Assignment 2 report
CS 381V Visual Recognition
William Xie

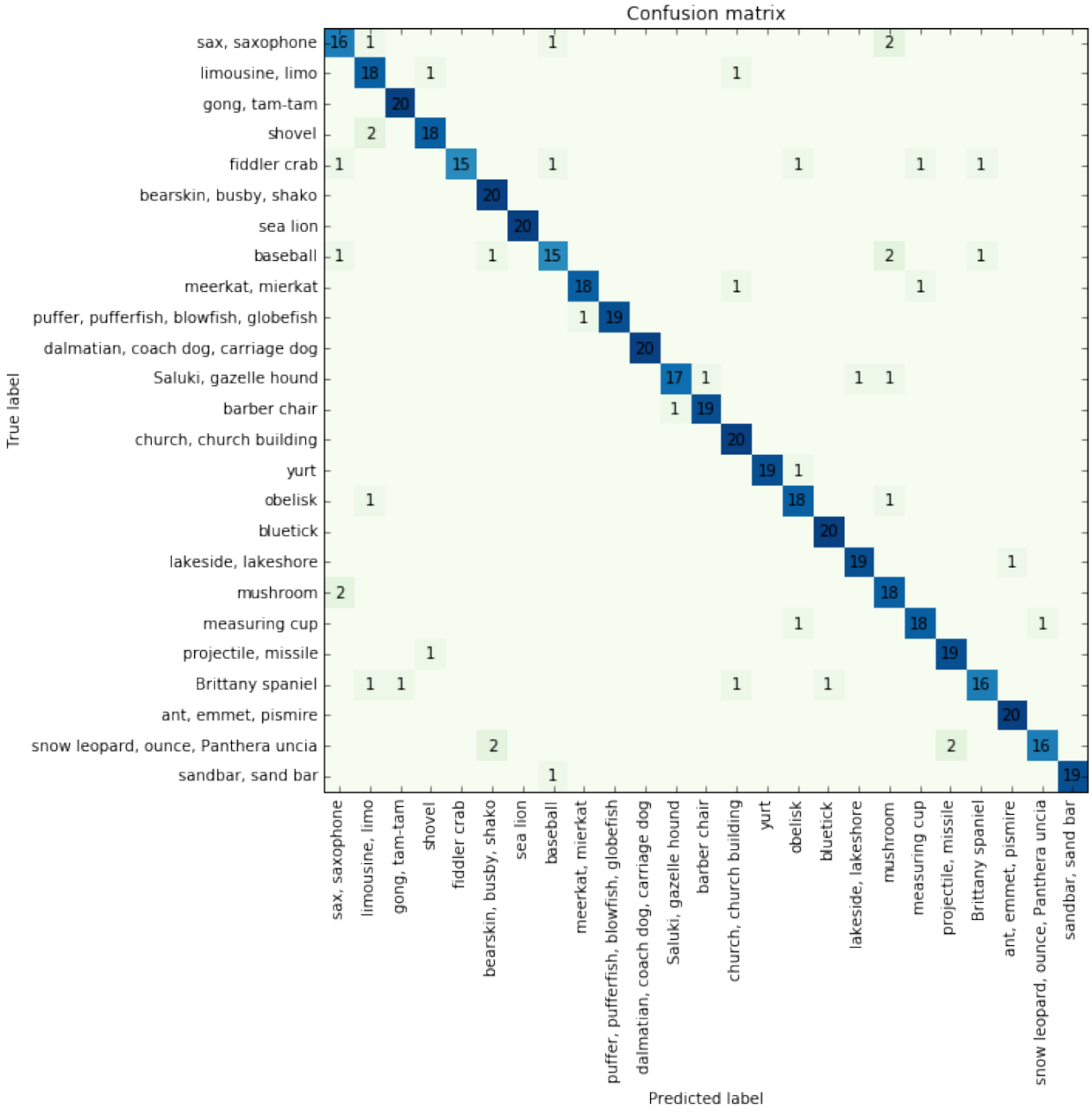
The goal of this assignment is to implement an image recognition system for classifying a 25-category image set. For training, we are allocated 100 images for each of the classes. Our validation set is consisting of 20 images per class. With the limited amount of data at hand, I decided to fine-tune a convolutional neural network (CNN) descriptor. As our readings suggest, this strategy is very viable for transfer learning when our domain data is very small.

The implementation is done in Caffe. The train and validation set images are extracted, resized, shuffled, and compiled into LMDB format for faster access. The chosen network is AlexNet with the last layer modified to to output 25 probabilities, one for each class. While more sophisticated models are available, AlexNet has been used as a benchmark in the vision community. I hope that through these experiments I can gain more insight to how to properly tune a CNN. For this purpose, I find AlexNet to be a good model to learn but perhaps not necessarily for achieving the highest performance. Caffe offers ImageNet pretrained weights for CaffeNet, a variant of AlexNet. It is used as a starting point for all the experiments I have conducted. After some experimentation, I modified some parameters such as bias and set the training batch size to 128. The learning rate is dropped periodically in steps. The step interval is set to 10,000 per drop in magnitude. In the default training set of 2500 images, it takes approximately 20 training iterations per epoch. For the extended training set (train + extra ~ 30,000 image), it takes approximately 230 iterations per epoch. Using the K40 GPU accelerated Maverick cluster, I was able to speed up the training process and run multiple jobs in parallel.

The below are some results of the different base learning rates (ir), configurations for fine-tuning, and size of train sets. The default fine tuning configuration is slow learning for all layers except the last layer (fc8), which has 10 times the learning rate as the rest of the layers.

