Machine Learning Engineer Nanodegree - Capstone Project

James Coffey
January 4, 2018

I. Definition

Project Overview

This project came from the Statoil/C-CORE Iceberg Classifier Challenge being run on Kaggle and concluding January 23, 2018. Statoil, an international energy company, stated that it was supporting the competition to attract a "new perspective on how to use machine learning to more accurately detect and discriminate against threatening icebergs as early as possible." The competition was further supported by C-CORE, a company that used satellite data and a computer vision based surveillance system to identify icebergs. (1)

Icebergs have presented a hazard to ship traffic and other maritime activities, famously exemplified by the 1912 RMS Titanic disaster. According to Statoil, "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." Therefore, identifying iceberg hazards via machine learning was important for maintaining safe working conditions while keeping costs down. This project used machine learning to identify those iceberg hazards using a satellite radar image dataset. (1)

Problem Statement

The challenge was to build a model that correctly identifies whether a remotely sensed target was a ship or an iceberg. The model had to take in two satellite radar bands and an incidence angle where available and assign a probability that the target was an iceberg with 1 indicating that the target was an iceberg and 0 indicating that the target was a ship. As the radar bands were images, this problem was a binary image classification problem. The problem did not require a model of time-series or sequence as the data was not in the form of video.

Given that this is an image classification problem, the solution used a convolutional neural network (CCN) as CNNs perform well for applications in that problem domain. CNNs were used to find and model patterns in the radar images. As the patterns in the two radar bands may be very different, it was found best to use a CNN for each band. The two CNNs were then be merged with the incidence angle input to capture and model that data. The merged data was then fed into a multilayer perceptron (MLP). The MLP had an output layer with a sigmoid activation function to convert the output into a range of 0 to 1 as this was a binary classification problem. To prevent overfitting and thereby make the solution replicable, the weights with the lowest error on a validation subset of the training data were used.

Metrics

The evaluation metric used in the competition was log loss for two classes: (1)

$$log\ loss = -\frac{1}{n}\sum_{i=0}^{n-1}\left[y_i \ln \sigma(x_i) + (1-y_i)\ln\left(1-\sigma(x_i)\right)\right]$$

where n was the number of samples, y_i was the correct label for the sample, and $\sigma(x_i)$ was the model's activation function output (prediction) for a given set of sample input features, x_i. Log loss was a good metric as it measures the distance from the ground truth of each prediction across the whole dataset. Also for a label of 1, the $\sigma(x_i)$ could be thought of as the predicted probability of that class. (2)

II. Analysis

Data Exploration

The datasets were available from the "Data" tab on the Kaggle competition page. There was a train dataset (train.json) and a test dataset (test.json). The datasets were lists of targets with the following fields: (1)

- Id: the id of the image
- band_1, band_2: the flattened array of 75x75 pixel image data. These were float numbers with unit being dB. Band 1 and Band 2 were signals characterized by radar backscatter produced from different polarizations at a particular incidence angle. The polarizations correspond to HH transmit/receive horizontally (HH) and transmit horizontally and receive vertically (HV).
- inc_angle: the incidence angle of which the image was taken. Missing data was marked "na" and missing data only existed in the training data.
- is_iceberg: the target variable (1 for iceberg and 0 for ship). This field only existed in train.json.

Lastly for competition purposes, machine-generated band data were included in the test set to prevent hand labeling and were excluded in scoring. A sample of the datasets are presented in Table 1 below.

Table 1 - Samples of the train.json and test.json datasets

Train Data Sample					
	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**
Test Data Sample					
	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**	**34.96640**	
1	**[-26.058969497680664, -26.058969497680664, -26...**	**[-25.754207611083984, -25.754207611083984, -25...**	**4023181e**	**32.61507**	
2	**[-14.14109992980957, -15.064241409301758, -17....**	**[-14.74563980102539, -14.590410232543945, -14....**	**b20200e4**	**37.50543**	
3	**[-12.167478, -13.706167, -16.54837, -13.572674...**	**[-24.32222, -26.375538, -24.096739, -23.8769, ...**	**e7f018bb**	**34.47390**	
4	**[-23.37459373474121, -26.02718162536621, -28.1…**	**[-25.72234344482422, -27.011577606201172, -23....**	**4371c8c3**	**43.91887**	

There were 1604 examples in the train dataset. The classes in the train dataset were roughly balanced except for the last quintile as shown in Figure 1. As the test dataset (8424 examples) was only used for competition scoring, the train dataset had to be spilt into a train and validation subset. To maintain the class balance, the classes were shuffled via the Keras library's "fit" method. If shuffle was set to False for the fit method, validation_split would continually sample from data with class imbalance as it would use the last fraction of the training data for validation. (3)

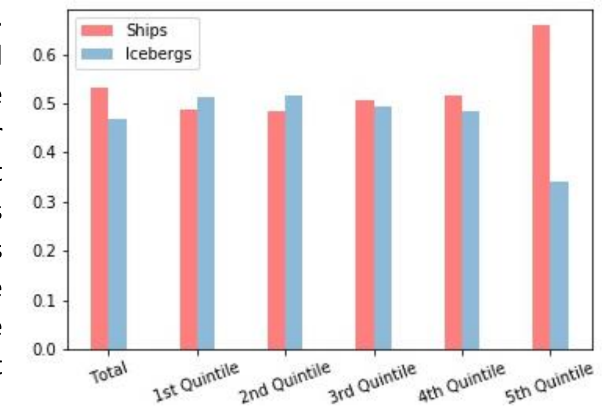


Figure 1 - Class distribution of the train dataset as a fraction of the total number of samples in the dataset and within each quintile

