Airbus Ship Detection

Christopher Broll
Zachary Stein
Shilpa Rajbhandari

Introduction

- Airbus is a aerospace defense company based in Europe, posted a challenge to build a model to classify ships in satellite imagery.

- The model will be useful to prevent privacy, illegal fishing, drug trafficking, illegal cargo movement. It will be beneficial to insurance companies and environmental agencies as well.

Data Source and Goals

- Kaggle competition (https://www.kaggle.com/c/airbus-ship-detection)

- Our objective is to classify ships in satellite imagery.

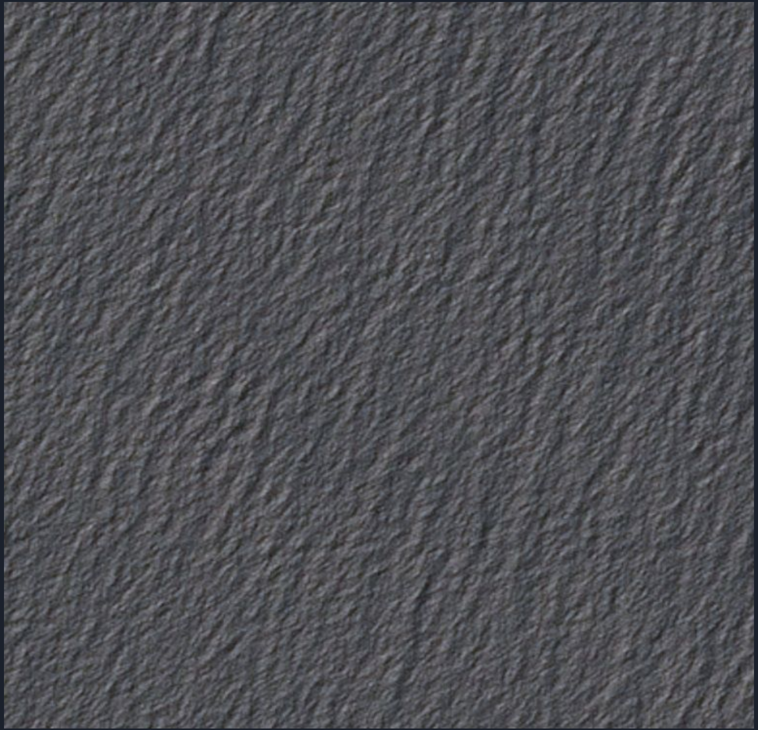
Project Data

- ImageId

- EncodedPixels

	ImageId	EncodedPixels
0	00003e153.jpg	NaN
1	0001124c7.jpg	NaN
2	000155de5.jpg	264661 17 265429 33 266197 33 266965 33 267733...
3	000194a2d.jpg	360486 1 361252 4 362019 5 362785 8 363552 10 ...
4	000194a2d.jpg	51834 9 52602 9 53370 9 54138 9 54906 9 55674 ...

Image with and without ship



Preprocessing

- 50000 images for training and 10000 for validation out of 192556
- Reduce the size of the image to 32 X32 for first test
- 80X80 for second test
- Create Lmdb

```python
# Get subset of data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_train, y_train, test_size=0.3, stratify=y_train)

# Split into training and validation image sets (50,000 and 10,000, respectively)
train_df = pd.DataFrame({"ImageID": X_train2[:50000], "label": y_train2[:50000]})
val_df = pd.DataFrame({"ImageID": X_test2[:10000], "label": y_test2[:10000]})

# Seperate labels for subfolders for Caffe CNN
train_df_lab_1 = train_df[train_df.label==1]
train_df_lab_0 = train_df[train_df.label==0]

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