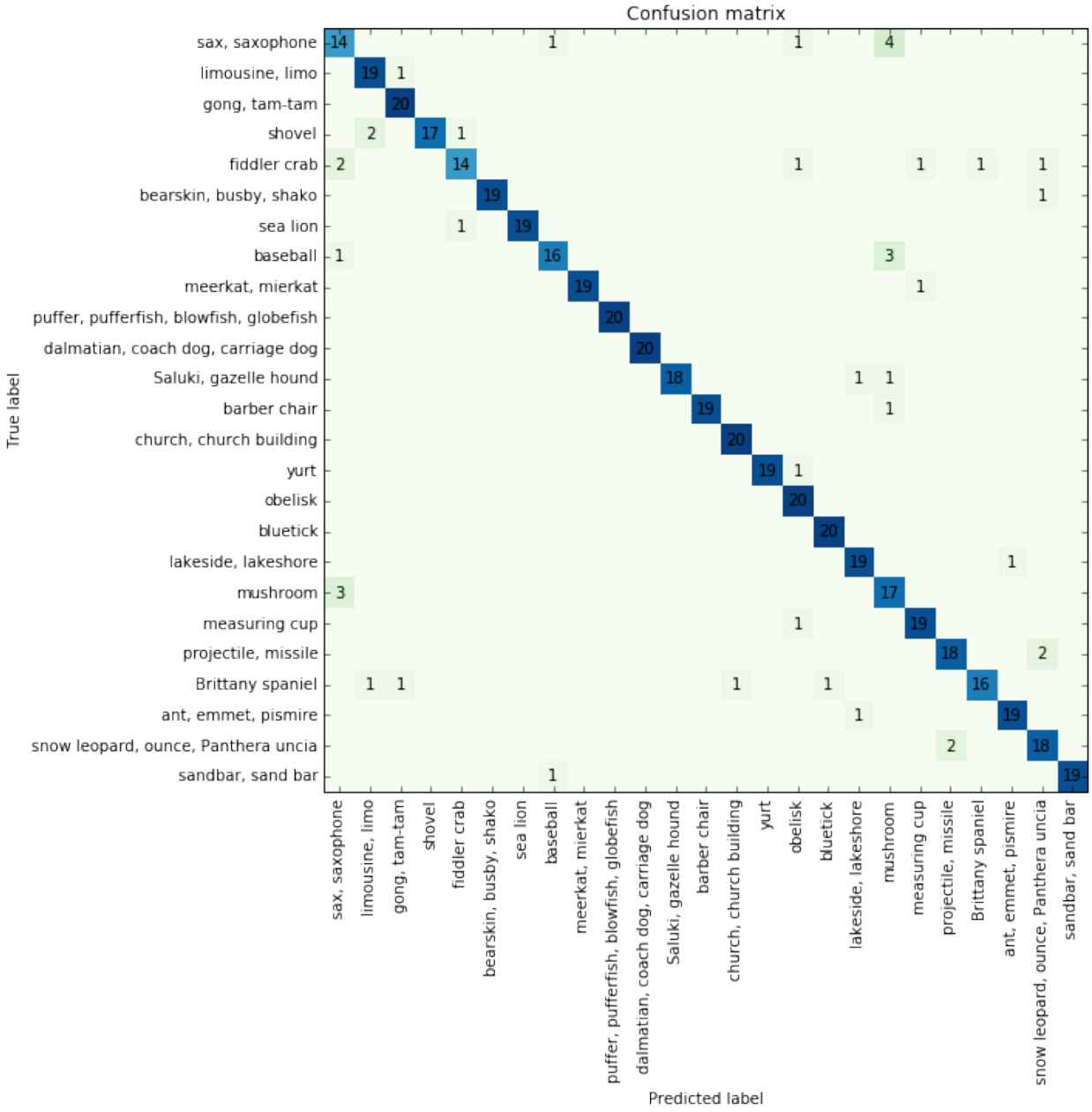
Training size: 2500; ir: 0.000001; fine-tune range: all layers
Mean class accuracy: 0.914

Using a very conservative learning rate and the default small training set, the network is able to gradually learn and achieve over 90% accuracy. This can be used as a rough baseline on what to expect for minor fine tuning.