The data was supplied by the Sentinel-1 satellite constellation (composed of Sentinel-1A and Sentinel-1B as of December 2017). (4) The constellation orbited earth 14 times a day at an altitude above 600km. The satellites used C-Band radar to bounce signals off objects on the ground and record the energy backscatter. The backscatter was then converted into an image with more reflected backscatter corresponding to a brighter spot in the image. Patterns in the image were used to characterize the various objects against the ocean background. The ocean background could be made brighter or darker depending on the wind velocity (high or low respectively) and on the incidence angle at which the side looking radar image was captured (as provided in the dataset). (1)

In addition to the ocean background generally appearing darker at a higher incidence angle, radar polarization could cause the image to have different patterns. Sentinel-1 satellites had radar that "can transmit and receive in the horizontal and vertical plane." The dataset for the competition provided a dual-polarization image with radar data given in two channels as described before: band_1 and band_2. The radar image representations of this radar band data are given below as greyscale images in Figure 2. (1)

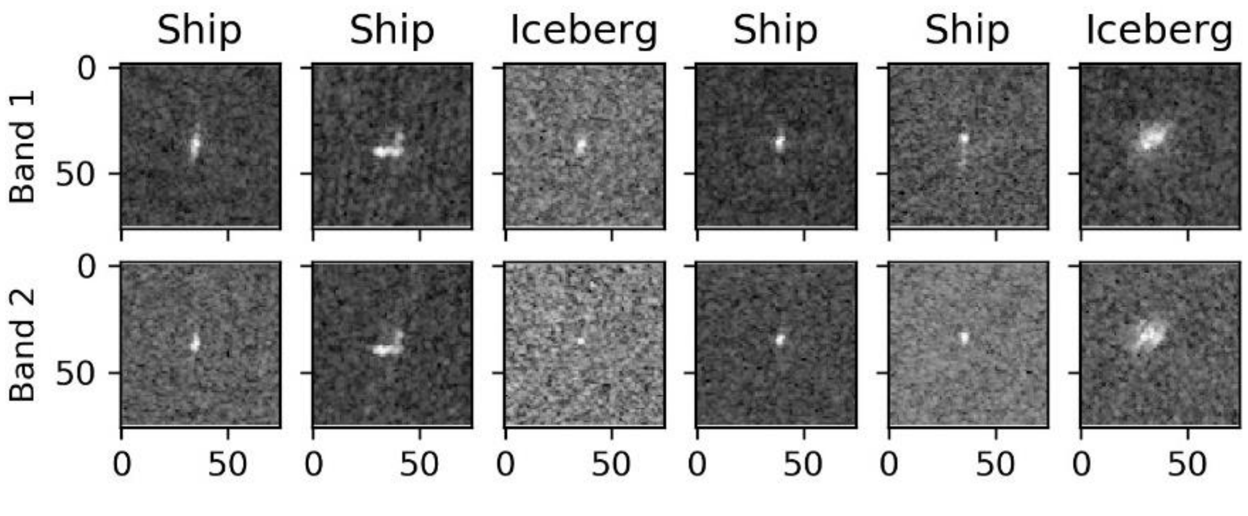


Figure 2 - Band 1 and Band 2 radar band vectors reshaped into 75 pixels by 75 pixels radar images with their coresponding classification

This radar band data was interesting in that it was not linear. As discussed previously, the radar band data was given in decibels (i.e. logarithmic). Moreover, the radar backscatter train data did not have the same mean nor standard deviation for Band 1 and Band 2. As seen in Figure 3, the Band 1 radar data had a larger mean and standard deviation than Band 2. The radar backscatter data will need to be normalized before use in the neural network. Similarly, the inclination angle needed to be normalized. In Figure 4, inclination angle had a mean around 40 degrees and that the distribution was similar between the train and test datasets.

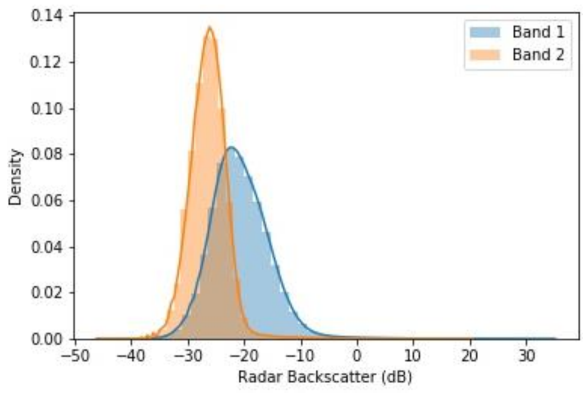


Figure 3 - KDE plot of the radar backscatter distribution for Band 1 and Band 2 in the train dataset

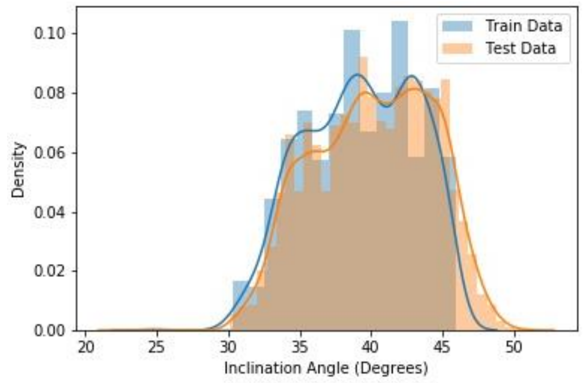


Figure 4 - KDE plots of the satellite inclination angle for the train data and the test data. Missing data from the train dataset was not included.