```python
# 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')
```

```python
# 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")
```

Histogram Equalization

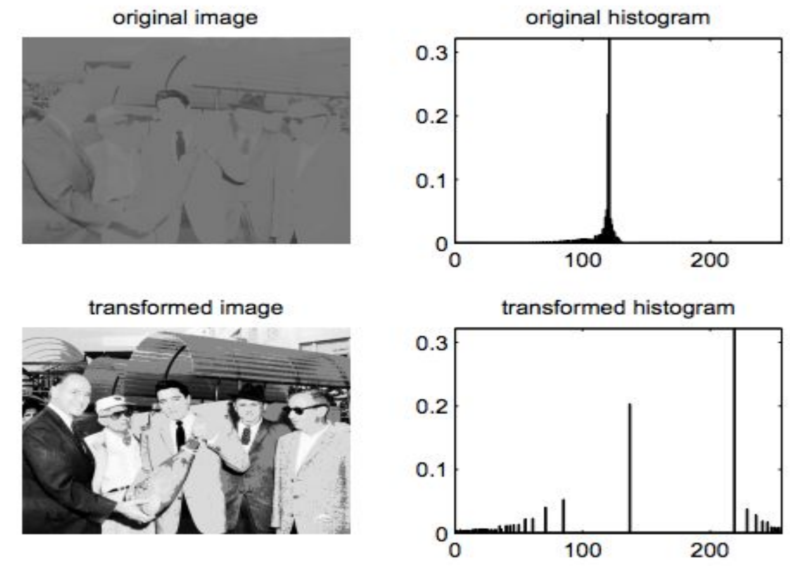


Figure 1: Histogram equalization applied to low contrast image

- Adjusting image for highlighting the feature

source:https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf

Caffe

- Implement network in convolution
- Caffe position layer from bottom to top

Caffe

```
// The learning rate decay policy. The currently implemented learning rate
// policies are as follows:
//    - fixed: always return base_lr.
//    - step: return base_lr * gamma ^ (floor(iter / step))
//    - exp: return base_lr * gamma ^ iter
//    - inv: return base_lr * (1 + gamma * iter) ^ (- power)
//    - multistep: similar to step but it allows non uniform steps defined by
//      stepvalue
//    - poly: the effective learning rate follows a polynomial decay, to be
//      zero by the max_iter. return base_lr (1 - iter/max_iter) ^ (power)
//    - sigmoid: the effective learning rate follows a sigmod decay
//      return base_lr ( 1/(1 + exp(-gamma * (iter - stepsize))))
//
// where base_lr, max_iter, gamma, step, stepvalue and power are defined
// in the solver parameter protocol buffer, and iter is the current iteration.
```

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Architecture file

Airbus_train_test.prototxt

For Convolution 1::

- num_output=60
- kernel_size=5
- stride=1

For Convolution 2:

- num_output=120
- kernel_size=5
- stride=1

For Pooling 1:

- Kernel_size: 5
- Stride=5

For Pooling 2:

- Kernel_size=5
- Stride=5

For inner product

- num_output =1000

Relu

Softmax with loss

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
```

```
layer {
  name: "loss_val"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss_val"
  include {
    phase: TEST
  }
```

