Visual Recognition of Ships Using CNN (Individual Report)

By Zachary Stein

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.

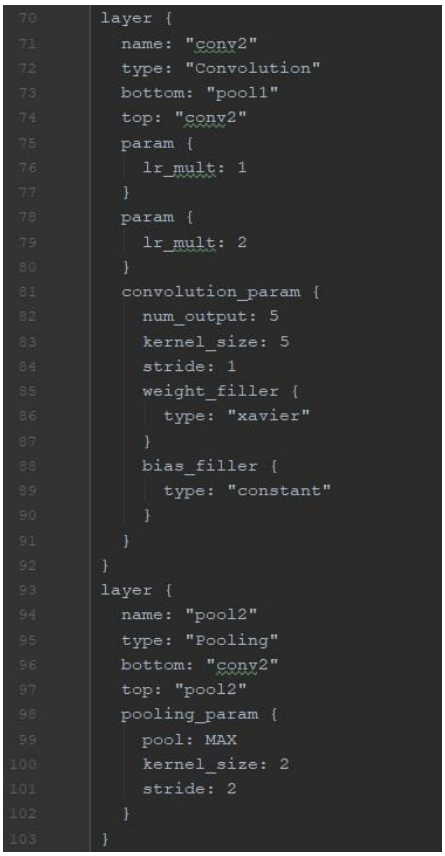
Description of Individual Work:

One of the main parts that I did was helping to set-up the code for resizing the images. We ended up using a standard size of 80x80 pixels, but before that we experimented with how the program ran with different sizes. I specifically tested changing the images to a size of 192x192 pixels. With the larger size, it took a lot longer for each iteration to run, and we judged that it would be impractical to use a larger size.

```
# 2. if you are using python3, you need to
# change all print function's format

# 3. Put the path that you want to write lmdb files into
# !!!Do not put exist folder here because it will delete these folder every time you run the script
train_lmdb = 'train_lmdb'
validation_lmdb = 'validation_lmdb'

# 4. Put training jpg images path here
JPG_train_path = 'train'
# Put validation jpg images path here
JPG_validation_path = 'validation'

# 4. Put the size you want your images resize to
IMAGE_WIDTH = 192
IMAGE_HEIGHT = 192

# You are ready to run this script now
# ================================================================================
def transform_img(img, img_width, img_height):

```

The part of the code where I changed the image size.

I also tried increasing the kernel size for the second convolution and pooling layer in lenet_train_test.prototxt, to see if that would help the running time. However, this greatly increased the loss value, and I stopped it.



```
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 5
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

The part of the code where I changed the kernel size.

For the rest of the project, I worked on putting together various parts of the code. One of the other parts that I was heavily involved with was pre-processing the data. We had to choose the right size for our data, which was a balancing act between too large and too small. If the data set is too small, then we won't get a good analysis with our train/test set. Too large, and it will take too long to upload the data to GCP and run. This was done with the preprocess.py file.

```
val_df_lab_1 = val_df[val_df.label==1]
val_df_lab_0 = val_df[val_df.label==0]

#--------------------------- Prepare folders for Caffe -------------------------------------

# train folder with label 1 (images with ships)
for file in train_df_lab_1.ImageID.tolist():
    try:
        os.rename("../ml2_final/train_v2/" + file, "../ml2_final/train/00001/" + file)
    except:
        print('No')

# train folder with label 0 (images with no ships)
for file in train_df_lab_0.ImageID.tolist():
    try:
        os.rename("../ml2_final/train_v2/" + file, "../ml2_final/train/00000/" + file)
    except:
        print('No')