Algorithms and Techniques

The primary algorithm used in solving the classification problem was a convolutional neural network (CNN). The reason for this was that it was a binary image classification problem and CNNs have had state-of-the-art performance on this problem domain. A typical CNN consists of three convolutional layers, each followed by a pooling layer, followed by a multilayer perceptron (MLP) with activation functions for each layer (except pooling). The whole CNN structure has weights that are updated via backpropagation of the error as described in the Metrics section.

A convolutional layer, shown in Figure 5, takes in an input tensor and slides a window horizontally and vertically across the tensor using a weight matrix called a kernel or filter of a given row and column dimension and yields an output tensor called a feature map through matrix multiplication of the input. In Keras, the input tensor takes the shape of a four-dimensional tensor representing $samples \times rows \times columns \times channels$ such that the input may be image samples of a single channel (greyscale) or multiple channels (such as RGB). (5) The kernel dimension was set 5x5 by default and 64 kernels were used by default to generate a feature map for each kernel. The output for each kernel was passed through an activation function before forming the feature map. The kernels were convolved horizontally and vertically using strides of 1 for each dimension. Where the kernel window

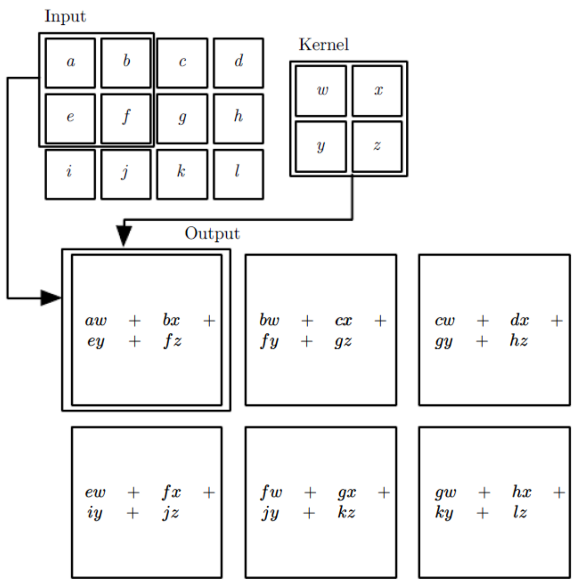


Figure 5 - An example 2-D convolution where the output is restricted to positions where the kernel lies entirely within the image (padding = valid). Here the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor. (6)

extended beyond the input matrix, zeroes were added to the input tensor. Thus “same” padding was used for the convolutional layers as opposed to the “valid” padding used in Figure 5. (6)

Between the convolutional layers, max pooling layers were used. The max pooling layers took the feature maps as input. For each feature map, the max pooling layer would convolve horizontally and vertically a kernel window of size 2x2 using a stride of two to create new feature map outputs that contained the maximum values of the inputs to the kernel window. This had the effect of reducing the dimensionality of the feature map outputs between convolutional layers. At the end of the convolutional block, a global average pooling layer was used. The global average pooling layer used the final feature maps for the convolutional block as input and for each feature map it returned the average value. (6)

Finally, the convolutional block outputs were fed into a MLP consisting of multiple fully connected layers. Each fully connected layer took an input vector (**x**), multiplied it by a weight matrix (**W**), added a bias vector (**b**) and passed it through an activation function (ϕ) of the form: (7)

$$fully\ connected\ layer\ output = \Phi(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b})$$

The column dimension of the weight matrix and bias vector corresponds to the number of artificial neurons in the fully connected layer. 500 artificial neurons were used for the fully connected layers except the output layer where one neuron was used.

Two activation functions were used. For the output neuron, a sigmoid function was used: (7)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The reason a sigmoid activation function was used for the output layer was because this was a binary classification problem and probability of the input representing an iceberg needed to be the output of the model and the sigmoid function generates output in the range of 0 to 1. For the other layers (convolutional and fully connected), an Exponential Linear Unit (ELU) was used: (8)

$$f(x) = \begin{cases} x & if\ x > 0 \\ \alpha(e^{x} - 1) & if\ x \leq 0 \end{cases}$$

where α is greater than 0 and in Keras α is set to a value of 1. The ELU activation function was used as it addresses both the vanishing gradient and dying ReLU problems associated with deep neural network activation functions.

Lastly, the optimizer used was Adaptive Moment Estimation (Adam) as it computes an adaptive step size and it keeps exponentially decaying averages of the first and second moments of the gradients. Adam was chosen as these properties allow it to work well with “very noisy and/or sparse gradients,” avoid getting stuck in local minima, and optimize faster than other algorithms such as stochastic gradient descent and Adagrad. The algorithm pseudocode below is taken directly from Kingma and Ba’s 2015 conference paper “Adam: A Method for Stochastic Optimization”: (9)

```
Require: α: Step size
Require: $\beta_1, \beta_2 \in [0,1]$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
```

```
Require: $\theta_0$: Initial parameter vector
    $m_0 \leftarrow 0$ (Initialize 1st moment vector)
    $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
    $t \leftarrow 0$ (Initialize timestep)
    while $\theta_t$ not converged do
        $t \leftarrow t + 1$
        $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients of stochastic objective at timestep t)
        $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
        $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
        $\widehat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Compute bias-corrected first moment estimate)
        $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Compute bias-corrected second raw moment estimate)
        $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \cdot \widehat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$ (Update parameters)
    end while
    return $\theta_t$ (Resulting parameters)
```

In the above algorithm g_t^2 indicates the elementwise square $g_t \odot g_t$, all operations on vectors are element-wise, and β_1^t and β_2^t denotes β_1 and β_2 to the power t. The algorithm was implemented in Keras with the default parameters of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$. Adam was used to update the weights of the neural network as given above using backpropagation where the gradients g_t were calculated as follows: (7)

