Introduction

Based on the Kaggle competition, Airbus ship detection, we decided to take the training images and labels to make our own ship detection problem; more specifically, a binary classification problem. The preprocessing was simple: we used the csv file from the Kaggle repository to count the number of ships in each image and converted the counts to ones and zeros. From here, took a sample of the images and divided the data into testing and validation sets (50,000 images vs 10,000 images). To make this work on Caffe, as can be seen in the preprocessing file, we created two directories training2 and validation2, each with subdirectories 00001 and 00000, with the images containing ship and no ships, respectively. Once this is done, we used the create_lmdd.py file from Prof. Amir Jafari's GitHub to create lmdb databases for Caffe. We then created text files for the solver and training/testing with a LeNet architecture. As for the team members roles, I was primary coder and additionally created a bucket on Google Cloud Storage to get access to the data. My team members supported me with ideas and creating the presentation slides and group project report. Additionally, they each had a chance to create new lmdb with varying image sizes to experiment with the network.

Individual Work

Mentioned previously, I wrote all the scripts, borrowing much of the code from Prof. Jafari's GitHub Caffe_ repository. Since we had much success with the MNIST dataset in Caffe, I thought that a network like this could also be used on our dataset. That said, since the images were resized to 80 by 80 pixels, I tried to keep the proportions close to those the MINST 32 by 32 pixel images: In particular, the pooling kernel size increase to five and the feature maps were 60 and 120 for the two convolutional networks. I tried tuning some parameters, such as lr_policy, weight_filler and trasnform_param, but none of my choices improved the algorithm. In addition, I wrote a loss layer for the validation set to keep track of the performance of algorithm. This was designed to search for overfitting graphically.

Results

Visually, we detected overfitting at 5000 iterations and chose early stoppage at this point to prevent overfitting. Since the Caffe documentation is not clear about evaluation metrics, we chose accuracy and we averaged around 50 percent accuracy.

Conclusions

Some improvements that can be made are preprocessing and image resizing. I believe this is the key to making these ships clearer in the images and lead to a better classifier. Another thing to consider, since many people on Kaggle used this method, is pre-trained weights and biases. This is not my first choice because you cannot tell how well you understand the network with somebody else's model.

Code Percentage

Preprocessing.py = 20/(20+73) * 100 = 21.5

Create_lmdb.py = (155-2)/155 * 100 = 98.7 (Based on Prof. Jafari create_lmdb_tutorial.py)
Train_airbus.py = (90-10)/90. * 100 = 88.88 (Based on Prof. Jafari train_mnist.py)
Airbus_solver.prototxt = (27-1)/27 * 100 = 96.30 (Based on Prof. Jafari lenet_solver.prototxt)
Airbus_train_test.prototxt = (179-10)/(179+10) * 100 = 89.42 (Based on Prof. Jafari lenet_train_test.prototxt)
Get_data.py = 0/3 * 100 = 0

References

Dr. Amir Jafari: https://github.com/amir-jafari
Kaggle: https://www.kaggle.com/c/airbus-ship-detection
Airbus Ship Detection: Data Visualization: https://www.kaggle.com/meaninglesslives/airbus-ship-detection-data-visualization