CS B657 Workshop: Hands-on Deep Learning with Caffe

Today's workshop will introduce you to using the open-source Caffe software package for deep learning with convolutional neural networks. Caffe is just one of many packages that have become available recently (Theano, TensorFlow, etc.), but is one of the most popular in the vision community, mostly because it is supported by an academic group (Berkeley) as opposed to being tied to any one company (Google for TensorFlow, Facebook for Torch, etc). As we saw in class, deep learning can be extremely computationally demanding. Caffe is highly optimized to take advantage of modern Graphics Processing Units (GPUs), and easily outperforms CPU-only implementations by a factor of 10 or more with a high-end GPU. Fortunately, you have access to many GPUs – 676 high-end GPUs, each with 32 compute cores – on IU's Big Red II supercomputer. (Big Red II was, when launced in 2013, the fastest supercomputer on a university. It's a Cray XE6/XK7 with a total of 1,364 CPUs, 21,824 processor cores, 43,648 GB of memory, nearly 10 petabytes of disk space, and a peak performance over 1 petaFLOP (1 quadrillion operations per second)).

Caffe is a complicated software package and getting it running is always a little tricky. We'll walk you through the steps here, and also give you some experience on using Caffe for real-life classification problems.

Part 1: Getting started

1. Hopefully you've already applied for a Big Red II account and one has been created for you. If not, join a group with someone who does have an account, just for today's workshop. Go to `https://itaccounts.iu.edu/` to create one for later use.

2. Log in to the supercomputer via ssh to `bigred2.uits.iu.edu`.

3. You have a home directory on BRII that has a maximum storage area of 100GB. It's mounted on a shared network drive, which is problematic for Caffe's internal database (LMDB), so we'll do our work in a scratch space instead. **However the scratch space is automatically deleted after 60 days** so make sure to move files you care about to your home directory!

   Go to your scratch space and make a local copy of the Caffe distribution:

   ```
   cd /N/dc2/scratch/your-user-id   # substitute in your user ID of course!
   cp -R -d /N/soft/cle5/caffe/20170112/caffe-master/ caffe
   cd caffe
   ```

4. Big Red II supports a large variety of different libraries and software packages, many of which would conflict with one another if all were installed at the same time. Big Red II thus uses the Linux module system to allow you to easily select which libraries and packages you need. To load the ones required for Caffe, type:

   ```
   module switch PrgEnv-cray/5.2.82 PrgEnv-gnu/5.2.82
   module add cudatoolkit/7.5.18-1.0502.10743.2.1
   module add opencv
   module add boost/1.54.0
   module add cray-hdf5/1.8.16
   module add intel/15.0.3.187
   module add python/2.7.8
   module add caffe
   export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/N/soft/cle5/caffe/20170112/caffe-master/.build_release/lib
   ```

   (You may want to create a UNIX script that will do this automatically, so you don't have to type this in every time you want to use Big Red II.)

Part 2: A first example: MNIST

Let's start by working through one of the example datasets that comes with Caffe, the MNIST data. This is the original handwriting recognition dataset that inspired CNNs nearly 20 years ago.

1. Download and prepare the MNIST data:

   ```
   ./data/mnist/get_mnist.sh
   ./examples/mnist/create_mnist.sh
   ```

2. You might be wondering which of Big Red II's 1,364 CPUs you're currently using, and how you can get access to the other 1,363 of them. The answer is that you're not able to access any of the compute or GPU nodes directly. Instead, you request compute nodes through a job scheduler: you request a certain number and type of nodes, you wait until the system has enough nodes available to meet your specifications, and then you run the program though a special scheduling program called `aprun`. In our case, let's request interactive (as opposed to batch) time on a single GPU node:

   ```
   qsub -I -q gpu
   ```

   You may have to wait a while – a few seconds, or maybe a few minutes – for a node to become available.

3. Head over to your directory on the Data Capacitor:

   ```
   cd /N/dc2/scratch/your-user-id/caffe
   ```

4. Repeat the commands in Part 1 Step 4 above.

5. Now, let's check that everything so far is set up properly and that Caffe can access the GPU:

   ```
   aprun build/tools/caffe device_query -gpu 0
   ```

   If successful, you should see a list of information about the system's GPU.