- Calculate the gradient for updating the weight matrix $W_{j,i}$ between the output layer, i, and the layer before it, j. Use the activation output of layer j and the first partial derivative of the loss function with respect to the network output, $\hat{y}$. The loss function in this case is the log loss function as specified in the "Metrics" section. The weight matrix $W_{j,i}$ is updated using the update parameters step in the Adam optimizer algorithm.

$$g_{j,i} \leftarrow output_j \frac{\partial L}{\partial \hat{y}_i} \Phi'(\boldsymbol{W_i} \cdot \boldsymbol{x_i} + \boldsymbol{b_i})$$

- Calculate the gradient for each subsequent network connection in the network using the chain rule.

$$g_{k,j} \leftarrow output_k \Phi'(\boldsymbol{W_j} \cdot \boldsymbol{x_j} + \boldsymbol{b_j}) \sum_i W_{j,i} \frac{\partial L}{\partial \hat{y}_i} \cdot \Phi'(\boldsymbol{W_i} \cdot \boldsymbol{x_i} + \boldsymbol{b_i})$$

Benchmark

The Statoil/C-CORE Iceberg Classifier Challenge did not offer a very competitive benchmark. The challenge listed a "Sample Submission Benchmark" log loss metric of 0.6931, otherwise known as chance in binary classification ($\sigma(x_i) = e^{-0.6931} = 0.5$). (1) A more realistic benchmark was offered on C-CORE's website. C-CORE stated that using their "proprietary Iceberg Detection Software […] 100% classification accuracy has been achieved using quad-polarization data, while over 95% classification accuracy has been achieved with dual-polarization modes." (10)

For this project, the benchmark model was a single vanilla CNN on the project band_1 data without including the inclination angle. A vanilla CNN was that as described in Algorithms and Techniques with tree convolutional layers followed by a MLP. The radar band was fed in as a 75x75 pixel image with one channel. The final model architecture was then objectively comparable to this benchmark model.

III. Methodology

Data Preprocessing

For preprocessing, there were three tasks: manage missing inclination angles, normalize inclination angle values, and normalize the radar backscatter values. For the missing inclination angle values, filtering out the samples might have introduced bias and given us fewer training points. As the values were to be normalized to a mean of 0, setting the missing values to 0 would introduced a minimal amount of noise. But first, the inclination angle values that were missing were first removed from the data so that the remaining values could be normalized to a mean of 0 and a standard deviation of 1 via the following equation: (7)

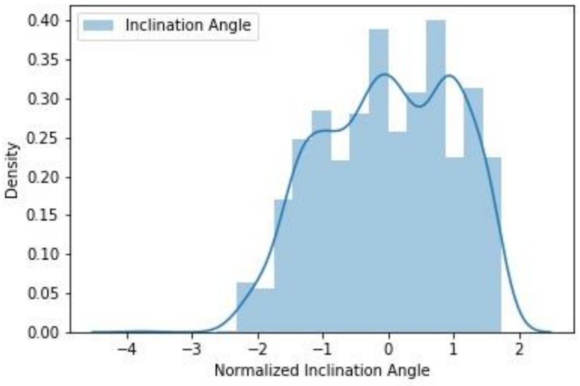


Figure 6 - Normalized inclination angle training feature

$$z = \frac{x - \mu}{\sigma}$$

In the above equation, x is the original value, μ is the mean of the dataset, σ is the standard deviation of the dataset, and z is the normalized value. After normalization of the inclination angle data, the missing value samples were reintroduced with their missing values set to 0. Figure 6 shows the resulting normalized inclination angle feature.

For normalizing the radar backscatter, the values needed to be converted to a range of [0, 1] on a per sample basis as is typical of image data for CNNs. To do this min-max scaling was performed on each data point within a samples band data using the following equation: (7)

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

In the above equation, X is the original value, X_{min} is the minimum value of the band data, X_{max} is the maximum of the band data, and X_{norm} is the normalized value. Also during normalization, the radar band vectors were reused from their 5,625 element arrays to 75 pixels by 75 pixels images in the form of a

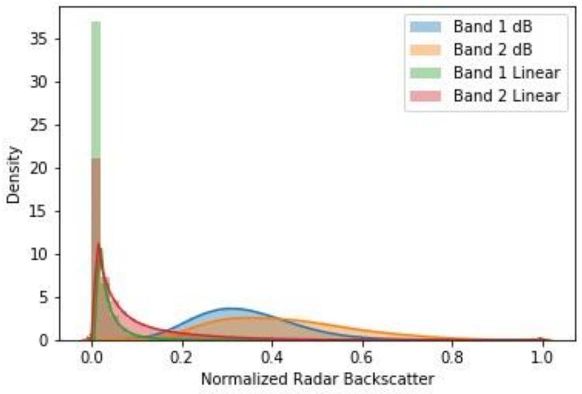


Figure 7 - Normalized radar backscatter training features comparing the as-is decibel values to their linearized values

fourth-order tensor of the form [samples, rows, columns, channels] where channels equals 1 and samples equals 1604 for each radar band. As it was not clear if the decimal values should be converted first to their linear values before normalization, normalization was performed on both decimal and linear versions of the band data as shown in Figure 7. As the linearized data confined the dataset to a vary narrow band, it was decided that normalized data using min-max scaling on the decimal data was to be used.

Implementation

The models were implemented in a Jupyter notebook using Python 3 using the Keras library with the TensorFlow library as a backend. Dataset and matrix operations were performed using the Numpy and Pandas libraries. Visualizations were performed using the Seaborn library. Computation was performed on an Amazon Web Services EC2 instance using model p2.xlarge (1 NVIDIA K80 GPU with 12 GiB of memory, 4 vCPUs, and 61 GiB of memory). The Amazon Machine Image used was "Deep Learning AMI CUDA 8 Ubuntu Version."