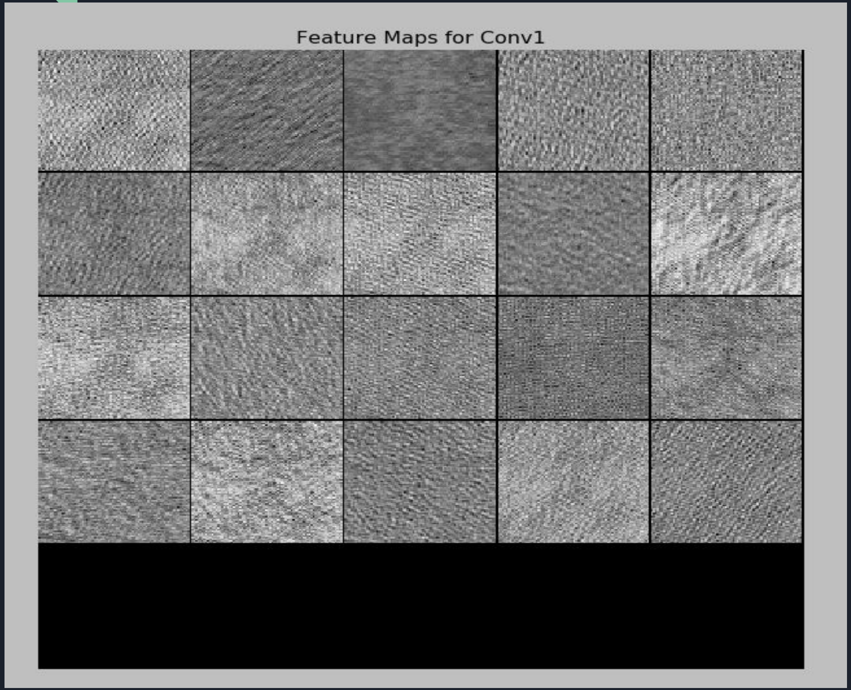
```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

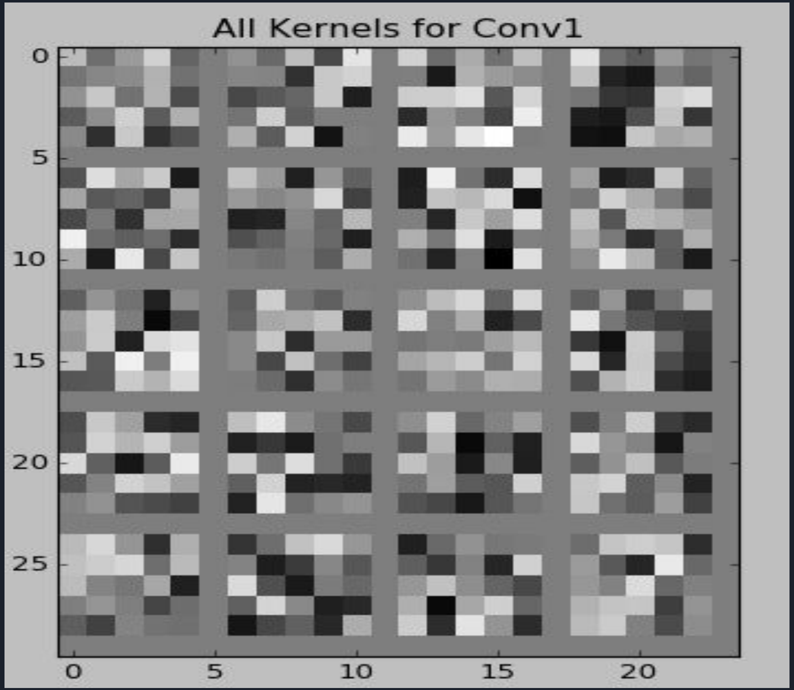
Solver file

```
# File for training and testing
net: "airbus_train_test.prototxt"

# 100 test iteration of a batchsize of 100 for 10,000 testing images
test_iter: 100

# test the network every 500 iterations
test_interval: 500

# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
```

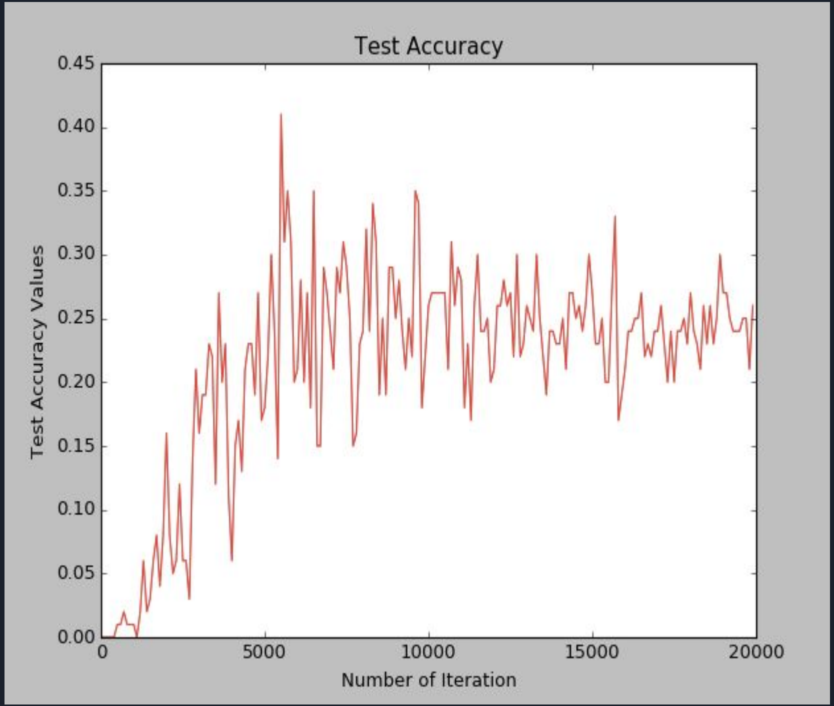
```
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75

