Machine Learning Engineer Nanodegree

Capstone Project

Aymar Thierry LIGA

28 January, 2018

I. Definition

I.1) Project Overview

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 **[1] [5] [7]**.

The remote sensing systems used to detect icebergs are housed on satellites over 600 kilometers above the Earth. The Sentinel-1 satellite constellation is used to monitor Land and Ocean. Orbiting 14 times a day, the satellite captures images of the Earth's surface at a given location, at a given instant in time. The C-Band radar operates at a frequency that "sees" through darkness, rain, cloud and even fog. Since it emits its own energy source it can capture images day or night. Satellite radar works in much the same way as blips on a ship or aircraft radar. It bounces a signal off an object and records the echo, then that data is translated into an image. An object will appear as a bright spot because it reflects more radar energy than its surroundings, but strong echoes can come from anything solid - land, islands, sea ice, as well as icebergs and ships. The energy reflected back to the radar is referred to as backscatter. When the radar detects an object, it can't tell an iceberg from a ship or any other solid object. The object needs to be analyzed for certain characteristics - shape, size and brightness - to find that out. The area surrounding the object, in this case ocean, can also be analyzed or modeled **[1] [5] [7]**.

Many things affect the backscatter of the ocean or background area. High winds will generate a brighter background. Conversely, low winds will generate a darker background. 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. We 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, we can get what is called a dual-polarization image **[1] [5] [7]**.

I.2) Problem Statement

In this project, we're trying to build an algorithm that automatically identifies if a remotely sensed target is a ship or iceberg, through function approximation by training a convolutional neural network (CNN).

CNN generally makes strong and mostly correct assumptions about the nature of images. Its goal is to learn higher-order features in the data via convolutions. CNN tends to be most useful when there is some structure in the input data. In our case for example, images have a specific set of repeating patterns and input values next to each other spatially related. With local connections, neurons can extract elementary characteristics such as shape, size or brightness in our case. These features will be combined by the subsequent layers in our CNN in order to detect higher-order features.

I.3) Metrics

The metric used for this Kaggle competition is a **binary cross entropy** defined in theory as:

$$ERROR = -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)(\ln(1-\hat{y}_i))+y_i\ln(\hat{y}_i)$$

Where y_i is the true variable to predict and $\hat{y}_i$ is the y_i estimate. Since our benchmarks use accuracy rate, we will try to mimic this metric by way of comparison. We know on the other hand that we have more ships than iceberg in our data set. Because of this, we are particularly interested in predicting which image is ship or an iceberg accurately. It would seems that using accuracy as a metric for evaluating a particular model's performance would be appropriate. Additionally, identifying some images that are not iceberg as images who are, would be detrimental for our analysis. Therefore, a model's ability to precisely predict iceberg or ship is more important than the model's ability to recall images. We can use F-beta score as a metric that consider both precision and recall:

$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{(\beta^2 \cdot precision)+recall}$$

The performance of our model in this project with the goal to understand how well it gives the correct classification will be the F-score. The confusion matrix is used to better understand how well a classifier is performing based on giving the correct answer at the appropriate time. We measure these answers by counting the number of True positives, False positives, True negatives and False negatives. In traditional statistics, a false positive is also known as "type I error" and a false negative is known as a "type II error." By tracking these metrics, we can achieve a more detailed analysis on the performance of the model beyond the basic percentage of guesses that were correct. We can calculate different evaluations of the model based on combinations of the aforementioned four counts in the confusion matrix. The accuracy is the degree of closeness of measurements of a quantity to that quantity's true value **[17]**.

$$accuracy = \frac{(TP+TN)}{(TP+FP+TN+FN)}$$

It measures how often the classifier makes the correct prediction. Precision tells us what proportion of images we classified as iceberg, actually were iceberg. It is the ratio of,

$$precision = \frac{TP}{(TP+FP)}$$

Recall tells us what proportion of images that actually were iceberg we classified as iceberg. It is the ratio of,

$$recall = \frac{TP}{(TP + FN)}$$

Since we have a classification problem that is skewed in their classification distributions, we will combine recall and precision to get the F1 score, which is the harmonic mean of the precision and recall. This score can range from 0 to 1, with 1 being the best possible F1 score.

The specific contributions of this project are as follows: we trained a convolutional neural network on the subsets of Statoil/C-CORE Iceberg Classifier Challenge. We wrote optimized CPU of 2D convolution in Keras with Tensorflow backend. Our final network contains 4 convolutional and 2 fully-connected layers, and this depth seems to be important: we found that removing any convolutional, fully-connected layer or changing the order of filters in the architecture of the model resulted in inferior performance. We also found that by adding fully connected dense layers we were able to minimize the loss rate but this comes with an increase in training time and a decrease in F score.

