Data Science Report

Jeffrey Everett
CSCI 4802

December 19, 2017

Learning Goals

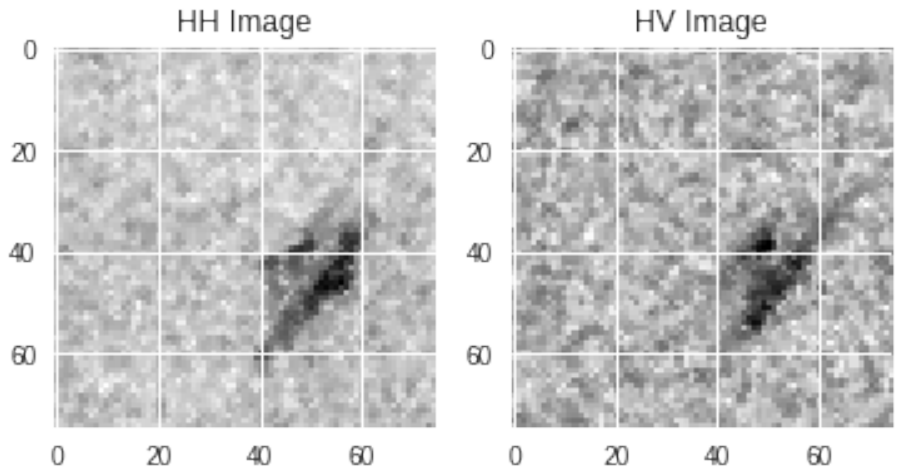
Since the midterm report, I have changed my goals to learning deep learning techniques. The resources I have used to do so are the Deep Learning book by Ian Goodfellow and the online lectures for the CS231n Stanford CNN class. Progress has not been as fast as I would have liked, but I believe I have learned the basics. The techniques I have learned were used on the Statoil Challenge, outlined below.

Statoil Challenge

The Statoil Challenge has been my main focus on the implementation side. In this section, I will introduce the task and then discuss the techniques I used to attempt to solve it.

Task

The basic premise of this challenge is binary classification between ships and icebergs using satellite-based radar data. This data is created by bouncing a signal off of an object and recording the amount of reflected data. These recorded reflections, or backscatter, can then be considered as an image, where large amounts of reflection create bright spots in the image and small amounts of reflection create dark spots. In this dataset, there are two such images for each classification: one in which the signals are both transmitted and received horizontally (HH), and one in which the signals are transmitted horizontally and received vertically (HV). There is also an accompanying incidence angle, which gives the angle with which the above is measured. This feature can play a fairly significant role in amount of backscatter produced. Below is an example of the data given for a prediction:



These images were collected at an incidence angle of 40.7192.

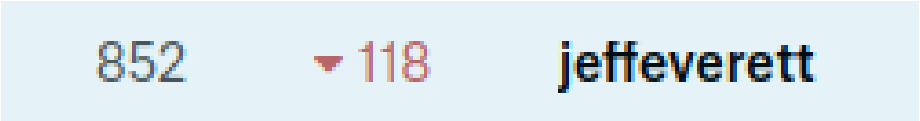
Methods Used

My first attempts were all based on the global statistics of the data. To begin, I calculated statistics about the images such as the minimum, maximum, first quartile, median, third quartile, etc. I then fed this data, along with the incidence angle, into first a logistic regression model and then a gradient-boosting classifier (the former using `sklearn` and the latter using `xgboost`). The results were predictably unimpressive: the logistic regression approach received a log-loss of 0.5178 and the gradient-boosting approach received a log-loss of 0.3702; these scores correspond to current-day leaderboard positions of 2225/2536 and 2089/2536, respectively.

To progress further, it was clear that I needed to incorporate the entirety of the given data. Because the competition involves images, the most obvious way to do so was using CNNs. The input to a CNN is usually a single image; I wasn't sure how to use both images at first, so I started out using just the HH image. The next step was constructing and training some CNN. I tried to use Caffe to do so at first, but I found it difficult to use and I didn't like the `protobuf` approach to declaring the models. I eventually migrated to `keras`, which I found much more enjoyable. However, my initial CNNs still performed poorly. My first major problem was that, although I knew some basic guidelines for how to create CNNs (convolutional layers before fully-connected layers, dropout layers to prevent overfitting, etc.), I didn't have the expertise to create an effective one. After experimenting with different neural network architectures to no avail, I decided that the problem might be that there wasn't enough data (there were 1604 examples in the training set). I performed some basic data augmentation using `kera`'s `ImageDataGenerator`–there were non-negligible improvements, but the impact wasn't enough to create an effective model.

I knew during this time about transfer learning; however, it seemed to me that the radar imaging used in the competition was too different from the ImageNet data (from which most of the available "pre-trained models" are derived) to be useful. Moroever, it was my belief (and still is) that using the well-known architectures like VGG16, GoogLeNet, ResNet, etc. without using the pre-trained weights as a starting point would lead to overfitting because of the vast differences between the number of parameters and the number of examples in this dataset. After all, most of these CNNs were constructed with the ImageNet dataset in mind. However, since nothing else was working, I decided to try to implement transfer learning.

The model from which I began is VGG16. This takes in images with three channels, and unfortunately the given images have a single channel. At first, I simply replicated the HH channel into the two unfilled channels. However, I eventually read Kaggle kernels using similar techniques, and they created a three-channel image by placing the HH data in one channel, the HV data in a second channel, and their per-pixel mean in a third channel; I eventually adopted this approach. Because of the discrepancy in image size, I removed the fully-connected layers and added two of my own. The result was much better than anticipated, with a log-loss of about 0.21. After tweaking some parameters and adding some features like early stopping, I was able to reduce this log-loss to its current value of 0.1774. This corresponds to a leaderboard position 852/2560, shown below:



Future Work

The most glaring issue in my current model is that it doesn't incorporate the incidence angle. This was the most significant predictor out of all the global statistics used in my earlier attempts, which suggests that it's a very important piece of data to include. Some possible ways to include this feature are as follows:

- Somehow manually include the information into the input image. This could likely be done by extending the image with replications of this data, ideally with enough padding such that the original image data isn't lost to this extension data by the convolutional and pooling layers.
- Extract the resulting features from the next-to-last layer, manually add the incidence angle to this feature list, and then train using a gradient-boosting based classifier.
- Combine the angle data directly with the pixel data. For example, adding some scaled version of the data point to each pixel (while clamping these pixels to a valid range).

I'm not sure how viable these ideas are, but a robust inclusion of the incidence angle would likely have a fairly significant effect on the log-loss.