-16

MODIFIED MODEL

64-128-128-64

96% ACCURACY MODEL

128-256-256-128

97% ACCURACY MODEL

256-512-512-256

99% ACCURACY MODEL

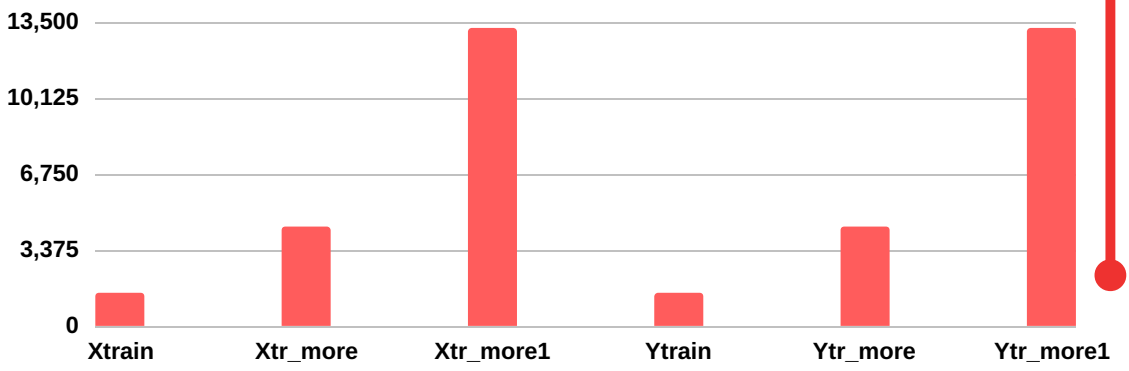
```
In [94]: Ytrain.shape
Out[94]: (1471,)

In [57]: Xtr_more1.shape
Out[57]: (4413, 75, 75, 3)

In [45]: Xtr_more.shape
Out[45]: (13239, 75, 75, 3)

In [62]: Ytr_more.shape
Out[62]: (13239,)
```

Extended Training Parameter



FINAL WRAP AND DECIDED TO CHANGE THE DESIGN COMPLETELY

- As time was near to submit the report on time as well as the shortage of GPU on final days, we were not able to perform further than 99% accuracy with 0.1869 public score and 768 Kaggle rank with our current model.
- This is the limitation of this model and it cant go beyond these results for this given data.
- As you taught in class regarding how to select the performance metrics and if the results are stuck somewhere then look for a solution and over here the final solution is to change the model.
- Till now we were using our own designed models for this project but as I said there are pre-added models of VGG16, VGG19, ResNet50 etc in the Keras, so we shifted from our model to VGG model in the search of improved score results.
- We tried the InceptionV3 model from Keras, which is the winner of the ImageNet competition, it has the depth of 156 layers and it's a type of a GoogleNet only.
- To implement the InceptionV3 model, there are few pre-requirements that one has to follow to run the model.
- These requirements are, change the image size from 75x75 to 224x224, include more intense data augmentation.
- We have given the freedom of data augmentation by defining six classes of different data augmentation techniques in one function only by using a random integer.
- We have used a rotation of 90, rotation of 180, flip horizontally, flip vertically, rotate 90 counterclockwise and rotate 180 with flip horizontally for six types of data augmentation in this new model.
- We went on Kaggle and saw many examples of simple easy codes and mix and match many different methods in one code only.
- After doing this improvement we finally got the success on our path.
- We got the result of 99% accuracy with a public score of 0.1780 and Kaggle rank of 712.
- We are using KNN to train training data, it is time-consuming but has a surety of fine-tuned result.
- We are using K=3 and running it again for the best K on training data for the final time.
- The whole purpose of designing our own model instead of taking one from the Keras is to learn the deeper knowledge of CNN and what the change in hyperparameter can do to the outcoming results.
- After using our own model we are still ahead of everyone from our project pool group, that shows how much you can learn and gain knowledge from your own design and implementations.

99%

Accuracy percentage of the final InceptionV3 model.

0.1780

Kaggle public score after final InceptionV3 model.