II. Analysis

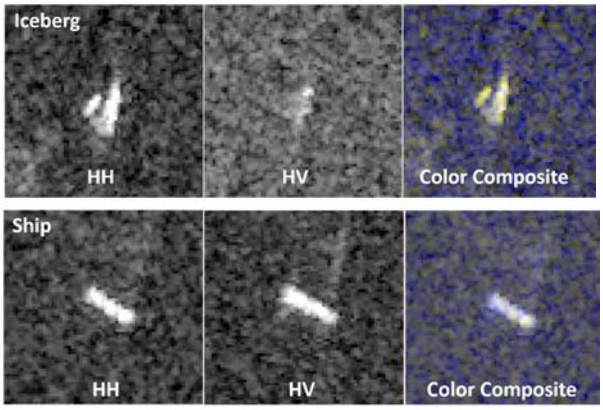
II.1) Data exploration

The data (train.json, test.json) are presented in json format. We can find below the fields description:

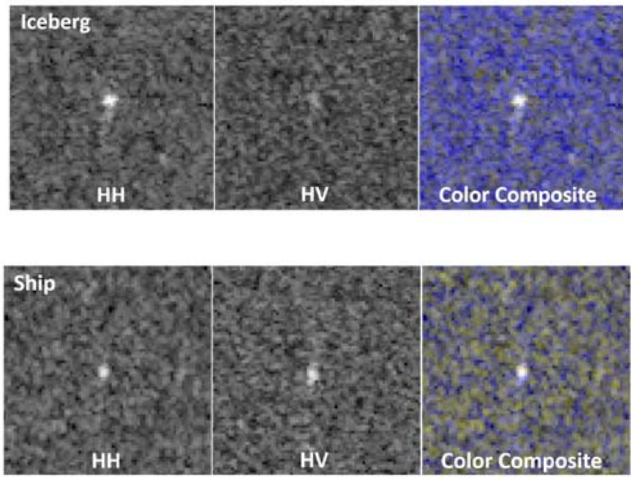
- Id - the id of the image
- band_1, band_2 – the flattened image data. Each band has 75 x 75 pixel values in the list, so the list has 5625 elements. These values are not the normal non-negative integers in image files since they have physical meanings - these are float numbers with unit being dB. Band 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. 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 is_iceberg is an iceberg, and 0 if it is a ship. This field only exists in train.json **[1] [5] [7]**.

The Statoil/C-CORE Iceberg Classifier Challenge has 75*75 images with two bands belonging to two categories. The labels are provided by human experts and geographic knowledge on the target.

The data come from Statoil, an international energy company working closely with C-CORE. C-CORE have been using satellite data for over 30 years and have built a computer vision based surveillance system. The data are seen with two channels: HH (transmit/receive horizontally) and HV (transmit horizontally and receive vertically). This can play an important role in the object characteristics, since objects tend to reflect energy differently. Easy classification examples are seen below. These objects can be visually classified. But in an image with hundreds of objects, this is very time consuming **[1] [5] [7]**.



Below, we see challenging objects to classify but we need to figure out if these objects are Ship or Iceberg.

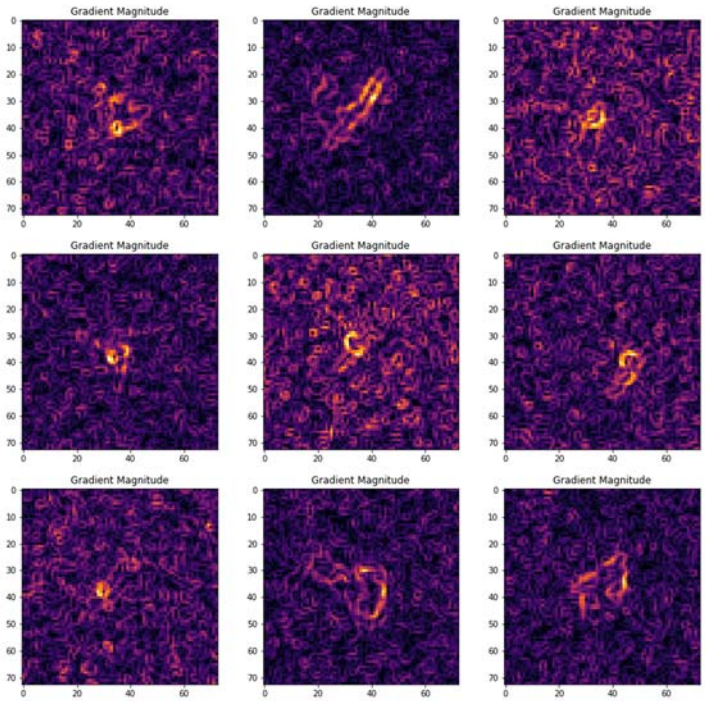


We have 1 604 images on the training set: 851 ships and 753 icebergs. On the other hand, we have 8 424 images on the testing set, 5 times as many images than training images. We did not have information about latitudes or longitudes of images. Features engineering came from radar echoes. By investigating the statistical properties of the two separate bands and coercing incidence angle to numeric, or combining training and test sets for feature engineering, we found that for the first band, there are some significant differences. The middle 50% range has around the same size for both, but the minimum, maximum, standard deviation, median, and mean all show noticeable differences in some range of the values. Evidently, these basic variables seem to have some sensitivity to what we are trying to measure. We encounter similar results for the second band **[27].**