The train and test data were first imported from *.json files and stored as Pandas data frames. The train features were assigned to their own Numpy arrays and the train labels were assigned to its own Numpy array. The train features were preprocessed as discussed before. Using Keras with TensorFlow as the backend, model architectures were defined using both the sequential model and functional API. Where multiple inputs were used, a concatenate merge layer was added to merge convolutional branches. One complication occurred when trying to include the inclination angle feature input. Whenever it was added to a model architecture via the concatenate merge layer, the model would stop training. Therefore, the inclination angle feature was not used in the models.

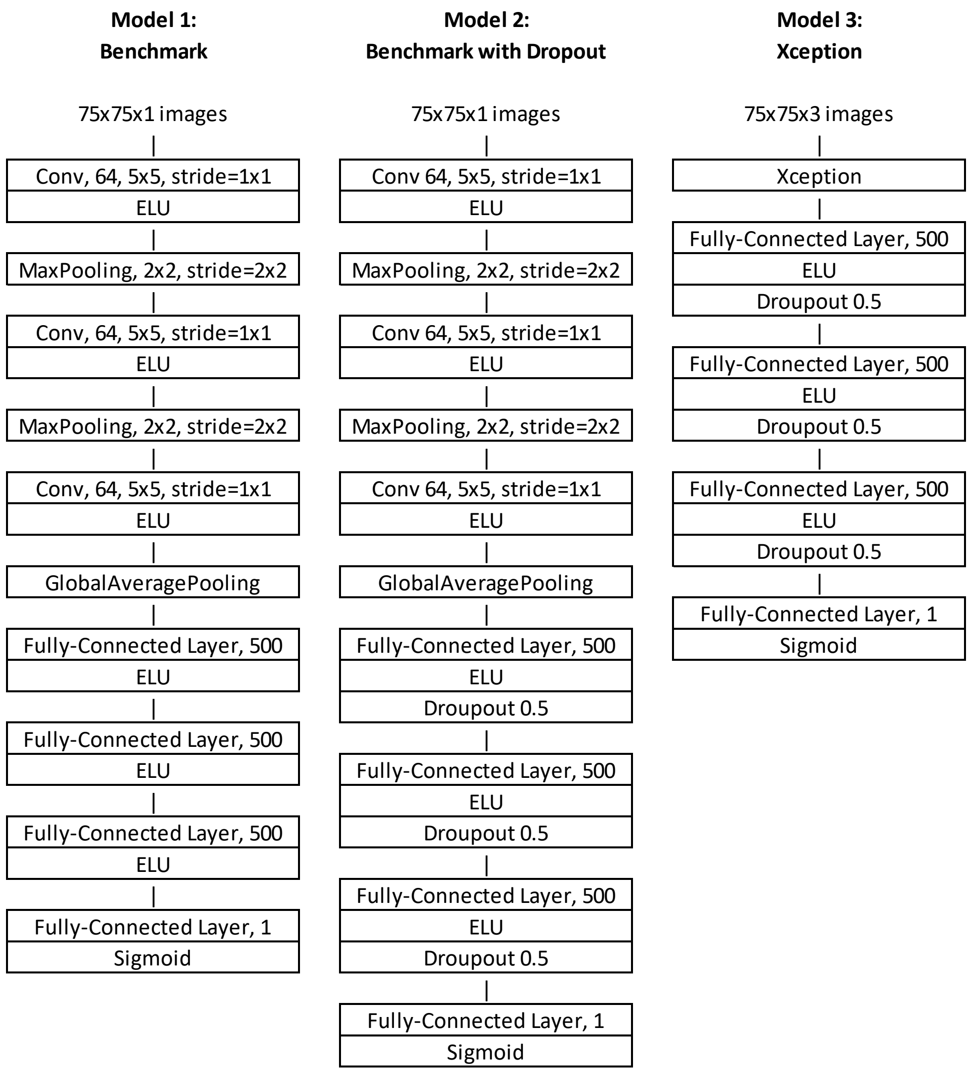
After defining the model architecture, the model was compiled using Adam to optimize the model based on binary cross entropy as the loss function. Accuracy was a reported metric for both the training and validation sets. After compilation, the model was fitted to the training data using 20% of the training data as a validation set with the data shuffled before each epoch. Using a batch size of 100 and 300 epochs, the best weights were saved for the lowest validation loss during training.

Refinement

The progression of models used can be seen in Figure 8, which were all trained using the Band 1 radar training data. The first model architecture used was a vanilla CNN in that it had three convolutional layers interspersed with maximum pooling layers, ending with a global average pooling layer, and followed by a multilayer perceptron. This model achieved a lowest validation cross entropy of 0.35711 (a validation accuracy of 0.8442) after 202 epochs. This first model also serves as our benchmark.

To try to bump up the validation accuracy, dropout layers were added to the CNN between the fully connected layers. The dropout layers were set to the typical dropout probability of 0.5. The dropout layers did not have the desired effect however. The second model achieved a lowest cross entropy loss for the validation set of 0.35273 (a validation accuracy of 0.8411) after 217 epochs. The dropout layers were kept in subsequent models, however, as they did not decrease the validation accuracy (still 84%) and they might help prevent overfitting.