# Display every 100 iterations
display: 100

# The maximum number of iterations
max_iter: 100

# Run on GPU (put CPU if not GPU available)
solver_mode: GPU
```

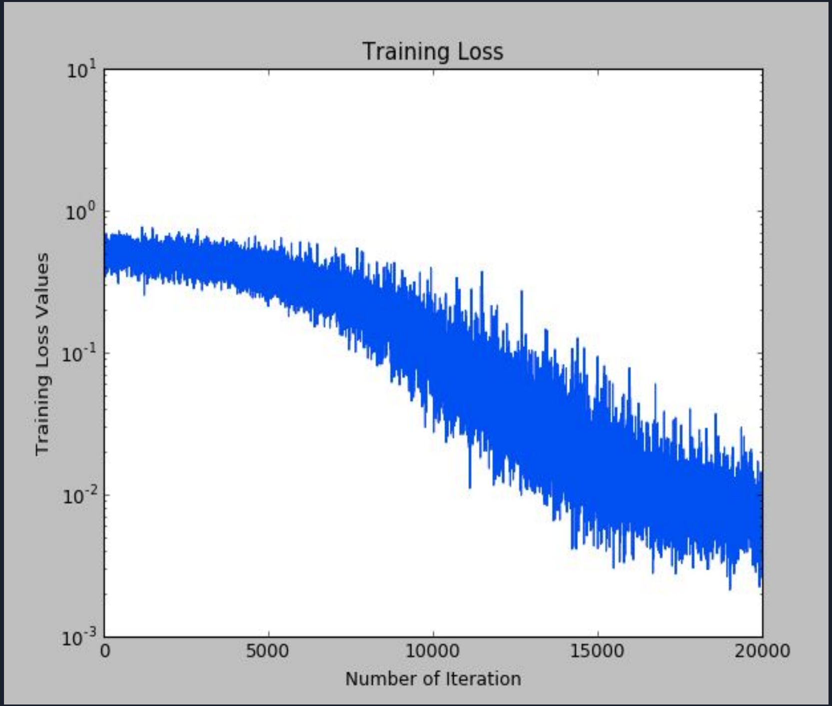
Feature map and kernel for 32X32 pixel





Accuracy and Loss for 32X32 pixels





Accuracy for 32X32 pixels (begin)

```
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
```

Accuracy for 32X32 pixels (end)

```
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
```

Accuracy for 80X80 pixels (begin)