By plotting some raw images on the notebook, it appears that the background is not really random noise but rather, has some spatial correlations. If the background is dominated by waves rather than noise, then spatial correlations would clearly be expected. The ships seem to have a more regular structure, with a pronounced skewness in the blobs for larger signals. Some of these blobs are not that high above noise, so it may be advantageous to first transform the images in some way to enhance the contrast between the signals and the background **[27]**. Inc_angle is the most important feature, but it does not appear to be sufficient on its own. When training, we replaced missing values angles with 0.

II.2) Exploratory Visualization

The process of transforming the images is defined by FIR filters. The scipy convolved2d function will run a convolution of two arrays, so we only need to define the kernels. Images can also be smoothed with the goal to blur the features. By smoothing images, we also enhances the contrast between bright and dark regions. An X-derivative of the original images will be antisymmetric with respect to reversing the values around the x-axis. This will provide some level of edge detection in the x-direction. We can also look at the magnitude of the gradient, that is treat the x and y derivatives as a gradient vector at each position and take the magnitude at each point. Below, we can observe the plots of band1 regarding the gradient magnitude **[27]**.



We see interesting circular shapes everywhere in these images. The ships in particular, show fairly bright edges and most create nice loops. Some of the transformations of the images we made show that edge detections based methods seem to get some nice features, and smoothing may help out with images with small signal size **[27]**.

II.3) Algorithms and Techniques

Deep learning reflects the convergence of a word which can produce much more data about speech, images, social network or even network traffic. We need the pattern or to understand what it's really going on behind these images, or text, or network traffic samples. Behind deep learning, we can build electronic system, hardware and software which can recognize patterns and extract useful non trivial patterns about what is going on. In a more formal way, we define deep learning as neural networks with a large number of parameters and layers in the Convolutional neural networks which are one of four fundamental network architectures.

In Neural networks, every neuron in the network is connected to every neuron in adjacent layers. In particular, for each pixel in the input image, we encoded the pixel's intensity as the value for a corresponding neuron in the input layer. For the 75×75 pixel images we are using, this means our network has 5 625 (=75×75) input neurons. We then trained the network's weights and biases so that the network's output would correctly identify the input image. Networks with fully-connected layers do not take into account the spatial structure of the images. They treat input pixels which are far apart and

close together on exactly the same footing. Such concepts of spatial structure must instead be inferred from the training data.

The main principle behind convolutional neural networks is that they use architectures which try to take advantage of the spatial structure. Convolutional neural networks use three basic ideas:

- Local receptive fields: here, we only make connections in small, localized regions of the input image. To be more precise, each neuron in the first hidden layer will be connected to a small region of the input neurons. That region in the input image is called the local receptive field for the hidden neuron
- Shared weights*:* are weights that define the map from the input layer to the hidden layer; share weights and share bias define what it is called a kernel or filter.
- Pooling: they are usually used immediately after convolutional layers. What the pooling layers do is simplify the information in the output from the convolutional layer. A pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map. One common procedure for pooling is known as max-pooling. In max-pooling, a pooling unit simply outputs the maximum activation in the input region **[10]**.

Generally, CNNs have best-in-class performance on problems that significantly outperforms other solutions in domain like speech, language, vision, etc.… They reduce the need for feature engineering and have architectures that can be adapted to new problems relatively easily. On the other hand, CNNs require a large amount of data, are extremely computationally expensive to train, do not have a strong theoretical foundation meaning that defining the topology, flavor, training method as well as hyper parameters for deep learning is a black art with no theory to guide **[26]**.