712

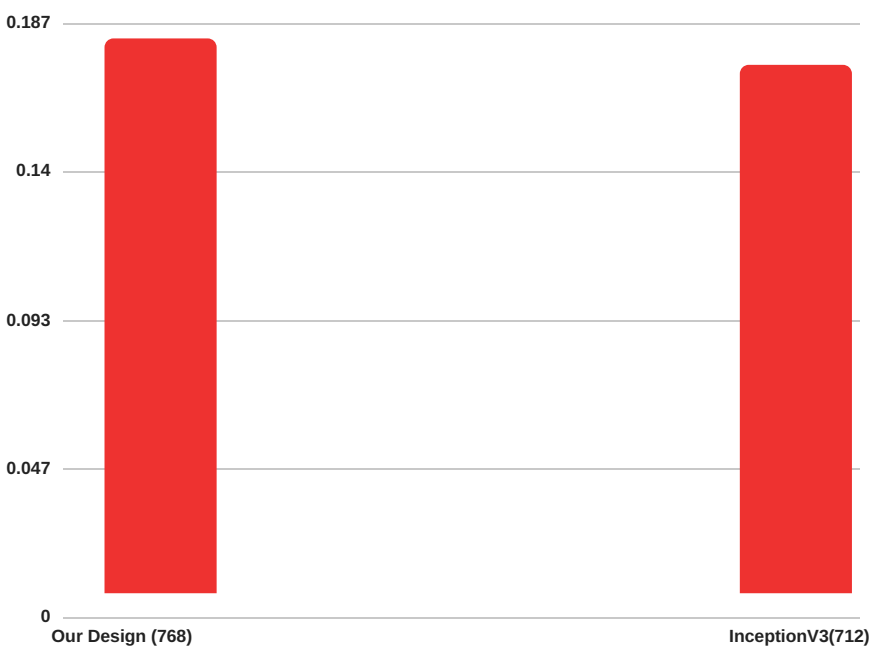
Kaggle rank after running final InceptionV3 model.