As it was suspected that the CNN model architecture might not be sophisticated enough to represent the data, the third model used the Xception model architecture (Figure 9) as its convolutional block. (11)The Xception model architecture had a Top-1 Accuracy of 0.790 on the ImageNet validation dataset, so it was thought that it would do well in representing the radar image data. Since the ImageNet dataset did not contain radar images, the ImageNet trained weights were not used. and the model's weights were randomized. The resulting Xception model's global average pooling layer was then fed into the multilayer perceptron from the second model with the dropout layers. Unfortunately, the model had a higher cross entropy validation loss of 0.64274 (a validation accuracy of 0.6573) and quickly stopped training after only 13 epochs.



Model 1: Benchmark
75x75x1 images
Conv, 64, 5x5, stride=1x1
ELU
MaxPooling, 2x2, stride=2x2
Conv, 64, 5x5, stride=1x1
ELU
MaxPooling, 2x2, stride=2x2
Conv, 64, 5x5, stride=1x1
ELU
GlobalAveragePooling
Fully-Connected Layer, 500
ELU
Fully-Connected Layer, 500
ELU
Fully-Connected Layer, 500
ELU
Fully-Connected Layer, 1
Sigmoid
Model 2: Benchmark with Dropout
75x75x1 images
Conv 64, 5x5, stride=1x1
ELU
MaxPooling, 2x2, stride=2x2
Conv 64, 5x5, stride=1x1
ELU
MaxPooling, 2x2, stride=2x2
Conv 64, 5x5, stride=1x1
ELU
GlobalAveragePooling
Fully-Connected Layer, 500
ELU
Droupout 0.5
Fully-Connected Layer, 500
ELU
Droupout 0.5
Fully-Connected Layer, 500
ELU
Droupout 0.5
Fully-Connected Layer, 1
Sigmoid
Model 3: Xception
75x75x3 images
Xception
Fully-Connected Layer, 500
ELU
Droupout 0.5
Fully-Connected Layer, 500
ELU
Droupout 0.5
Fully-Connected Layer, 500
ELU
Droupout 0.5
Fully-Connected Layer, 1
Sigmoid

Figure 8 - Model progression from vanilla CNN benchmark in (Model 1) to vanilla CNN with dropout in Model 2 to model via transfer learning of Xception (Model 3).

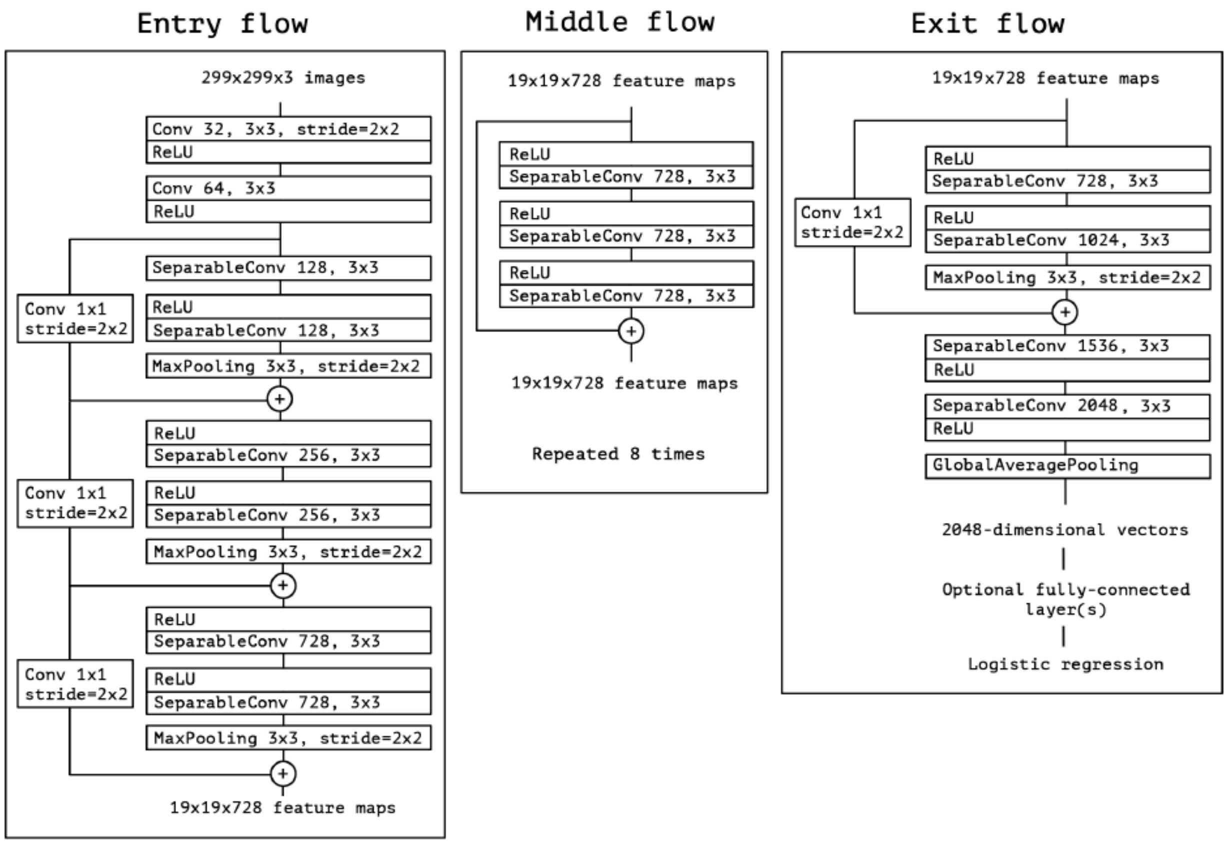


Figure 9 – Xception model architecture. The model architecture used in the Keras library had been modified to accept 75x75x3 pixel images. (11)

IV. Results

Model Evaluation and Validation

With the poor performance of the Xception model, it was decided to use the three convolutional layers from Model 2. These three convolutional layers were used to form two convolutional branches, one for each radar band. Both convolutional branches were merged via a concatenate layer and then fed into a multilayer perceptron with dropout layers. This model was chosen as two convolutional layers each with about 84% accuracy on their given radar band image dataset should give a final accuracy of about 97% (via the product of the probability of each branch failing). The resulting final model is given in Figure 10. It achieved a cross entropy validation loss of 0.3133 (a validation accuracy of 0.8660) after 209 epochs.