The architecture of our network contains six learned layers – four convolutional and two fully-connected. Usually, convolutional layers are built by doubling the number of filters as we add hidden layers in the network. We used an activation function in every hidden layer to propagate the output of one layer's node forward to the next layer. The standard way to model a neuron's output f as a function of its input x is with

$$f(x) = \tanh(x)$$

Or

$$f(x) = \left(1 + \exp(-x)\right)^{-1}$$

Activation functions into our hidden neurons help to introduce nonlinearities into our network's modelling capabilities. This allows the network to learn patterns in the data within a constrained space. In terms of training time with gradient descent, these aforementioned saturating nonlinearities are much slower than the non-saturating nonlinearity

$$f(x) = \max(x,0)$$

Neurons with this nonlinearity are Rectified Linear Units (Relus) **[3] [4]**. Deep convolutional neural networks with Relus seem to train several times faster than their equivalents with tanh unit. The size of our network made overfitting a significant problem. In our network, we only have 1 604 labeled training examples, and performing modelling on these training have generated a lot of parameters, so we used several effective techniques for preventing overfitting. In convolutional networks, more filters mean a bigger stack, which means that the dimensionality of our convolutional layers can get quite large. Higher dimensionality means we will need to use more parameters which can lead to overfitting **[13] [17]**.

Dropout is a mechanism used to improve the training of neural networks by omitting a hidden unit. It is driven by randomly dropping a neuron so that it would not contribute to the forward pass and

backpropagation **[17]**. Applying Max pooling layers and dropout layers to our networks help reduce the number of parameters in the model. In our design, 20% of neurons will be drop during training.

II.4) Benchmark

There has been a significant amount of study on automated target recognition in radar imagery. Of interest to this body of work are iceberg and ship target detection and discrimination. Hidetoshi FURUKAWA et al. (April 2017) in Deep Learning for Target Classification from SAR Imagery used a CNN with augmented training data and achieved a classification accuracy of 99.6% on classifying the SAR images from the Moving and Stationary Target Acquisition and Recognition (MSTAR) public release data set **[8]**. Howell Carl (2008), used the cross-validation leave-one-out methodology and a Bayesian based maximum likelihood quadratic discriminant approach for target discrimination. He has used known iceberg and ship samples to build the quadratic discriminant functions and produce at least 90% classification accuracy for said targets for the three different sensors: RADARSAT-1, ENVISAT ASAR HHIHV, and EMISAR **[3]**.

Chen et al. firstly indicated that one single convolutional layer could effectively extract SAR targets feature representation with unsupervised learning using randomly sampled SAR targets patches and achieve the accuracy of 84.7% in 10-class classification tasks [22]. Muhammad Jaleed Khan et al, in Automatic Target Detection in Satellite Images using Deep Learning, used the publicly available military target dataset containing 500 aircraft patches, 5000 non-aircraft patches and 26 test images taken from Google Earth. They resized the patches to 32×32 and used them to train the CNN for classification in satellite images and Edge Boxes algorithm for object detection. For each detected object in the input image, their trained CNN was used to predict whether it was aircraft or a non-aircraft object. They found that the proposed system reached the maximum precision and recall both of 100% and achieved an average precision and recall of 77.9% and 91.3% respectively on the complete dataset [21].

Morgan et al. proposed an architecture of three convolutional layers, following a fully connected layer of Softmax as a classifier, increasing the accuracy to 92.3%. Wilmanski et al. explored different learning algorithms of training CNNs, finding that the AdaDelta technique that can update the various learning rates of hyper-parameters outperformed the other techniques such as stochastic gradient descent (SGD) and AdaGrad [24]. Recently, a five-layer all-convolutional network was proposed. The authors adopted a drop-out method in a convolution layer and removed the fully connected layer to avoid over-fitting since the limited training data was insufficient to train the deep CNNs. The experiment results showed the state-of-the-art performance of SAR target recognition in the MSTAR dataset, reaching an accuracy of 99.13% [23].

III. Methodology

III.1) Data Preprocessing

Before building our model, we reshaped and feature scaled the images. Some of the incident angle from the satellite are unknown and marked as "NA". We replaced them with 0 and find the indices where the incident angle was >0. This allows us to use a truncated set or the full set of training data. To help our model in the learning phase, we added more data not by augmentation but by including horizontally and vertically flipped data thanks to the OpenCV library for the refined model **[16]**.

For the initial model, we've concatenated band1 and band2 data from the training set and treated angle variable as a separate dataframe. We have done the same operation in the testing set. We have not done pre-processing in the images in any other way. We've then trained our network on the raw RGB values of the pixels and found slightly better results training with only the known incident angles.

We have also hold out part of the available data as a validation set using the train_test_split helper function from scikit-learn with a 75% random split into training and validation sets and a random state sets to 123. This strategy, when the angle data were considered as a separated dataframe, have generated validation and training angle data.

III.2) Implementation

We have images with 75 pixels in width by 75 pixels in height with 3 channels of RGB information. This would create 16 875 connection weights per hidden neuron, showing that using a fully connected multilayer network will end up creating a massive number of connection when modelling our image data. With convolutional neural network, our input data are transformed from the input layer through all connected layer.