# validation folder with label 1 (images with ships)
for file in val_df_lab_1.ImageID.tolist():
    try:
        os.rename("../ml2_final/train_v2/" + file, "../ml2_final/validation/00001/" + file)
    except:
        print('No')
# validation folder with label 0 (images with no ships)
for file in val_df_lab_0.ImageID.tolist():
    try:
        os.rename("../ml2_final/train_v2/" + file, "../ml2_final/validation/00000/" + file)
    except:
        print("No")
```

Code for splitting the data for preprocessing.

I also spent some time working on the layers of the code, making sure that the ones that we had set-up were working properly, and testing if we needed to make any changes to it. In the end, the changes that we made were pretty minimal, other than adding a layer to test for validation.

Results:

The results that were produced were for the images after they were rendered into a size of 32x32 pixels. We used the following feature maps and kernels for the analysis:

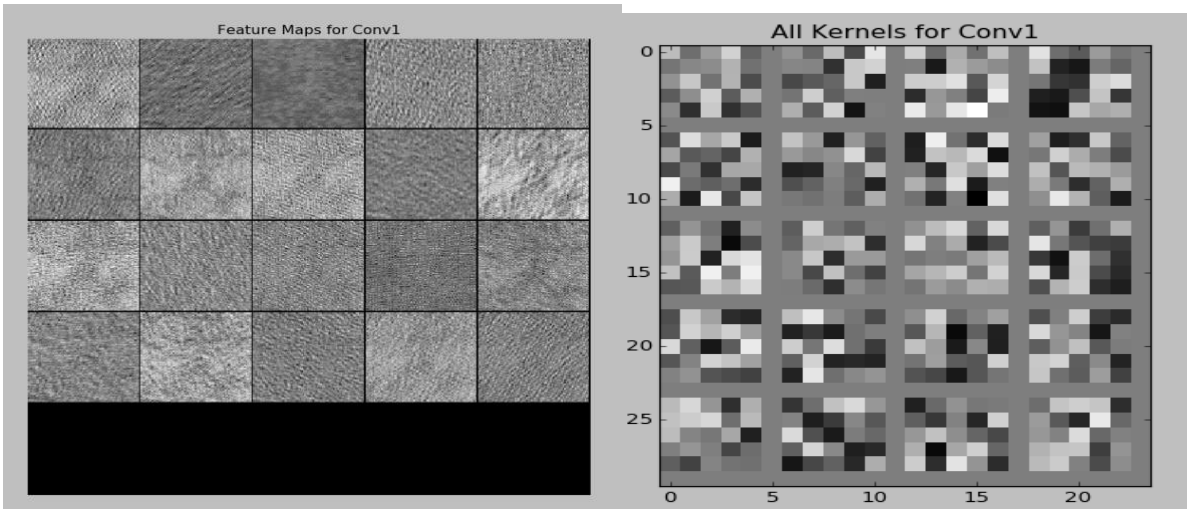