```
Iteration 0 testing... accuracy: 0.0
I1209 14:54:50.121146  3137 solver.cpp:239] Iteration 100 (24.4522 iter/s, 4.08961s/100 iters), loss = 0.655891
I1209 14:54:50.121218  3137 solver.cpp:258]     Train net output #0: loss = 0.655891 (* 1 = 0.655891 loss)
I1209 14:54:50.121229  3137 sgd_solver.cpp:112] Iteration 100, lr = 0.00992565
Iteration 100 testing... accuracy: 0.0
I1209 14:54:54.230852  3137 solver.cpp:239] Iteration 200 (24.3329 iter/s, 4.10966s/100 iters), loss = 0.439718
I1209 14:54:54.230926  3137 solver.cpp:258]     Train net output #0: loss = 0.439718 (* 1 = 0.439718 loss)
I1209 14:54:54.230938  3137 sgd_solver.cpp:112] Iteration 200, lr = 0.00985258
Iteration 200 testing... accuracy: 0.1
I1209 14:54:58.346338  3137 solver.cpp:239] Iteration 300 (24.2987 iter/s, 4.11544s/100 iters), loss = 0.47
I1209 14:54:58.346408  3137 solver.cpp:258]     Train net output #0: loss = 0.47 (* 1 = 0.47 loss)
I1209 14:54:58.346433  3137 sgd_solver.cpp:112] Iteration 300, lr = 0.00978075
```

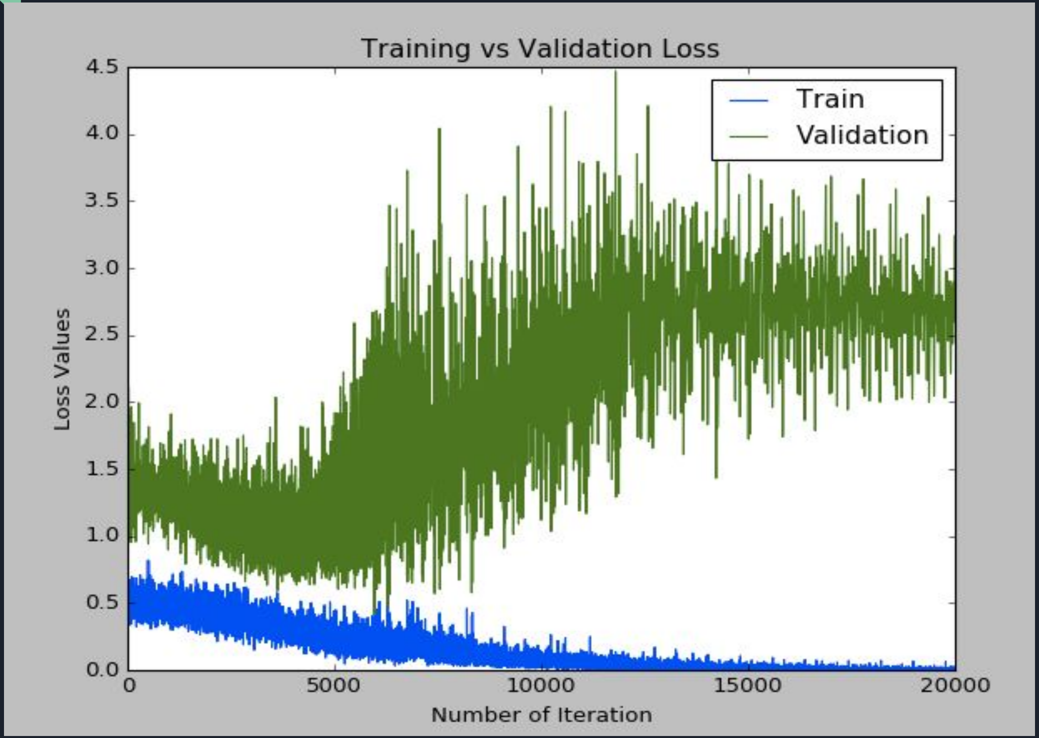
Accuracy for 80X80 pixels (mid)