Our initial CNN architecture has 2 input layers, seven feature-extraction layers from our image which are concatenated with a hidden angle layer, and one classification layers. Every feature-extraction layers but the last have the same pattern: 2 convolutional layers each with the same number of filter, a max pooling layer, a dropout layer which drops 20% of neurons during training, and a global max pooling layer at the end of every hidden layers. In addition, to batch normalize the 1st hidden layer and the hidden layer with the angles which has 128 filters, we added a global max pooling layer at the end of every hidden group of hidden layer. Our convolutional layers transform their concatenated input data by using a patch of locally connecting neurons from the previous layer and the weights to which they are locally connected in the output layer. The layer will compute a dot product between the region of the neurons in the input layer and the weight to which they are locally connected in the output layer. Convolutional layers have parameters for the layer and additional hyper parameters. Gradient descent is used to train the parameters in this layer such that the class scores are consistent with the labels in the training set. We assigned respectively 16, 32, 64, and 128 filters from the first to the last hidden layer. We have initialized the weight as a 3*3 matrix in every hidden layer. This weight shall now run across the image such that all the pixels are covered at least once, to give a convolved output. The weights are learnt such that the loss function is minimized. Here, the initial convolution layer extracts more generic feature, while as the network gets deeper, the features extracted by the weight matrices are more and more complex and more adapted to the problem we are facing. In every hidden layer, we used an ELU activation function, with no striding. The sigmoid activation function is used for the output layer.

Finally, we have the classification layers in which we have 1 fully connected layers to take the higher-order features and produce class probabilities or scores. This layer is fully connected to all of the neurons in the previous layer. The output of this layer produces a two-dimensional output of the dimensions [$b \times 2$], where b is the number of examples in the mini-batch and 2 is the number of classes. The convolution and max pooling layers would only be able to extract features and reduce the number of parameters from the original images. We tuned our model by

III.2) Refinement

To reach a 93% accuracy rate, we revisited our initial architecture by designing a new one. Our new CNN architecture has an input layer, four feature-extraction layers, and two classification layers. All our feature-extraction layers have the same pattern: a convolutional layer, a max pooling layer, and a dropout layer. To tackle the problem of poor performance erases when doubling filters as hidden layers increase, we assigned respectively 64, 128, 128, and 64 files from the first to the last hidden layer. We have initialized the weight as a 3*3 matrix in every hidden layer. In our output layer we used a sigmoid activation function. During training, we instructed our CNN to drop about 30% of neurons.

We tuned our new deep network by considering a weight initialisation strategy, an activation function, a loss function, an optimization algorithm, by defining a learning rate, as well as mini-batching and regularisation. Initializing weights is a key starting place in the learning process for neural networks

and deep networks. If weights are too large, it can mean large outputs and large gradients, which can have a detrimental effect during learning. The weights were initialized with the default values of kernel initializer and bias initializer, which are 'glorot_uniform' and 'zeros' respectively. We used activation functions for hidden layers to introduce nonlinearities into our network's modelling capabilities and to propagate the output of one layer's nodes forward to the next layer. ReLU activation functions are the most popular type of hidden unit we see in modern deep networks. We see ReLUs used commonly in CNNs as the activation function of convolution layers. Stochastic Gradient Descent (SGD) has been found to speed up learning with ReLUs when compared to sigmoid and tanh functions. ReLUs are also computationally cheaper than sigmoid and tanh functions. Because the gradient of a ReLU is either zero or a constant, it is possible to reign in the vanishing exploding gradient issue. ReLU activation functions have shown to train better in practice than sigmoid activation functions **[17]**.

We needed to know how well our optimization function is doing by defining a loss function. The goal was to find a model parameter that gives us predictions able to minimize the loss. We calculated a metric based on the error we observed in the network's predictions. We then aggregated these errors over the entire dataset and averaged with the goal to have a single number representative of how close the neural network was to its ideal. When building neural networks for classification problems, the focus is often on attaching probabilities to these classifications.

Overfitting is the tendency of machine learning workflows to learn the training dataset so well that it performs poorly on unseen datasets. To detect overfitting, we performed a cross-validation on the training set and we evaluate the performance on a held-out test set. It's not always clear how many layers the network should have or how many nodes to place in each layer, or how many epochs to use. One method use in practice is breaking the dataset into 3 sets: the training set, the validation set and the testing set. In the other hand, regularisation can give us a more efficient representation of a model by using different methods to prevent our parameter values from becoming too large. The right combination of settings for regularisation is done via manual tuning but also be accomplished with 'random search' or 'grid search'.