Figure 2 - Feature maps (Left) and Kernel (Right) for 32x32 pixel images.

Then, we started with 20,000 iterations for our program and checked the results. This was done for images that were 32x32 pixels and images that were 80x80 pixels.

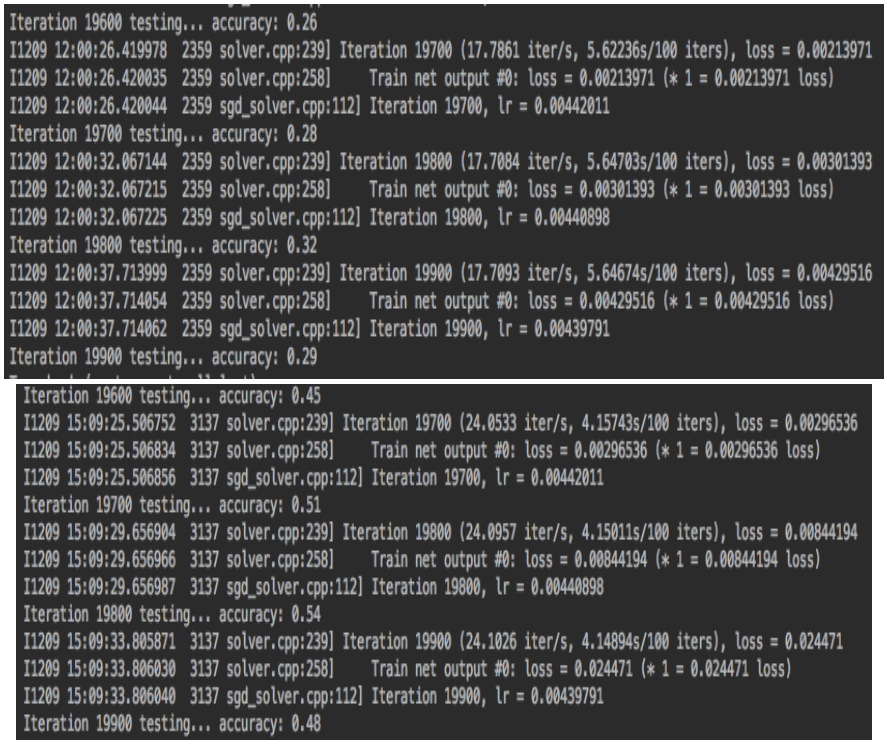


Figure 3 - Accuracy for images that are 32x32 pixels (Top) and 80x80 pixels (Bottom).

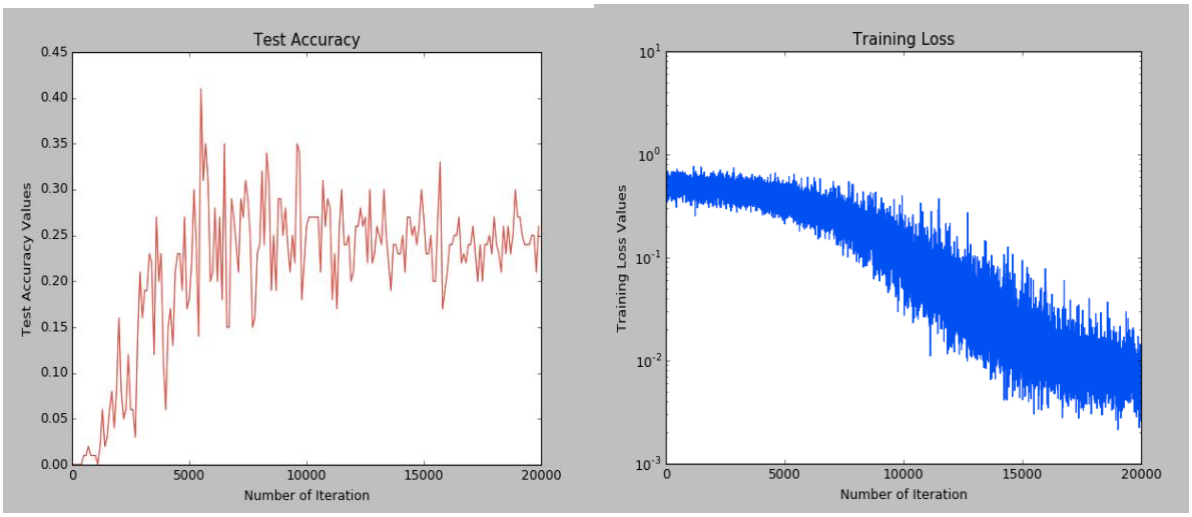


Figure 4 - Accuracy (Left) and loss (Right) for 32x32 pixel images.

After getting these results, we wanted to check the limitations of our data analysis. The first thing that we did was test the validation and testing loss over the number of iterations.

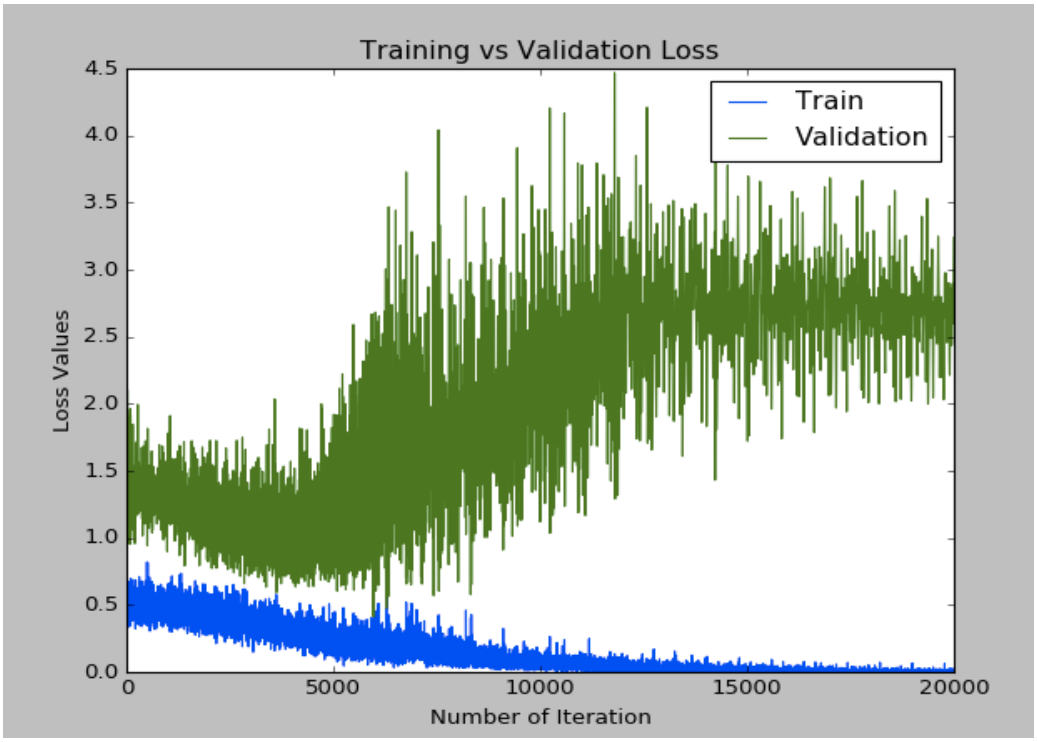


Figure 5 - Validation (Green) and Testing (Blue) Loss

Next, we checked for overfitting. We found that this happened after 5,000 iterations, so we used this as our cutoff point.

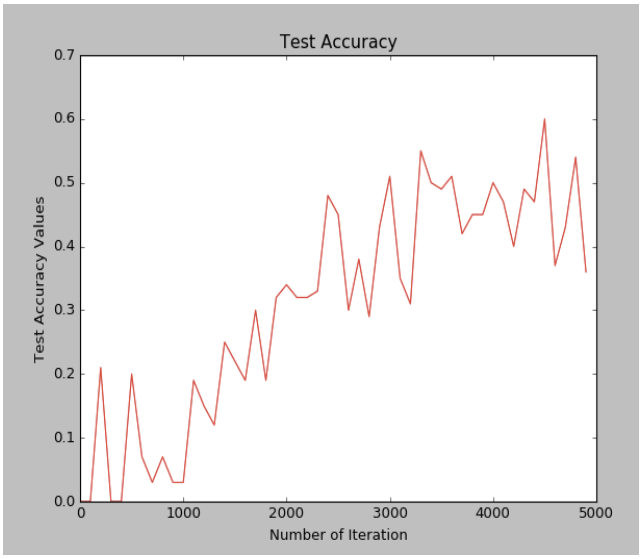


Figure 6 - Early Stopping

Summary and Conclusions:

These results show that the results for our data analysis are best when we convert the images to 80x80 pixels. At his resolution, we get an accuracy rating of around 50%, though one we are past

5,000 iterations we start to get overfitting. This means that we can successfully identify satellite images with ships with an accuracy equal to a coin flip.

Through our work with the code, we have learned that our results are very sensitive to the resolution of an image, and that there is such a thing as too much resolution and too little resolution. We have also found that the number of convolution layers and the parameters we choose for them have an effect on our results.

If we want to improve this, we need to work to find the ideal resolution, layers, and parameters. This is a delicate act, which requires a great deal of trial and error to learn what will work with the images. 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.

Percentage of the Code:
((112 -5)/(112+15))*100 = 91.45%

References:
Data - https://www.kaggle.com/c/airbus-ship-detection
Code - https://github.com/amir-jafari/Deep-Learning/tree/master/Caffe_