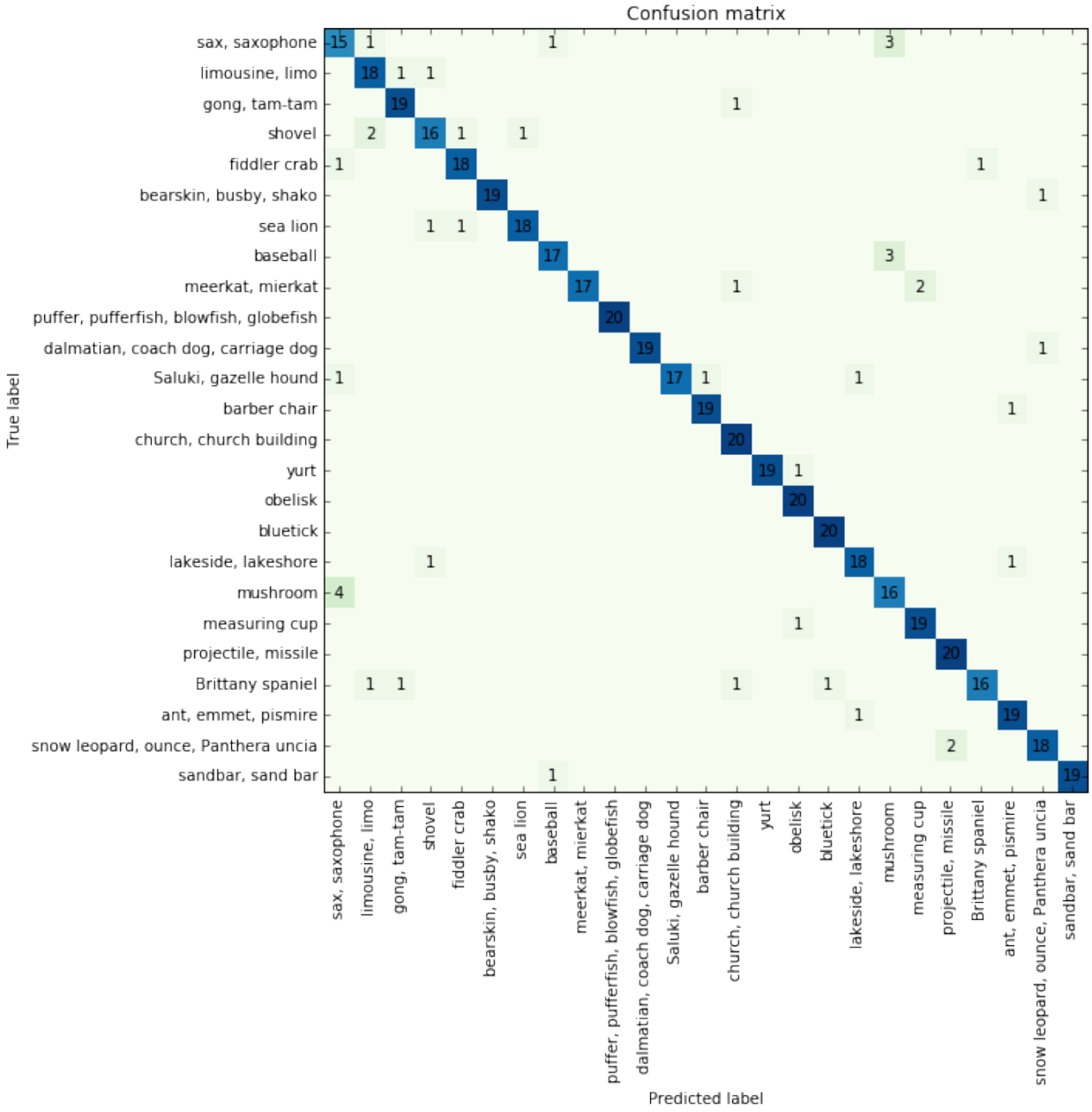
Training size: 31409; ir: 0.000001; fine-tune range: all layers;
Mean class accuracy: 0.916

Similar setup but now with more images to train (train + extra sets). This also results in more epochs in more iterations.



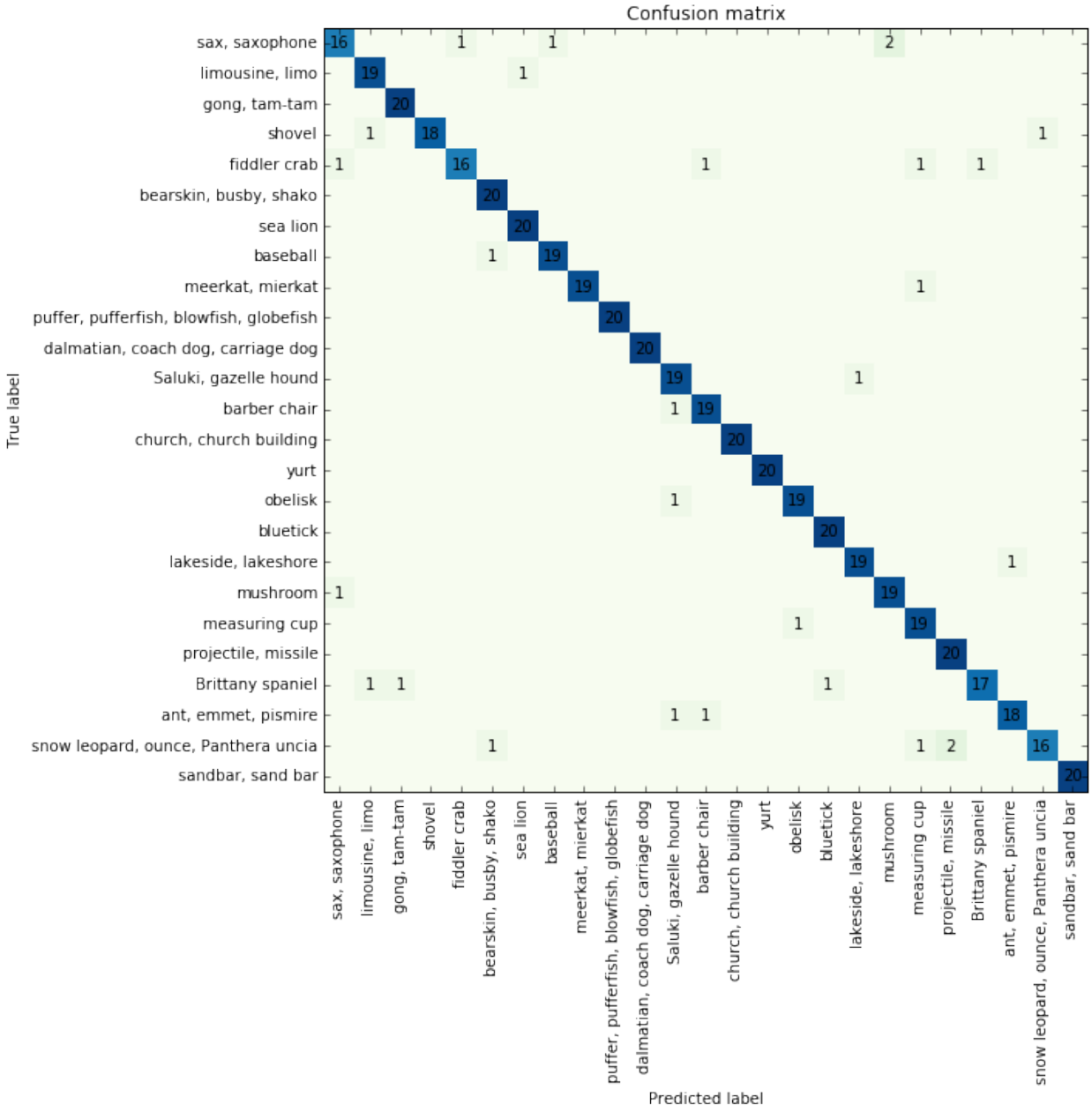
Training size: 31409; ir: 0.000001; fine-tune range: fc layers only;
Mean class accuracy: 0.914