In our project, we've used dropout after every hidden and dense layers to mitigate overfitting besides reducing large weights. Before training our model, we need to configure the learning process, which is done via the compile method. We've instantiated our optimizer and passed it into our method. The most prevalent methodology for optimizing deep learning networks is stochastic gradient descent (SGD). The vanilla version of SGD uses gradient directly, but gradient can be nearly zero for any parameter, causing SGD to take tiny steps in some cases, and steps that are too big for situations in which the gradient is too large. We used Adam to alleviate these issues with a 0.001 learning rate and 0.0 decay. The learning rate hyper parameter in neural networks is one of the most important hyper parameter to set when tuning a neural network. It has a strong impact on both the stability and efficiency of training times. If the rate is too large, the network training will likely be unstable or diverge. If our learning rate is too small, training can take orders of magnitude longer than it needs. We ideally want to see a large learning rate in the beginning of training and then see it fall over time as we approach convergence of the training process. We've chosen to ignore the learning rate decay because in theory, Adam already handles learning rate optimization.

When training our deep learning model, the checkpoint is the weights of the model. These weights can be used to make predictions as is, or used as the basis for ongoing training. The Model Checkpoint callback class allows us to define where to checkpoint the model weights, how the file should be named and under what circumstances to make a checkpoint of the model. The callback API allows us to specify which metric to monitor, the loss in our case, on the training or validation dataset. We've specified to look for an improvement in minimizing the score. The filename that we've used to store the weights includes the epoch number and the loss. We've passed the Model Checkpoint to the training process by calling the fit function on our model. When training our neural network, we were interested in obtaining a network with optimal generalization performance. Early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration. Up to

a point, this improves the learner's performance on data outside of the training set. Past that point, however, improving the learner's fit to the training data comes at the expense of increased generalization error. We wanted to stop training when the loss did not improve ten epochs or when training goes above the validation by using early stopping. We also used the Reduce Learning rate **[17]** callback to reduce the learning rate by a factor of 0.1 when the learning stagnates and if no improvement is seen after seven epochs. We know that as mini-batch size increases, more computation means gradients might be smoother but are more costly to compute. If our mini-batch size increases by a factor of two, we need to increase the number of epochs by a factor of two in order to maintain the same number of parameters updates. In this project, we've chosen a batch size of 32 with a number of epochs of 50.

With our initial architecture, our loss function is 0.2996 while the accuracy rate achieved an 89.03% accuracy rate. By adjusting our model and adopting a measure of performance adapted to classification issue with imbalanced classes, our loss function dropped from 0.2996 to 0.1853 and our f1 validation score showed that we correctly classified 93.94% of our input data.

IV. Results

IV.1) Model Evaluation and Validation

Using the CNN architecture described above, we have achieved a F1 score of 98.6% and a loss of 0.0498 in the Statoil/C-CORE Iceberg Classifier Challenge training data set after around 4 hours of training without data augmentation or transfer learning. Here is a list of our specific layer architecture:

- INPUT layer [75*75*3] will hold the raw pixel values of our image with width of 75, height of 75, and with 3 colors channel R, G, B.
- CONV layer: [75*75*64] with a kernel size of 3 will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in a volume such as [73*73*64] because we used 64 filters.
- RELU function will apply elementwise activation, such as ***max(0, x)*** thresholding at zero in every hidden layer. This leaves the size of the volume unchanged ([73*73*64]).
- Max pooling layer with a striding size of 2 will perform a down sampling operation along the spatial dimensions (width, height) resulting in volume such as [36*36*64].
- Dropout layer: will take into account only 30% of neurons during the training but the volume remains unchanged [36*36*64].
- CONV layer [34*34*128] with a kernel size of 3 will compute the output of neurons that are connected to local regions of the previous drop out layer
- Max pooling layer with a striding size of 2 will perform a down sampling operation along the spatial dimensions (width, height) resulting in volume such as [17*17*128].
- Dropout layer: will take into account only 30% of neurons during the training but the volume remains unchanged [17*17*128].
- CONV layer [15*15*128] with a kernel size of 3 will compute the output of neurons that are connected to local regions of the second drop out layer.
- Max pooling layer with a striding size of 2 will perform a down sampling operation along the spatial dimensions (width, height) resulting in volume such as [7*7*128].
- Dropout layer will take into account only 30% of neurons during the training but the volume remains unchanged [7*7*128].
- CONV layer [5*5*64] with a kernel size of 3 will compute the output of neurons that are connected to local regions of the third drop out layer.
- Max pooling layer with a striding size of 2 will perform a down sampling operation along the spatial dimensions (width, height) resulting in volume such as [5*5*64].

- Dropout layer will take into account only 30% of neurons during the training but the volume remains unchanged [5*5*64].
- The Flatten layer to flatten the data for the dense layers.
- FC1 layer with 512 filters and a RELU activation function will compute the class scores, resulting in volume of size [1*1*2] where each of the 2 numbers correspond to a class score. Each neuron in this layer will be connected to all the numbers in the previous volume. The probability inside the dropout layer is set to 30%.
- FC2 layer with 256 filters and a RELU activation function will compute the class scores, resulting in volume of size [1*1*2] where each of the 2 numbers correspond to a class score. Each neuron in this layer will be connected to all the numbers in the previous volume. The probability inside the dropout layer is set to 30%.