Conclusión

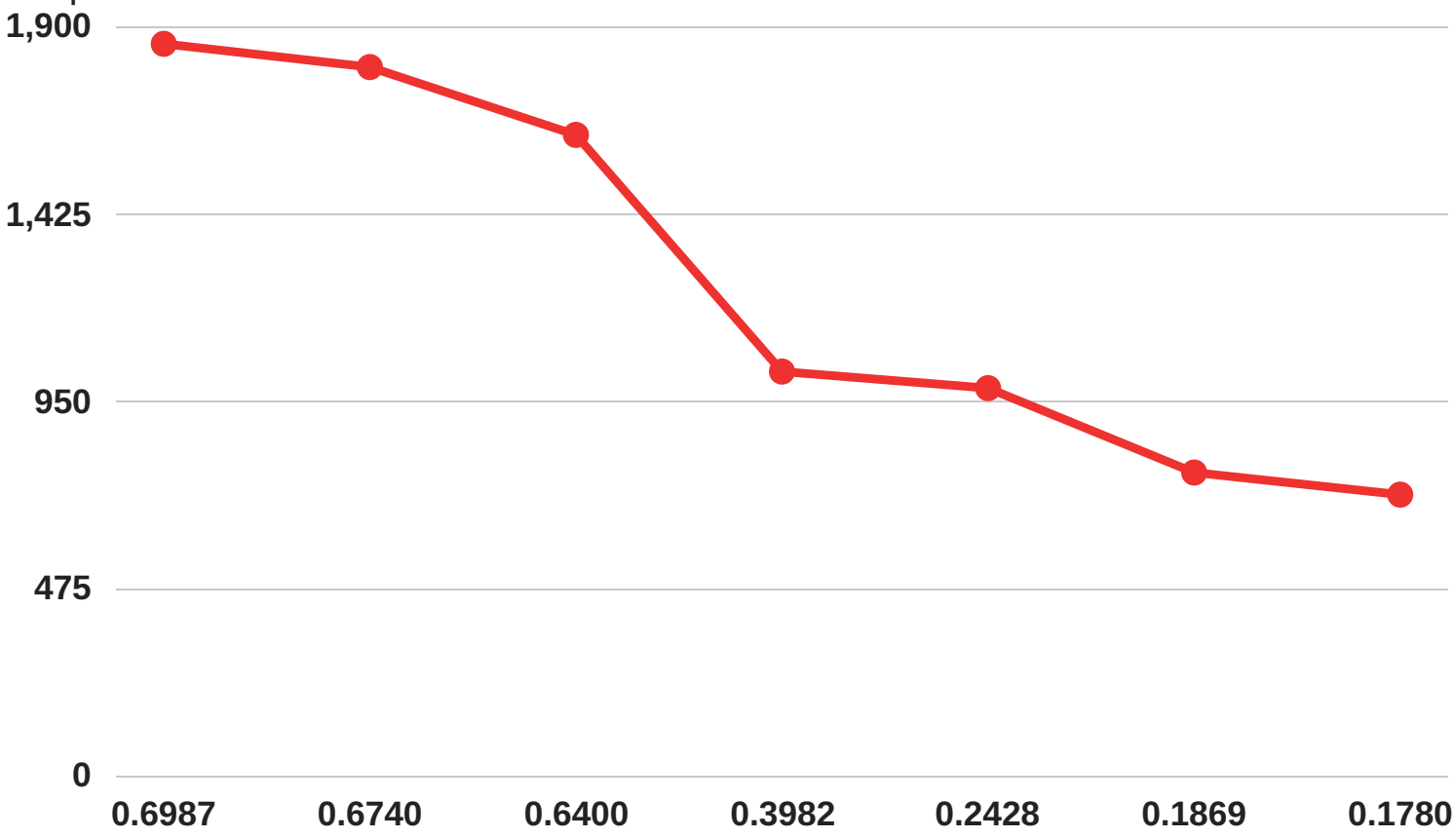
DISCUSSIONS AND CONCLUSION

- The journey was long and pond of knowledge is filled with the practical implementation from scratch.
- I will say we have two results, one with our own design model and one with using InceptionV3 from Keras.
- Our own design gave us the accuracy of 99% and a public score of 0.1869 whereas use of InceptionV3 gave us the accuracy of 99% and a public score of 0.1780.
- We have a submission of around 13 .csv files on Kaggle competition from start to till now.
- The competition on Kaggle is so intense that after submitting the best results that one have, within few minutes that result will be outdated.
- If we see the top 20 people in our competition and go through their profiles, we see that they all are from some big machine learning companies.
- They have the available resources to make more deep models and designs to implement their imagination.
- You can imagine from the fact that, the number one from our competition is working for Airbnb as a Machine Learning Scientist.
- This was the limitation of knowledge as well as resources on our end. If we talk about technology that we have used to generate the result is in very basic capabilities and we could have performed much better if we had more powerful GPU and compatible processor to work on.
- We faced the challenge of time consumption for running each model on our laptop, we also faced the challenge of getting GPU slots for our implementations.
- We wanted to develope our own design of InceptionV3 and wanted to run it on powerful GPU, but due to time and resources constrains we were not able to perform it.
- We also wanted to create functions for F1 score, precesion and recall on our own as Keras dont have these metrices in built on it but due to time constrain we were not able to design it.
- We have designed the model on our own because we wanted to learn the basics of everything from our model and there is no use of just calling the model and getting good result.

GRAPHICAL INSIGHT OF KAGGLE RANK AND PUBLIC SCORE



- The final conclusion of the public score, Kaggle rank, and accuracy is given here on this page.
- The comparison of our design and InceptionV3 is also here.



99%

Accuracy percentage of the final InceptionV3 model.

0.1780

Kaggle public score after final InceptionV3 model.

712

Kaggle rank after running final InceptionV3 model.

REFERENCES

- Book "Deep Learning" by Ian Goodfellow, Yoshua Bengio and Aaron Courville
- https://www.tensorflow.org/
- https://keras.io/
- https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/kernels
- Tensorflow and Keras tutorials from - https://github.com/Hvass-Labs/TensorFlow-Tutorials
- All the heading Images are from Google Search.
- Dr. Birsen Sirkeci's class notes.