Now we freeze all the conv layers and only fine-tune on fully connected layers.

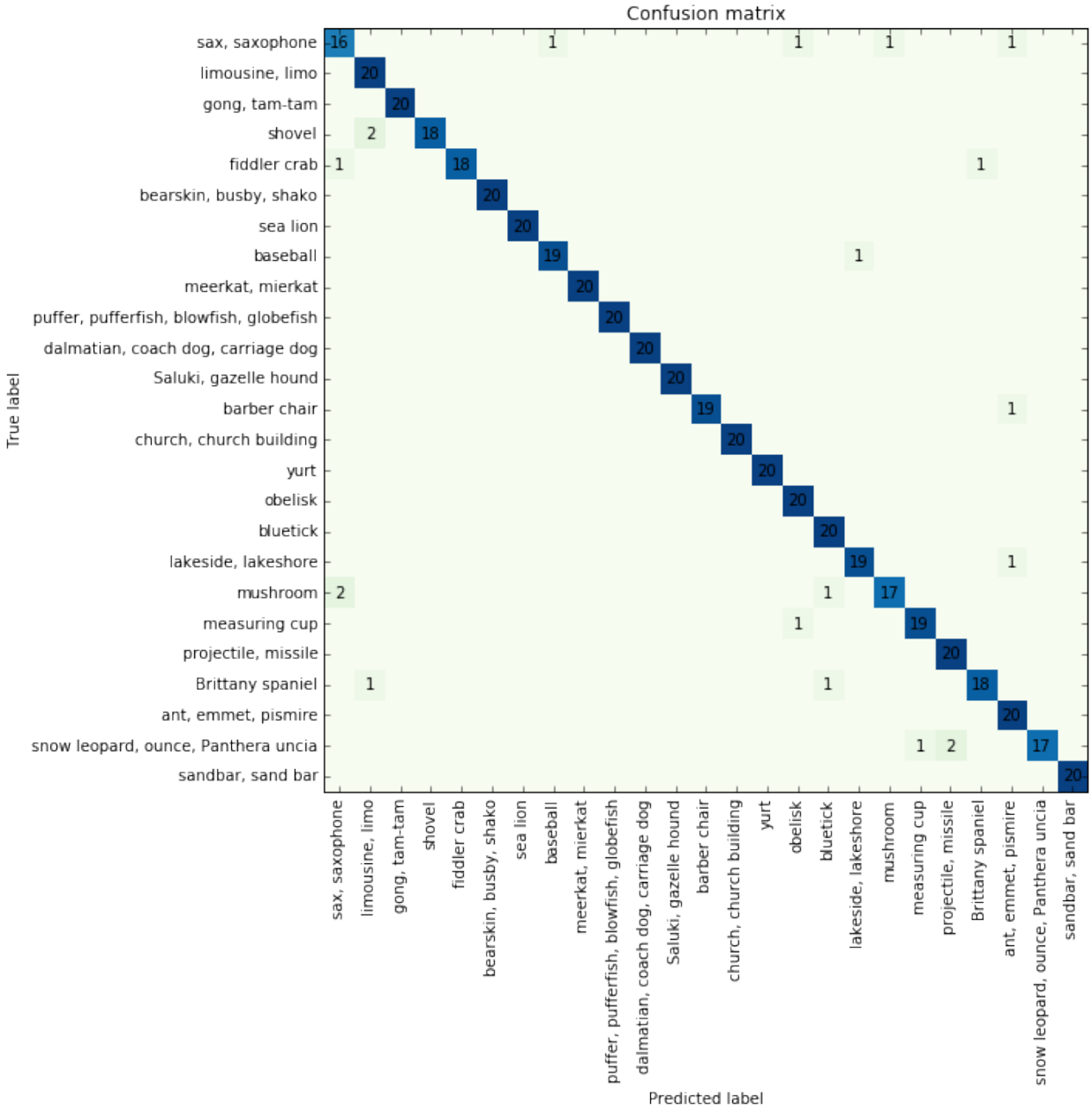


Training size: 2500; ir: 0.0001; fine-tune range: all layers;
Mean class accuracy: 0.944