In this way, CNNs transform the original image layer by layer from the original pixel values to the final class scores. Convolutional and Fully connected layers perform transformations that are a function of not only the activation in the input volume but also of the weights and biases of the neurons which will be trained with gradient descent so that the class scores that the CNN computes are consistent with the labels in the training set for each image. On the other hand, the max pooling layers will implement a fixed function **[28]**.

IV.2) Justification

To validate the model, we have generated predictions for the validation data set. Our model look only at the training data when deciding how to modify its weights. At every epoch, our model checks how it's doing by checking its F1 score on the validation set but don't use any part of the validation set for the backpropagation step. The testing set is used to test our model because even if we used the validation set to update the weights, our model selection is biased in favor of the validation set. We have a F1-score of 93.94% and a log loss of 0.1853 on the validation data. Our log loss on the test data is 0.1714 versus the 0.0801 of the leader, placing us to top 45% of the participants. The size of the training data set versus the size of the test data set seem to be a major issue here. To overcome this issue, we think that semi-supervised learning coupled with standard data augmentation could be the solution.

In term of classification rate, our final result is stronger that the solution proposed by Howell Carl in 2008. By using the cross-validation leave-one-out methodology and a Bayesian based maximum likelihood quadratic discriminant approach for target discrimination, he produced a 90% classification accuracy for said targets for the three different sensors: RADARSAT-1, ENVISAT ASAR HHIHV, and EMISAR. On the other side, our final result seems less accurate than the solution proposed by Hidetoshi FURUKAWA et al. (April 2017) in Deep Learning for Target Classification from SAR Imagery. They used a CNN with augmented training data and achieved a classification accuracy of 99.6%.

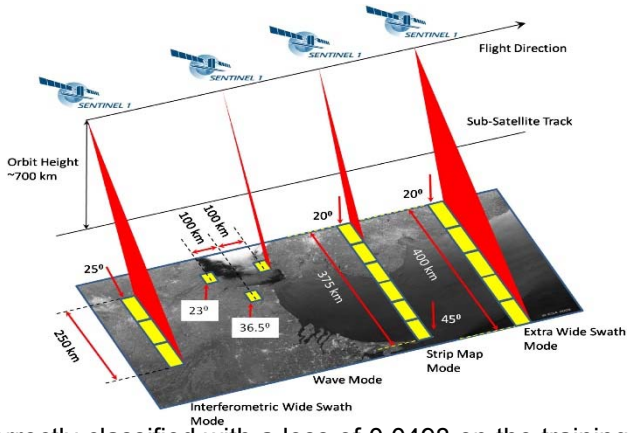
The comparison of our solution and our benchmarks made us believe that the final solution we developed is significant enough to have solved the problem of automatically identifying if a remotely sensed target is a ship or iceberg.

V. Conclusion

V 1) Free Form Visualization

We think that the incidence angle variable can play a great role in modelling satellite images classification. With the objectives of Land and Ocean monitoring, the SENTINEL-1 satellite illustrated below is composed of two polar-orbiting satellites operating day and night, and performs Radar imaging, enabling them to acquire imagery regardless of the weather. A pervasive feature of spatial analysis is that data are often presented in a two-dimensional format. A map view is great, and we all

know the value that such spatial products can have, but sometimes creativity can be employed to extract valuable three-dimensional data from a two-dimensional representation. We know that angle of incidence influences significantly the received signal, particularly in the modes of sensor operation that use the full swath of the orbit track. Wind-induced waves can cause a roughening of the water surface which results in a high return signal making the separation of water and land problematic using radar. This latter problem can be mitigated, by making use of the difference in the relationship between the incidence angle of the radar signal, and backscatter **[29]**. We think that performing that difference might have enhance our F-score performance.



V.2) Reflection

We have trained a CNN model using images from Statoil/C-CORE Iceberg Classifier Challenge for classification. With our architecture made with an input layer, four feature-extraction layers, and two classification layers, where every hidden layer has a kernel size of 3, a relu activation function for nonlinearities purposes, a max pooling layer of size 3 as well as a stride of size 2 and a 30% drop out strategy, we've showed that 98.6% of our images are correctly classified with a loss of 0.0498 on the training data set by using an Adam optimizer function with a learning rate of 0.001 and a decay of 0.0, setting the batch size to 32 and using an early stopping strategy which monitor the validation loss with a patience of 10 and finally, fitting the model with 50 epochs. Similarly, the validation F1 score achieved a 93.7% classification rate while the validation loss achieved a score of 0.1853.

Our final model kept learning regularly by improving its performance until epoch 23 when the validation loss and the validation F1 score ceased their learning process. We may need to improve the accuracy as well as the loss by may be using data augmentation or transfer learning. The Image net classification developed by Hinton at all can eventually be used as transfer learning to amend the accuracy of this model.