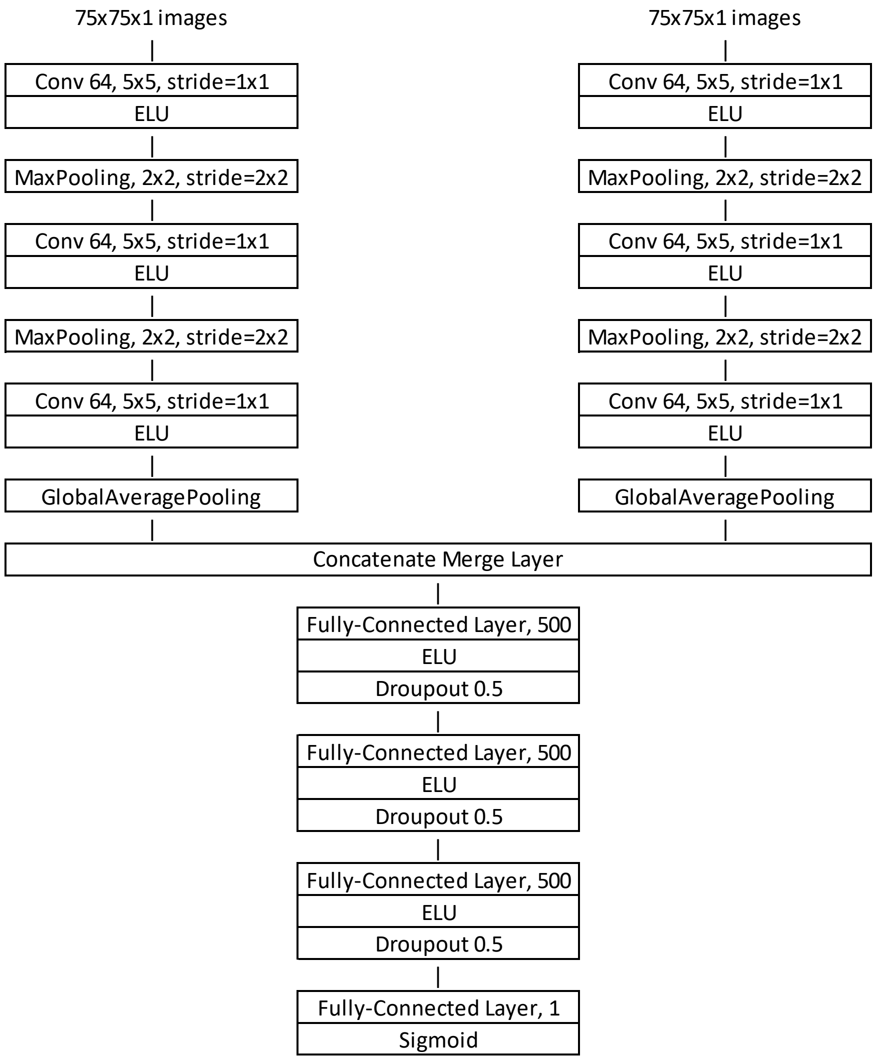


Figure 10 – The final model architecture. It takes two images of dimensions 75x75x1 and outputs a probability of being an iceberg (1) or ship (0). The model uses two convolution branches, merges the result, and sends it through a multilayer perceptron like that used Model 2 in Figure 8.

The final model achieved a cross-entropy loss of 0.2966 on the test data. Thus, the model has proven to be generalize well to unseen data. With small perturbations between the training data and the test data, the model was able to still achieve similar error (i.e. the results were not greatly affected). Therefore, the model has proven to be robust. As the model achieved a lower error than the single input CNNs, the final model aligned with the solution expectations and seems reasonable. As the final model has been shown to generalize well and be robust to changes in the input, the results from the model can be trusted.

Justification

The final model was trained towards accomplishing two goals: outperforming a benchmark model and being useful for classifying radar images of icebergs and ships. The final model clearly accomplished the first goal in that it had a much lower cross-entropy loss of 0.2966 on the test data versus the benchmark CNN’s cross-entropy loss of 0.4161. Therefore, the results from the final model are much stronger than the benchmark results. The final model was able to outperform the benchmark model by taking advantage of twice the amount of data available to the vanilla CNN.

However, the performance of the final model was not significant enough to solve the problem of useful classification of the iceberg/ship radar images. The test cross-entropy loss of 0.2966 translates to an accuracy of 74%. As mentioned previously, C-CORE has a classification accuracy of 95% for dual-polarization radar. Given that as of December 24, 2017 the leading entry for the Statoil/C-CORE Iceberg Classifier Challenge on Kaggle had a test cross-entropy loss of 0.1010 (90% accuracy), it might not be possible to reach 95% classification accuracy using the data provided for the competition. Indeed, Statoil/C-CORE were "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." Therefore, this competition may have been exploratory and additional data may be used in practice to achieve the 95% classification accuracy.