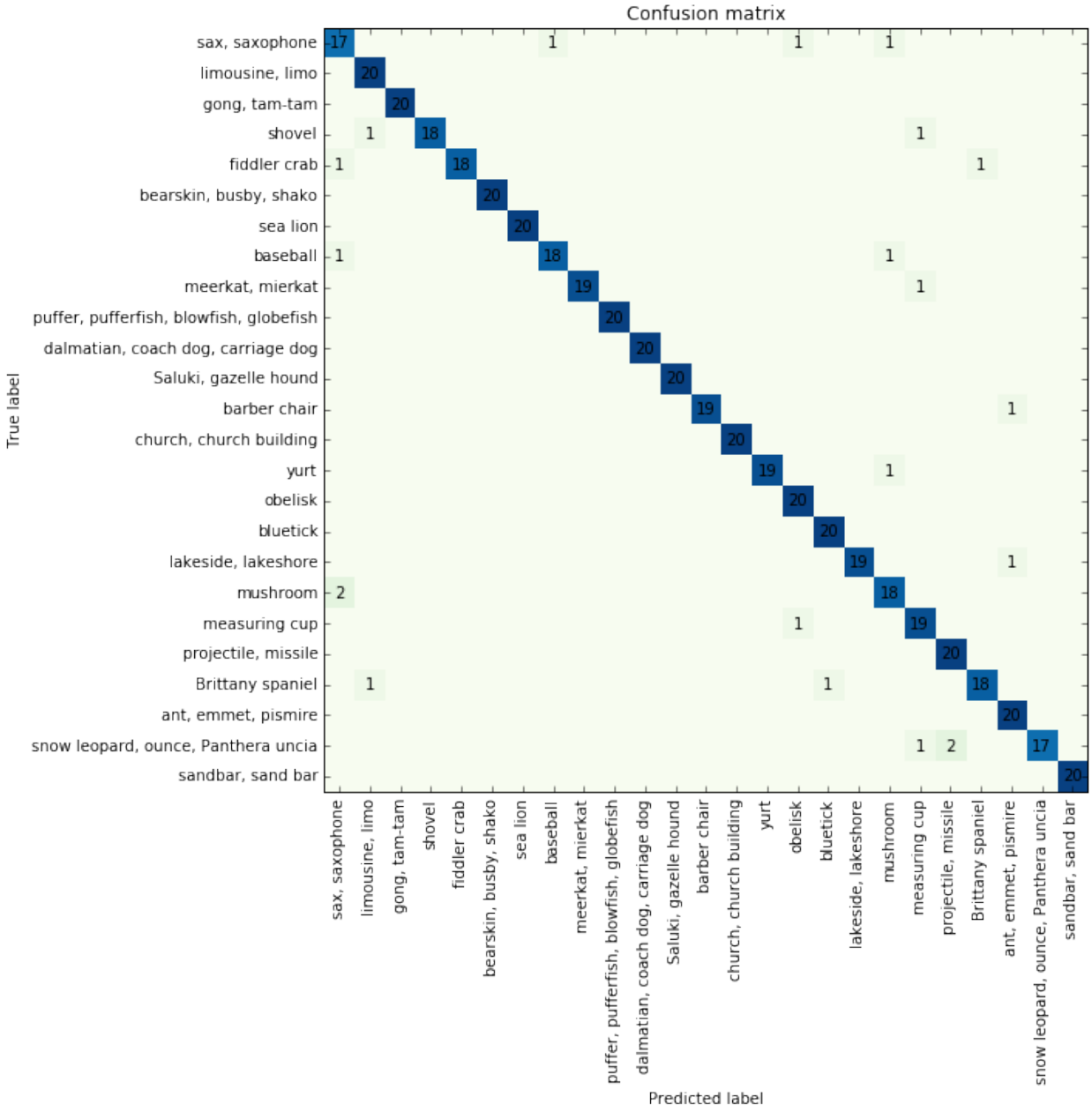
Now for the results for a faster base learning rate (100x previous experiments).



Training size: 31409; ir: 0.0001; fine-tune range: all layers;
Mean class accuracy: 0.960

