Individual Final Report
Visual Recognition of Ships Using CNN

Shilpa Rajbhandari

Introduction:
We are using convolutional neural networks and caffe to create a program that can carry out visual recognition of ships at sea. Taking satellite images, sometimes with fog and clouds, we will check the accuracy and loss values as the program analyzes them.

After acquiring the data, we took a subset of it, reduced the pixel values, and uploaded the data and the various code files to Google Cloud Platform (GCP). Once we ran them, we were able to look at how the accuracy and loss values changed with each iteration. Based on this, we made some changes to our code and the values that we chose in order to get better results. We were also able to see when our program started to overfit the data and determined the cutoff for the number of iterations that we should use.

Description of the Data Set:
We acquired our data set from Kaggle (**https://www.kaggle.com/c/airbus-ship-detection**). The data consists of a series of satellite images of the ocean. In some of the photos, there are ships, in others only empty sea. Some of the photos also have clouds or haze, which add a complicating factor.

The dataset had 192,556 images. We thought that this was unnecessarily large for our analysis, so we pared it down to 50,000 images for training and 10,000 images for validation.

Description of the Algorithm:
There were multiple files of code that can be divided into three categories:

1. Pre-processing of the data.
2. Processing the data.
3. Support for the file that processes the data.

A Jupyter notebook (test_airbus_ship.pynb) was used to create the subset of images from the original dataset. These images are the ones that will be divided into the training and validation images, as described below. I worked closely with my group member in preprocessing, finding the right amount of training and validation of data and creating a different folder for our two class with ship an no ship. Because, for caffe we need to create a different folder with each class. After this, the images were uploaded to GCP, along with all of the other programs that we used.

The main code file that we used for analysis is train_airbus.py. This code is based on the use of Caffe and a Convolutional Neural Network. It runs through all of the images and creates a binary output based on whether or not its evaluation determines if a ship is present. It then compares this to the actual results, displaying the accuracy and loss values.

There were a variety of things that we tried before settling on our current set-up. We started off by resizing the images to 80x80 pixels and checking the results, which came off well. We then tried resizing to 32x32 pixels, but the accuracy results became worse. We also tried resizing the

images to a size larger than 80x80 pixels, but there were problems when we tried to run it, so we just stayed with 80x80 for all of our tests and final analysis.

There are then three main programs that are used to do the actual analysis:

1. Our training algorithm is create_lmdb.py. This program resizes the images, and then divides images into training and validation test sets. Our original images were 768x768 pixels in size, but we tried multiple different sizes. The one that we settled on was 80x80 pixels. This program was the one placed on Dr. Amir Jafari's GitHub page.
2. airbus_solver.protext sets the various parameters for the iterations: The number of iterations, the learning rate, etc. This program is called by the airbus_train_text.protext file.

test_iter	test_interval	base_lr	momentum	Weight_decay	lr_policy
100	500	0.01	0.9	0.0005	inv
gamma	**power**	**display**	**max_iter**	**solver_mode**	
0.0001	0.75	100	100	GPU	

3. airbus_train_text.protext lists the various layers that will be used by the program. This program is called by the airbus_solver.protext file.

Convolution 1	num_output	kernel_size	stride
	60	5	1
Convolution 2	**num_output**	**kernel_size**	**stride**
	120	5	1
Pooling 1	**kernel_size**	**stride**	
	5	5	
Pooling 2	**kernel_size**	**stride**	
	5	5	
For inner product	**num_output**		
	1000		

Once all of these files are created and set-up, we just need to run create_lmdb.py to properly set-up the data, and then run train_airbus.py to do the analysis.

Results:

I used the size of 32x32 pixels, with 20000 iterations for feature maps and kernel for analysis and accuracy. The accuracy is 0.29, thus to improve the accuracy, we resize the image into 80X80 pixels, and got the accuracy 0.48. Further, we resize image into 129X129 pixels, hoping for better result but our system got hung up. So, we stick into 80X80 pixels for the model.