V. Conclusion

Free-Form Visualization

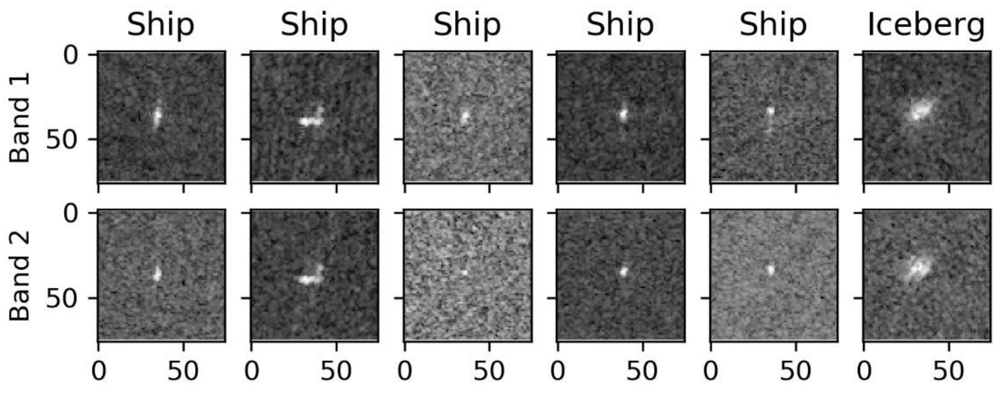


Figure 11 – A look at the first 6 radar band images from the train dataset labeled with the final model's prediction.

To understand how the model arrives at its predictions, Figure 11 shows first 6 samples with its predicted label. From the figure we can see that it has misclassified the third sample as a ship, whereas its correct label (as shown in Figure 2) is an iceberg. The model has clearly learned that Icebergs tend to be larger than ships and that ships tend to be elongated in their shape. Therefore, the model misclassified this sample as a ship because it was smaller than other icebergs, such as last sample in the figure. It likely also determined that the misclassified sample was too elongated as id doesn't appear to be as round as the last sample in the figure. This visualization confirms that the model has been able to determine some characteristic features of the image band data for differentiated icebergs from ships.

Reflection

For this project, a model was constructed and trained on dual-polarization radar image data of icebergs and ships. The model used two inputs of radar image data of two different polarizations and used two convolutional blocks to detect patterns in each image. The convolutional blocks outputted to a multilayer perceptron that detected patterns in the output of the convolutional blocks and classify the inputs as either that of an iceberg or a ship. What was interesting about the project was this was a problem of image classification on non-visible wavelength data. The patterns detected in the radar images were not equivalent to patterns seen by humans. This made finding a suitable architecture difficult in that architectures that performed well on visible wavelength image classification, such as Xception, did not perform well at capturing the patterns in the radar image data. The other aspect that was difficult was the incidence angle. The incidence angle didn't let the model train when added to the multilayer perceptron merge layer. In the end, it was discarded.

Overall, the final model and solution did fit expectations. The problem was a binary image classification problem. Therefore, it was expected that ac CNN would perform well at classifying the images. Furthermore, it was expected that more inputs per sample would decrease the classification error. Indeed, using two radar images as input per sample lowered the cross-entropy loss. This model architecture that concatenates the output of convolutional layers and inputs it into a multilayer perceptron should be used in a general setting to solve image classification problems with multiple inputs. The model architecture is general enough to be applied to problems of the same type.

Improvement

The model could be improved by adding the incidence angle data as input to the model. As previously mentioned, the incidence angle was added to the merge layer. As it prevented the model from training, the incidence angle data was discarded. Using the final solution as the new benchmark, cross-entropy error could be further decreased by incorporating this data. Either the input could be entered in the MLP or be used to modify the inputs to the convolutional layers. One technique that I did not know how to implement was image augmentation using multiple inputs. I tried creating a generator to generate pairs of augmented band 1 and band 2 data, but the model did not train so I decided not to use the technique.

References

1. **Statoil ASA.** Statoil/C-CORE Iceberg Classifier Challenge. *Kaggle.* [Online] Kaggle Inc, October 23, 2017. [Cited: November 8, 2017.] https://www.kaggle.com/c/statoil-iceberg-classifier-challenge.

2. **Nielsen, Michael A.** Chapter 3. *Neural Networks and Deep Learning.* s.l. : Determination Press, 2015.

3. **Chollet, Francois.** Keras FAQ: Frequently Asked Keras Questions. [Online] Keras, 2017. [Cited: November 11, 2017.] https://keras.io/getting-started/faq/#how-is-the-validation-split-computed.

4. **European Space Agency.** Sentinel-1. [Online] ESA Earth Online, 2017. [Cited: November 10, 2017.] https://earth.esa.int/web/guest/missions/esa-operational-eo-missions/sentinel-1.

5. **Chollet, Francois.** Convolutional Layers. [Online] Keras, 2017. [Cited: December 31, 2017.] https://keras.io/layers/convolutional/.

6. **Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron.** *Deep Learning.* Cambridge : MIT Press, 2016. 978-0262035613.

7. **Patterson, Josh and Gibson, Adam.** *Deep Learning: A Practicioner's Approach.* Sebastopol : O'Reilly Media, Inc., 2017. 978-1-491-91425-0.

8. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).* **Clevert, Djork-Arne, Unterthiner, Thomas and Hochreiter, Sepp.** San Juan : Conference on Learning Representations, 2016. arXiv:1511.07289v5.

9. *Adam: A Method for Stochastic Optimization.* **Kingma, Diederik P. and Ba, Jimmy Lei.** San Diego : International Conference on Learning Representations, 2015. arXiv:1412.6980v9.

10. **C-CORE.** Iceberg Surveillance. [Online] C-CORE, 2013. [Cited: November 10, 2017.] https://www.c-core.ca/IcebergSurveillance.

11. **Chollet, Francois.** *Xception: Deep Learning with Depthwise Separable Convolutions.* 2016. arXiv:1610.02357v3.