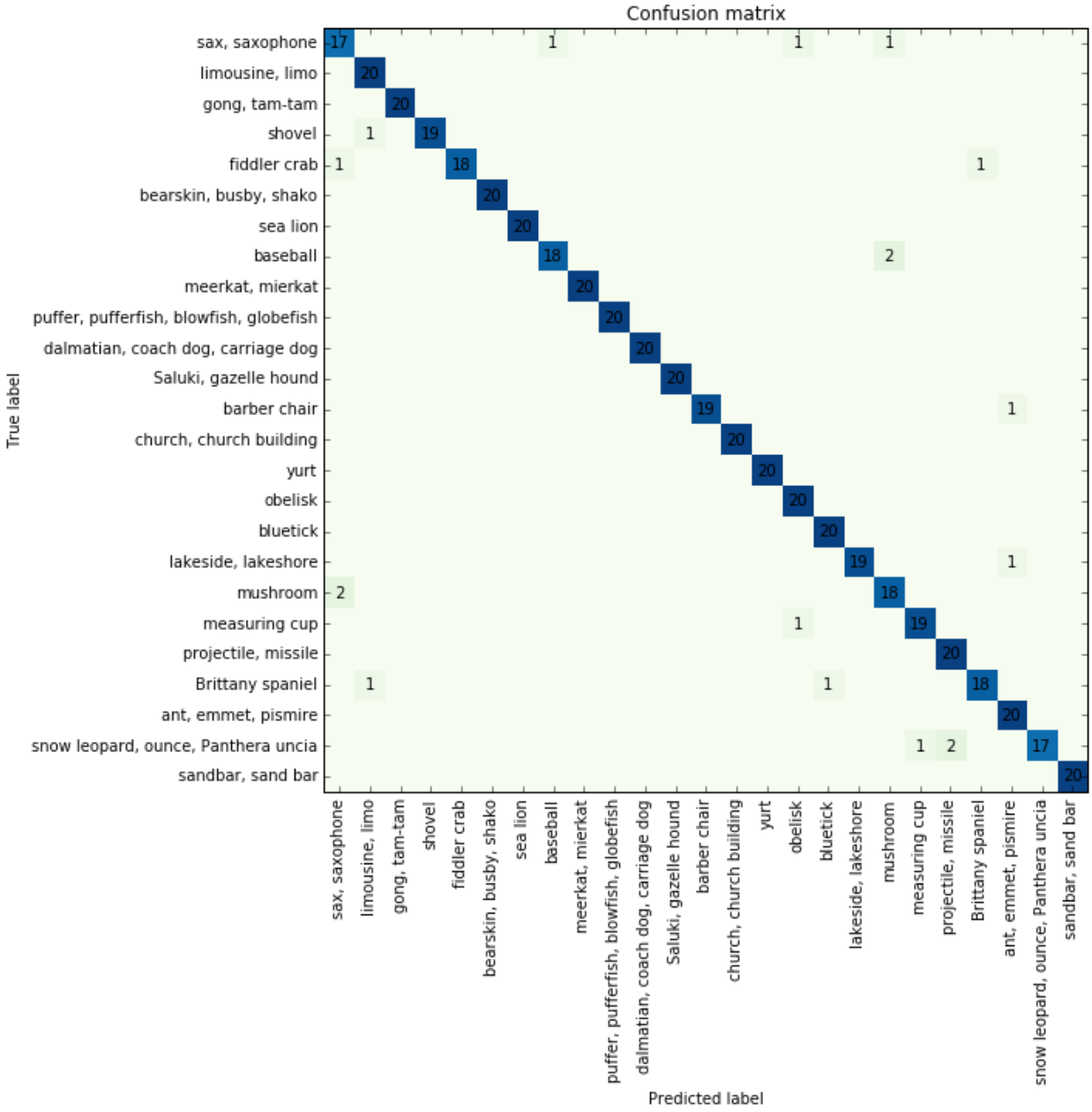


Training size: 31409; lr: 0.0001; fine tune range: fc layers only;
Mean class accuracy: 0.958



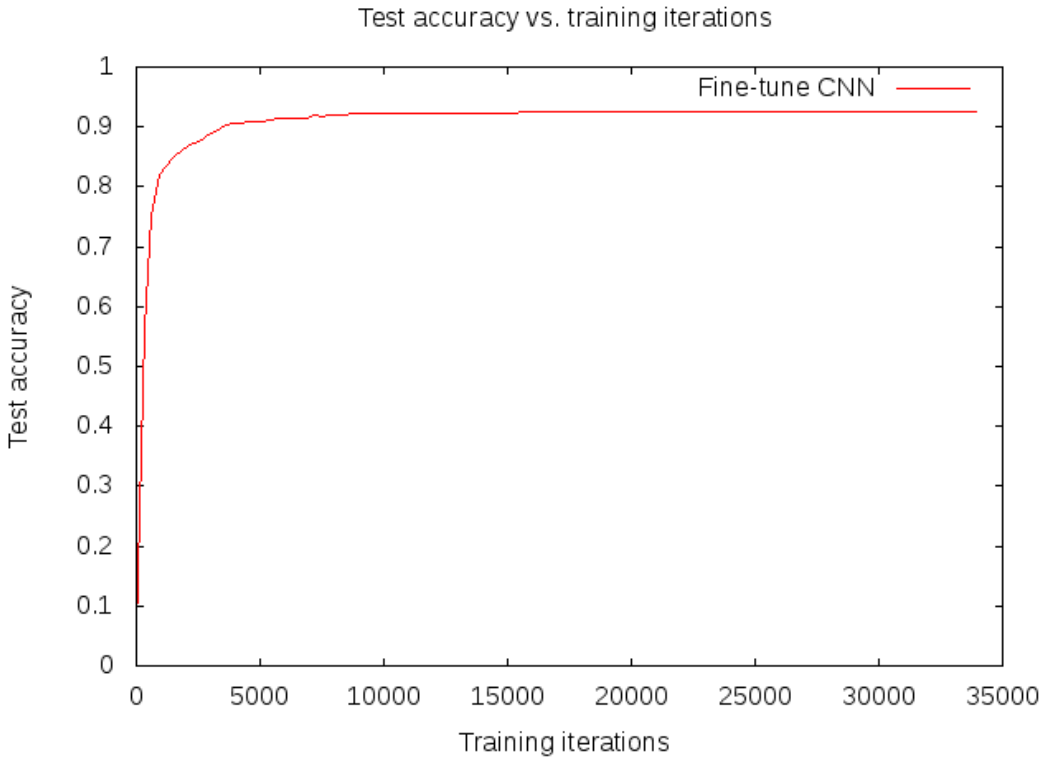
Training size: 31409; ir: 0.0001; fine-tune range: fc8 layers only;
Mean class accuracy: 0.964