```
Iteration 10000 testing... accuracy: 0.6
I1209 15:02:17.156886  3137 solver.cpp:239] Iteration 10100 (24.1427 iter/s, 4.14204s/100 iters), loss = 0.0467444
I1209 15:02:17.156965  3137 solver.cpp:258]     Train net output #0: loss = 0.0467444 (* 1 = 0.0467444 loss)
I1209 15:02:17.156975  3137 sgd_solver.cpp:112] Iteration 10100, lr = 0.00592384
Iteration 10100 testing... accuracy: 0.45
I1209 15:02:19.340059  3154 data_layer.cpp:73] Restarting data prefetching from start.
I1209 15:02:21.308784  3137 solver.cpp:239] Iteration 10200 (24.0856 iter/s, 4.15186s/100 iters), loss = 0.0638874
I1209 15:02:21.308858  3137 solver.cpp:258]     Train net output #0: loss = 0.0638874 (* 1 = 0.0638874 loss)
I1209 15:02:21.308887  3137 sgd_solver.cpp:112] Iteration 10200, lr = 0.00590183
Iteration 10200 testing... accuracy: 0.46
I1209 15:02:25.459769  3137 solver.cpp:239] Iteration 10300 (24.0909 iter/s, 4.15094s/100 iters), loss = 0.124397
I1209 15:02:25.459861  3137 solver.cpp:258]     Train net output #0: loss = 0.124397 (* 1 = 0.124397 loss)
I1209 15:02:25.459870  3137 sgd_solver.cpp:112] Iteration 10300, lr = 0.00588001
Iteration 10300 testing... accuracy: 0.64
```

Accuracy for 80X80 pixels (end)

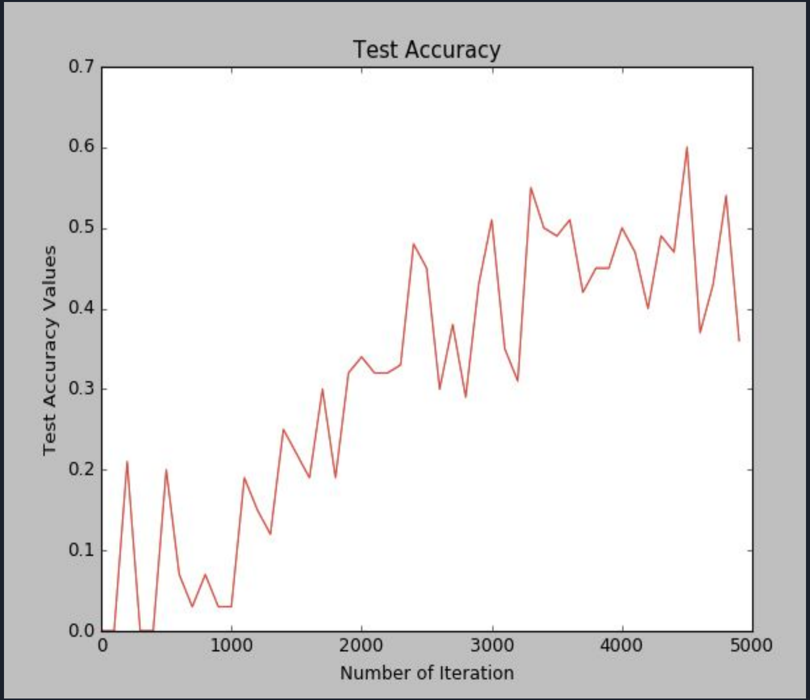
```
Iteration 19600 testing... accuracy: 0.45
I1209 15:09:25.506752  3137 solver.cpp:239] Iteration 19700 (24.0533 iter/s, 4.15743s/100 iters), loss = 0.00296536
I1209 15:09:25.506834  3137 solver.cpp:258]     Train net output #0: loss = 0.00296536 (* 1 = 0.00296536 loss)
I1209 15:09:25.506856  3137 sgd_solver.cpp:112] Iteration 19700, lr = 0.00442011
Iteration 19700 testing... accuracy: 0.51
I1209 15:09:29.656904  3137 solver.cpp:239] Iteration 19800 (24.0957 iter/s, 4.15011s/100 iters), loss = 0.00844194
I1209 15:09:29.656966  3137 solver.cpp:258]     Train net output #0: loss = 0.00844194 (* 1 = 0.00844194 loss)
I1209 15:09:29.656987  3137 sgd_solver.cpp:112] Iteration 19800, lr = 0.00440898
Iteration 19800 testing... accuracy: 0.54
I1209 15:09:33.805871  3137 solver.cpp:239] Iteration 19900 (24.1026 iter/s, 4.14894s/100 iters), loss = 0.024471
I1209 15:09:33.806030  3137 solver.cpp:258]     Train net output #0: loss = 0.024471 (* 1 = 0.024471 loss)
I1209 15:09:33.806040  3137 sgd_solver.cpp:112] Iteration 19900, lr = 0.00439791
Iteration 19900 testing... accuracy: 0.48
```

Validation and Testing loss



- Green line is the validation loss
- Blue line is the testing loss, after 5000 the loss is stagnant.

Early stopping



- To prevent overfitting we did early stopping on 5000 iteration

Conclusions

- Our model gives the best accuracy (around 50%) in 80X80 pixels.
- After 5000 iterations, we start to get overfitting.
- The number of convolution layers and parameters has an effect on our result.
- For future action, we can perform trial and error for image resolution, and focus on layer and parameter for better accuracy.
- Using more computational power to train with a larger number of images might help to improve classification.