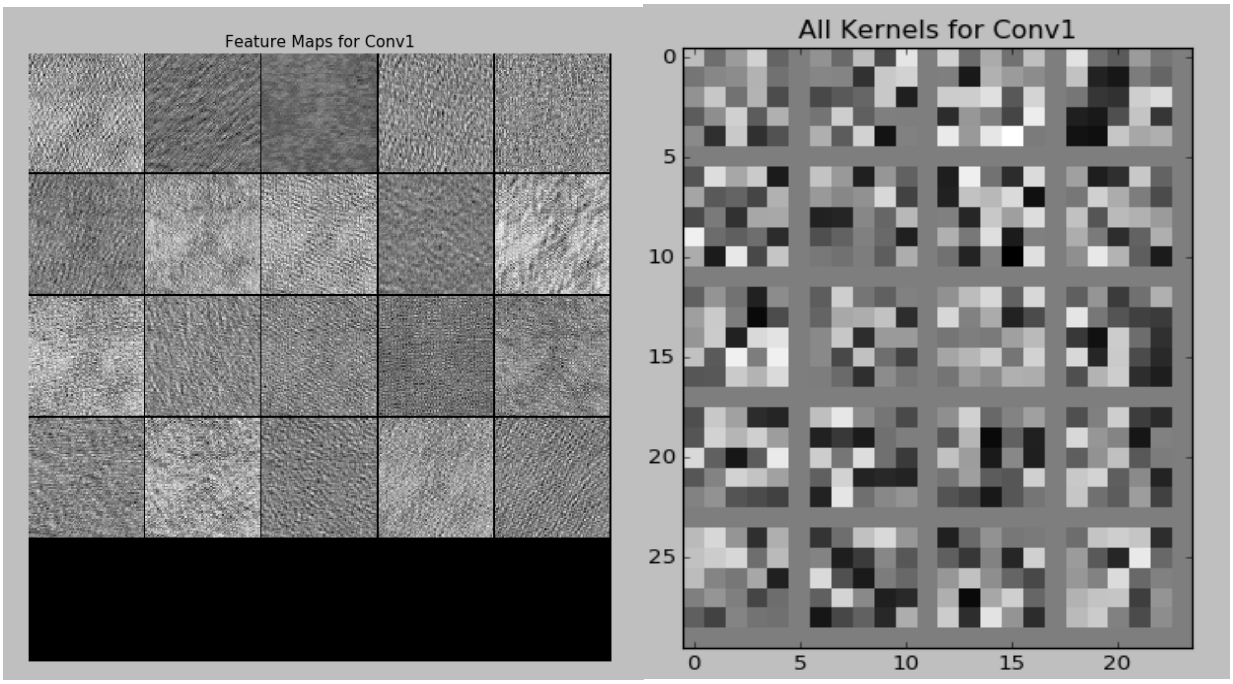

Figure 1 - Feature maps (Left) and Kernel (Right) for 32x32 pixel images.

```
Iteration 0 testing... accuracy: 0.0
I1209 11:40:14.834834  2359 solver.cpp:239] Iteration 100 (17.509 iter/s, 5.71136s/100 iters), loss = 0.664275
I1209 11:40:14.834915  2359 solver.cpp:258]     Train net output #0: loss = 0.664275 (* 1 = 0.664275 loss)
I1209 11:40:14.834928  2359 sgd_solver.cpp:112] Iteration 100, lr = 0.00992565
Iteration 100 testing... accuracy: 0.0
I1209 11:40:20.498921  2359 solver.cpp:239] Iteration 200 (17.6556 iter/s, 5.66394s/100 iters), loss = 0.498359
I1209 11:40:20.499007  2359 solver.cpp:258]     Train net output #0: loss = 0.498359 (* 1 = 0.498359 loss)
I1209 11:40:20.499032  2359 sgd_solver.cpp:112] Iteration 200, lr = 0.00985258
Iteration 200 testing... accuracy: 0.0
I1209 11:40:26.214808  2359 solver.cpp:239] Iteration 300 (17.4956 iter/s, 5.71573s/100 iters), loss = 0.480157
I1209 11:40:26.214905  2359 solver.cpp:258]     Train net output #0: loss = 0.480157 (* 1 = 0.480157 loss)
I1209 11:40:26.214916  2359 sgd_solver.cpp:112] Iteration 300, lr = 0.00978075
Iteration 300 testing... accuracy: 0.0
I1209 11:40:31.819653  2359 solver.cpp:239] Iteration 400 (17.8423 iter/s, 5.60465s/100 iters), loss = 0.525325
I1209 11:40:31.824090  2359 solver.cpp:258]     Train net output #0: loss = 0.525325 (* 1 = 0.525325 loss)
I1209 11:40:31.824117  2359 sgd_solver.cpp:112] Iteration 400, lr = 0.00971013
Iteration 400 testing... accuracy: 0.0
```

```
Iteration 19600 testing... accuracy: 0.26
I1209 12:00:26.419978  2359 solver.cpp:239] Iteration 19700 (17.7861 iter/s, 5.62236s/100 iters), loss = 0.00213971
I1209 12:00:26.420035  2359 solver.cpp:258]     Train net output #0: loss = 0.00213971 (* 1 = 0.00213971 loss)
I1209 12:00:26.420044  2359 sgd_solver.cpp:112] Iteration 19700, lr = 0.00442011
Iteration 19700 testing... accuracy: 0.28
I1209 12:00:32.067144  2359 solver.cpp:239] Iteration 19800 (17.7084 iter/s, 5.64703s/100 iters), loss = 0.00301393
I1209 12:00:32.067215  2359 solver.cpp:258]     Train net output #0: loss = 0.00301393 (* 1 = 0.00301393 loss)