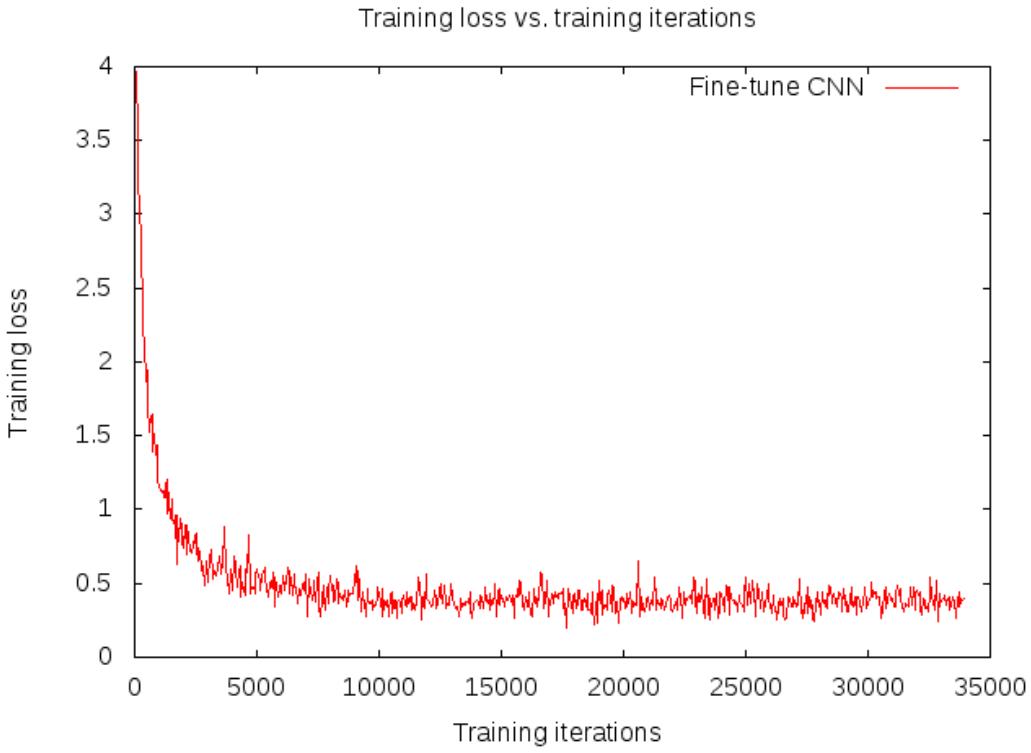
Lastly, we tried with only fine-tuning the last layer and freeze everything else.



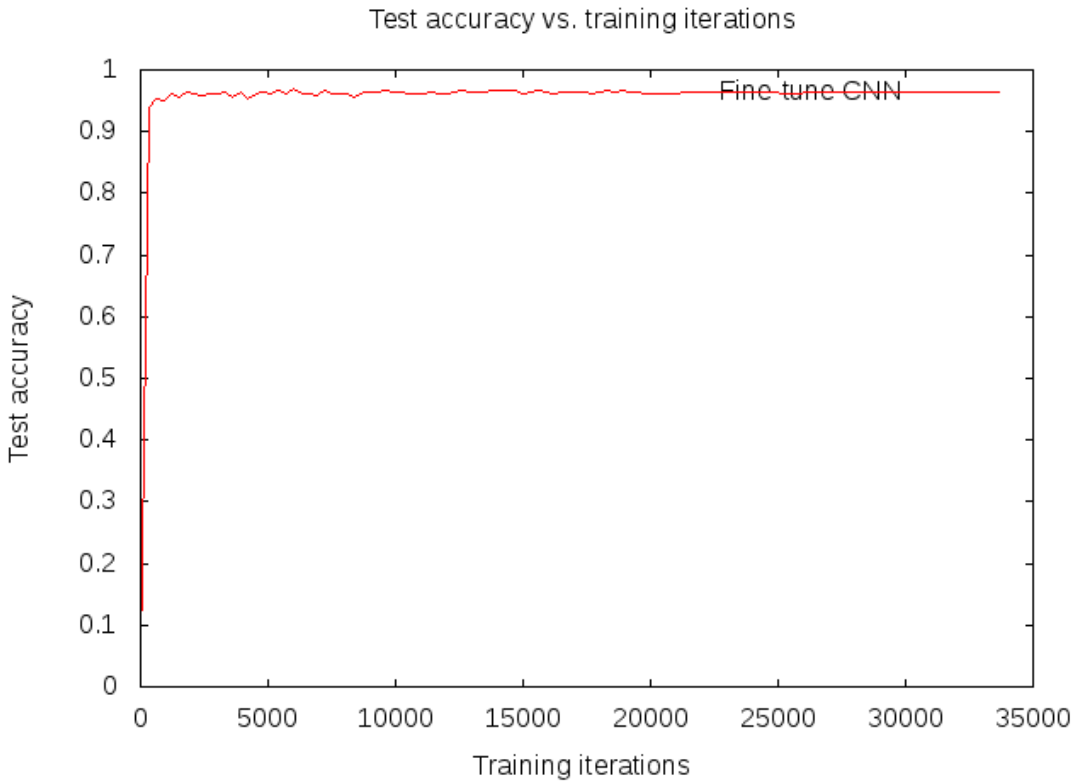
The below are the loss and test accuracy over time / iterations during training. I only showed the two runs fine-tuning with all fc layers, one with slower learning rate and the other faster. Recall that their converged accuracies were 0.912 and 0.958 respectively.