6. Finally, let's start deep learning!

   ```
   aprun examples/mnist/train_lenet.sh
   ```

   If all is set up correctly, you'll now see Caffe swinging into action, and displaying constant information about its learning progress. Every few iterations, it displays a message telling you the current *loss* (e.g. current training error – the sum of squared distances between predictions and ground truth across the training data). You should see this number generally go down over time. Every 1000 or so iterations, Caffe will pause training to test its current model on the test dataset, and will display the current accuracy and the current loss. Over time, if training goes well, we'd expect the loss to decrease and the accuracy to increase.

7. Training will stop automatically after 10,000 iterations, or you can quit it manually with CTRL-C if you get bored. The final model is stored in `lenet_iter_10000`, and this model can then be used for classifying new digits.

8. The power of Caffe is that it allows you to customize the network architecture and parameters. We just used the defaults above, but you can change these defaults by modifying the Caffe model and solver configuration files, which are called the "prototxt" files. The relevant files for this example are `examples/mnist/lenet_solver.prototxt` and `examples/mnist/lenet_train_test.prototxt`. Read about these files and their settings in the MNIST tutorial here: `http://caffe.berkeleyvision.org/gathered/examples/mnist.html`.

Part 3: Image classification on your own data from scratch

Choose a problem with some image data to try out. For example, you could use the landmark classification data from A2, e.g. with 75 images for training and 25 for test. To do this:

1. Generate two files, train.txt and test.txt, that contain the image file names and correct ground truth labels. The labels should be integers starting at 0, and the format of each line should be "image_filename label", e.g.

   ```
   image1.jpg 3
   image2.jpg 4
   ```

   The test.txt file is actually validation data, not testing data.

2. Design your network and set meta-parameters. The tricky part is how to set these meta-parameters. One method is to set the parameters and then train the network, observing two things: (1) if the network is converging, and (2) is it fast enough? Based on your observation you can then readjust your parameters. In addition, you can also set other things such as the output directory, name of the net, etc. To set parameters and architecture, you'll need to edit train_val.prototxt and solver.prototxt.

3. To do the training, first get access to GPU, as before:

   ```
   qsub -I -q gpu
   cd /N/dc2/scratch/your-user-id/caffe
   aprun build/tools/caffe device_query -gpu 0
   ```

4. Then perform training:

   ```
   aprun ./build/tools/caffe train -solver MyExample/solver.prototxt -gpu 0
   ```

5. How to get the results? Caffe provides python script to extract features from a special layer.

   ```
   aprun ./build/tools/extract_features.bin
   MyExample/my_flickr_style_iter_100.caffemodel MyExample/train_val.prototxt
   fc7 MyExample/features 10 leveldb
   ```

Part 4: Image classification with fine-tuning

Were you satisfied with the result of classification in Part 3? Well, we can borrow a pre-trained model (like AlexNet – which was trained on millions of images!) and use it on our comparatively small dataset to (hopefully!) get better results.

1. Add python

   ```
   module load python
   ```

2. Download 100 images and train/val file lists.

   ```
   python examples/finetune_flickr_style/assemble_data.py --images=100
   ```

3. Compute the imagenet mean file. (Why do we need this mean file? Is it necessary to compute mean file?)

   ```
   ./data/ilsvrc12/get_ilsvrc_aux.sh
   ```

4. Adjust the learning rate and number of outputs of the fully-connected layer. (Why do we need to do this? How do we adjust the overall learning as well as the learning rate of output layer? Why do we need to change the size of the output layer?)

   ```
   1. learning rate: vi models/finetune_flickr_style/solver.prototxt
   2. snapshotting: vi models/finetune_flickr_style/solver.prototxt
   3. number of output layer: vi models/finetune_flickr_style/train_val.prototxt
   ```

5. Download pre-trained model.

   ```
   python scripts/download_model_binary.py models/bvlc_reference_caffenet
   ```

6. Get access to GPU

   ```
   qsub -I -q gpu
   cd /N/dc2/scratch/your-user-id/caffe
   aprun build/tools/caffe device_query -gpu 0
   ```

7. Training

   ```
   aprun ./build/tools/caffe train -solver
   models/finetune_flickr_style/solver.prototxt -weights
   models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel -gpu 0
   ```