This project has 2 main interesting aspects: the test data set is much more important than training data set. We used a simple data augmentation technique with the help of the OpenCV python library to try dealing with this issue which, in my opinion, is the difficult aspect of this project. But using a standard data augmentation or a pseudo labelling technique would have been more effective to solve this problem. On the other hand, the missing incidence angle on some training examples can be problematic. We have 133 / 1604 (8.29%) of examples that are missing incidence angle. Deciding whether to remove them or not could impact the modelling process since the size of the training data set is already restraint compared to the machine learning we use for this classification challenge. Even though the results are satisfying and the final model and solution fit our expectations, one of the next step will be to amend the loss. We can improve the loss by carefully focusing our efforts on optimizers via the stochastic gradient descent and his many parameters on which we can leverage. Based on that, we can use this solution in a general setting to solve these types of problems.

V.3) Improvements

In remote sensing applications, we usually only have a small amount of labelled data for training because they are expensive to collect, although we still have abundant unlabelled data. Semi-supervised deep learning uses limited labelled data and abundant unlabelled data to train a deep neural network. It uses the unlabeled data to gain more understanding of the population structure in general.

In this project, we learnt features only from a small training set and we did not take advantages of the test set that contains a lot of valuable information.

Pseudo labeling is a simple and efficient method to do semi-supervised learning. It combines almost all neural network models and learning methods by taking the model we used with our training data set that gave us good results and using it with our unlabeled data test set to predict the pseudo-labels. We will then concatenate the training labels with the test set pseudo labels as well as concatenate the features of the training set with the features of the test set and finally train the model in the same way we did before with the training set. This method would make the error decreases and would improve the model by better learning the general structure **[19]**.

VI. References

[1] https://www.kaggle.com/yuhaichina/single-model-vgg16-mobilenet-lb-0-1568-with-tf

[2] https://keras.io/getting-started/functional-api-guide/#multi-input-and-multi-output-model

[3] Gradient-Based Learning Applied to Document Recognition, Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, November 1998.

[4] ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

[5] Iceberg and ship detection and classification in single, dual and quad polarized synthetic aperture radar, Howell Carl (2008), Master thesis, Memorial University of Newfoundland.

[6] http://elib.dlr.de/99079/2/2016_BENTES_Frost_Velotto_Tings_EUSAR_FP.pdf

[7] https://www.kaggle.com/c/statoil-iceberg-classifier-challenge

[8] https://arxiv.org/pdf/1708.07920.pdf

[9] http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[10] http://neuralnetworksanddeeplearning.com/chap6.html

[11] http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html

[12]https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2fa82

[13] https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks

[14] http://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/

[15] https://en.wikipedia.org/wiki/Early_stopping

[16] https://www.kaggle.com/cbryant/keras-cnn-statoil-iceberg-lb-0-1995-now-0-1516

[17] Deep Learning, A Practitioner's Approach, Josh Patterson & Adam Gibson

[18] https://machinelearningmastery.com/check-point-deep-learning-models-keras

[19]https://towardsdatascience.com/simple-explanation-of-semi-supervised-learning-and-pseudo-labeling

[20] Automatic Target Detection in Satellite Images using Deep Learning, Muhammad Jaleed Khan, Adeel Yousaf, Shifa Nadeem, Khurram Khurshid, July 2017

[21] Transfer Learning with Deep Convolutional Neural Network for SAR Target Classification with Limited Labeled Data, Zhongling Huang, Zongxu Pan, Bin Lei.

[22] Chen, S.; Wang, H. SAR Target Recognition Based on Deep Learning. In Proceedings of the 2014 International Conference on Data Science and Advanced Analytics (DSAA), Shanghai, China, 30 October–1 November 2014; pp. 541–547.

[23] Chen, S.; Wang, H.; Xu, F.; Jin, Y.Q. Target Classification using the Deep Convolutional Networks for SAR Images. IEEE Trans. Geosci. Remote Sens. 2016, 54, 4806–4817.

[24] Wilmanski, M.; Kreucher, C.; Lauer, J. Modern Approaches in Deep Learning for SAR ATR. Int. Soc. Opt. Photonics 2016, 9843, 98430N–98430N.

[25] http://neuralnetworksanddeeplearning.com/chap6.html

[26] https://www.quora.com/What-are-the-advantages-and-disadvantages-of-deep-learning-Can-you-compare-it-with-the-statistical-learning-theory

[27] https://www.kaggle.com/muonneutrino/exploration-transforming-images-in-python

[28] cs231n.github.io/convolutional-networks/

[29] https://sentinel.esa.int/web/sentinel/missions/sentinel-1/instrument-payload