Saving the Titanic

Introduction: I investigated the problem of classifying ships and icebergs from satellites in outer space in the Kaggle competition, “Statoil/C-CORE Iceberg Classifier Challenge”. To solve this problem, I used the Machine Learning Technique of Convolutional Neural Networks on the two types of radar images, HH and HV, which identified each training example.

Experimental Setup: The data I used from this project was a 1604 training example length data set from Kaggle which had 2 flattened radar images for each iceberg as well as an incidence angle for most training examples. I split my data up into 1400 training examples and a cross validation set of 204 examples. I had two metrics of assessment, accuracy and logarithmic loss. I chose logarithmic loss because it is the assessment method which Kaggle uses for its competition and I wanted to see if my algorithm would be competitive with the top entries but unfortunately in all my tests the best logarithmic loss I saw was .165 which I only achieved after overfitting the regularization parameters to the cross validation set (I did not end up using the dropout rate I had as the loss in the test was better than that in the last epoch of the training) and the current leaders loss is .1029.

In my data, the incidence angle refers to the angle at which the satellite was to the ship or iceberg when it took the radar scan. In order to account for the incomplete nature of my data, for every example with ‘na’ instead of an angle I tried setting those angles to -45, -10, 0, 10, 45 and by far the best hardcoded values were 0. Now, to process the two flattened radar images, I reshaped them from 1 by 5625 back to 75 by 75 pixels. The way these radar images differ is the orientation by which the satellite receives the radar signal. There are HV and HH images with the first H meaning a horizontally sent signal and the second letter referring to either horizontally or vertically received signals.
Once I had processed my data, I used Keras’ Convolutional Neural Network packages to create, train, and test a model. I fed each image through 3 layers of convolutional neural nets, using max pooling after each neural net. Initially I had a massive problem with overfitting but to resolve this issue I implemented a dropout after the first neural net. What I found with dropout was that there is a very small window for which the dropout will work properly, and I spent a long time finetuning my dropout rate. The acceptable range I found for my dropout rate was .685 to .725. I chose to use .715 because at this point my training accuracy would still be high and would be very similar to my cross-validation accuracy. Another form of regularization I used was early stopping because if I trained my Neural Net for enough Epochs I could always reach upwards of 99% training accuracy that wouldn’t be representative of the actual predictions. I intended to implement data augmentation but had trouble putting multiple inputs into Keras’ data generation package as the package only wanted 1 image to rotate as oppose to 2. After the image layers I flattened the output layers and merged them with each other and the incidence angle. Following this I used 1 dense layer and 1 sigmoid function dense layer to predict a classification.

**Results:**

The accuracy I was able to achieve after training my algorithm and not overfitting my regularization parameters to the cross validation data was 91.13%. and solving my regularization woes was: x%. In addition, the log loss in this test was: .2249.
Log Loss vs Epochs:

Accuracy vs Epochs:

**Conclusion:** My findings from this project are that convolutional neural nets are very convenient and efficient ways to process images as I was able to achieve a high classification accuracy with only a few weeks of work and as I was able to train my network within 10 minutes despite the plain CPU within my laptop. Further adding to this point, I found that the convolutional neural network is much faster than the full neural network as when I included a dense layer with 1000 nodes the program took far longer on just that 1 layer. Furthermore, I learned that machine learning is quite difficult as to just differentiate from the small number of pixels in each image which were either a ship or an iceberg I needed to finetune everything from the number of layers to the convergence function to the amount of regularization.

Link to my html: http://www.cs.middlebury.edu/~swestvold/cs451/fp/fp.html