I1209 12:00:32.067225  2359 sgd_solver.cpp:112] Iteration 19800, lr = 0.00440898
Iteration 19800 testing... accuracy: 0.32
I1209 12:00:37.713999  2359 solver.cpp:239] Iteration 19900 (17.7093 iter/s, 5.64674s/100 iters), loss = 0.00429516
I1209 12:00:37.714054  2359 solver.cpp:258]     Train net output #0: loss = 0.00429516 (* 1 = 0.00429516 loss)
I1209 12:00:37.714062  2359 sgd_solver.cpp:112] Iteration 19900, lr = 0.00439791
Iteration 19900 testing... accuracy: 0.29
```

Figure 2 - Accuracy for images that are 32x32 pixels (Top and bottom)

Conclusions:

This project was a good understanding of the topics (specially caffe) that I learned in this course. There were some challenges working on the new dataset of the Kaggle competition. The data size was huge 25 GB (192,556 images) and the most important is processing data and keeping in place to use for our model Caffe. It took us 3 hours to upload the training set of 50000 and 10000 validation set in the GCP. For every 100 iterations we used 100 images to validate the results. We used 64 iterations for the training set and 100 iterations for the testing set. Our accuracy rate is best (48%) when we convert the images to 80x80 pixels.

The most important thing I learnt in this project is overfitting. We wanted to check for overfitting and control by early stopping, for this we plot validation and training loss. We found that this happened after 5,000 iterations because the results start to decline and rise again, so we used this as our cutoff point. I learnt that to get better accuracy trial and error on layers, parameters, picture resolution will be helpful. It might also help to load a larger dataset, as the greater variety of images can help to train the program to better discriminate between oceans and ships.

Future work:
I would like to work on different framework, Pytorch, Tensorflow and Keras and see if we get better accuracy. I would also like to investigate more on architecture to help classify these images.

References:
https://github.com/amir-jafari/Deep-Learning/tree/master/Caffe_
https://www.kaggle.com/ezietsman/airbus-eda
http://cs231n.github.io/
http://hagan.okstate.edu/NNDesign.pdf