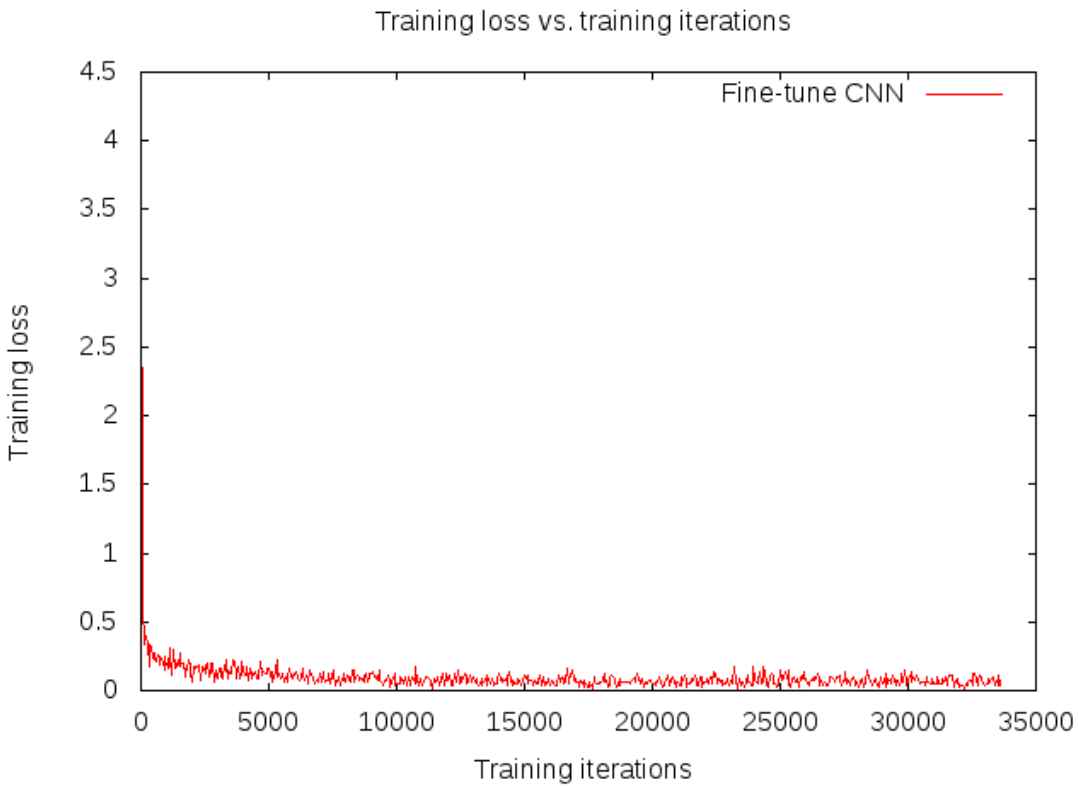
lr: 0.000001:


Test accuracy vs. training iterations
Fine-tune CNN
Test accuracy
Training iterations
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5000
10000
15000
20000
25000
30000
35000


Training loss vs. training iterations
Fine-tune CNN
Training loss
Training iterations
0
0.5
1
1.5
2
2.5
3
3.5
4
0
5000
10000
15000
20000
25000
30000
35000

lr: 0.0001:


Test accuracy vs. training iterations
Fine-tune CNN
Test accuracy
Training iterations
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5000
10000
15000
20000
25000
30000
35000


Training loss vs. training iterations
Fine-tune CNN
Training loss
Training iterations
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
5000
10000
15000
20000
25000
30000
35000

Analysis

From the results, we can conclude that the base learning rate is an important parameter to set in order to achieve good results. For the lower base learning rate runs, the system's loss eventually converged a value around 0.4, it can be a deceiving signal that the system has fully converged. However, with the later experiments running on a larger base learning rate, there's an average improvement of 4% for all the variations of fine-tuning. The loss and accuracy plots also show that the convergence in loss and accuracy are much faster and to a better value if selected properly.

It is also a trend that having a magnitude more data does not improve the accuracy of the test set by much if at all. For the slower learning rate, extra data contributed nothing in terms of accuracy. For the higher learning rate, the extra data did at the end provide a 1.6% improvement when fine-tuning on the entire architecture. This is reasonable because the CNN has already been pre-trained with large data for get a good and general feature extractor. By modifying the weights to fit the smaller data, this generality of the filter weights takes a hit. Since the data we train and test on has a good amount of intra-class variance, it is beneficial to be as general as possible.

This leads to the observation about fine-tuning strategies. Fine-tuning on fc8 alone proved to have the highest accuracy among all other types of fine-tuning. This shows the power of big data. Despite the fact that the classification task has been narrowed down to 25 from 1000 classes, the weights learned from a larger, even though might not contain all the classes in our domain, dataset produces better representations for the input images.

Additionally, I have done tests on a special case test to see the properties of the CNN. I fine-tuned the network on only 24 classes training examples. During test time, as expected the 24 classes trained are well predicted. However, when it comes to the unseen class, the guesses are rarely that of the untrained class. This shows that perhaps the more examples we have for one class the higher the bias is for that class. During training, it also showed random behavior of divergence using small base learning rate similar to the other experiments. This I have no explanation for.

Overall, there isn't one specific class that is often confused with other ones. Due to the high prediction accuracy, this is expected.

Most of my code can be found in the following repository:
https://github.com/willxie/visual_recognition