

# CSCI 4963/6963 – ML for Autonomous Systems

## Homework 5

### Overview

This is a large-scale deep learning assignment. You will work in teams of 2 or 3. You will train a reasonably large convolutional neural network (CNN) to classify buildings on the RPI campus. We have collected images of 11 buildings on campus: 87gym, Amos Eaton, EMPAC, Greene, JEC, Lally, Library, Ricketts, Sage, Troy Building, Voorhees. There are roughly 500 images per building, though the numbers are not exactly the same. You are given the full-size images,  $4032 \times 3024 \times 3$ , though you will probably want to reduce them before feeding them into the neural network. I have separated the data into training and test sets, and there is also a separate hidden test set that we will be using to grade you.

To warm up, I have also provided a very small CNN for CIFAR10 that you can use as a starting point. You are required to reach a certain accuracy on CIFAR10 also. The challenges presented by CIFAR10 are by and large the same (modulo the size of the image), so understanding how to improve your CIFAR10 performance will greatly help for the big task.

### Logistics

For this assignment, you will be using the AiMOS cluster. It is essentially a super-computer consisting of multiple machines. Since it is shared across RPI and beyond, AiMOS works by scheduling your jobs (i.e., programs). Every time you run a big-ish job, you are supposed to schedule it so that the machines don't get overwhelmed.

**Connecting to the server.** To connect to the server, you need to execute the `ssh` command (either from a Windows Powershell or a Unix command line (in the case of Mac/Linux)). In all examples, I will use my account name, **MLA1vnvr**. Your account name should start with the same first four letters (MLA1 is the name of the project). Make sure to replace my account name with yours in all commands.

To connect, you first need to `ssh` to a landing pad. Landing pads are essentially AiMOS's interface with the outside world. These are not powerful machines and are not supposed to execute any heavy-duty tasks. Typically, I just `ssh` to a landing pad and then directly to a main machine. There are four landing pads: `blp01`, `blp02`, `blp03` and `blp04`. You can connect to either one as follows:

```
ssh MLA1vnvr@blp01.ccni.rpi.edu
```

Once you enter the command, you will be asked for your PIC+Token. The PIC is 4-digit number that you set up online. The Token is a 6-digit number and should be available on your Google Authenticator app. Just enter them together and press Enter (e.g., if PIC=1234 and Token=012345, just type 1234012345 and press Enter). You will then be prompted for your password, which you must have also set up online.

Once you connect to a landing pad, you can connect to the main machines. There are two options here: dcsfen01 and dcsfen02. Again, it shouldn't matter which one you use, e.g.,

```
ssh MLAlvnvr@dcsfen01.ccni.rpi.edu
```

This time you only need to enter your password. Keep in mind that all machines mount the same file system, so you should see the same files no matter where you are (include the landing pads).

**AiMOS file system.** Your home directory has 4 folders: barn, barn-shared, scratch, scratch-shared. By convention, the data and installation files are in the scratch folders, and you code goes in the barn folders. **The barn-shared and scratch-shared folders are accessible by everyone, so be careful what you put there!**

Starting code is provided in the barn-shared folder. Please do not modify anything there and just copy everything from that folder to your own barn folder.

Images are in the scratch-shared/all folder. You are welcome to copy images to your own scratch folder (or to your own machine if you wish to do small tests). Please do not distribute any images to people who are not in the class.

The conda environment is in the scratch-shared/AI-TOOLS folder. You shouldn't have to touch this folder, except for the very first time you log on (see Getting started).

**Navigating the file system.** If you've never used a command line interface before, it might be hard initially. There are plenty of good resources online. The main commands you will need are `cp` (copy a file), `cd` (enter a directory), `ls` (list items in directory), `rm` (remove a file).

**Running your code.** Because AiMOS is a shared cluster, it works by scheduling jobs. While you can still run commands normally (e.g., `python train.py`), this should only be done for small tasks. If you run a bigger task this way, it may be killed and you are violating the responsible use agreement you signed. For bigger jobs, you need to use `srun`, which is the scheduling tool. The basic usage is as follows

```
srun -t 1 --gres=gpu:1 python train_buildings.py
```

The `-t` flag indicates how many minutes your code can run for before it is killed. This can be conservative (e.g., a couple of hours). The GPU command gives you 1 GPU – you shouldn't have to change that. The rest is the actual command you would like to use.

**Moving files to and from the server.** To copy a file from your machine to AiMOS, use the `scp` command:

```
scp your_file.py MLA1vnvr@blp01.ccni.rpi.edu:barn
```

You will once again need to do the 2-step authentication. Note the folder after the colon (:); you always have to specify a folder when you `scp`.

To copy a file from AiMOS to your machine, use `sftp`:

```
sftp MLA1vnvr@blp01.ccni.rpi.edu
```

After the 2-step authentication, you are essentially `ssh'd` into `blp`, but you can't do much beyond moving around. If you want to copy a file to your machine, just call `get`:

```
get filename.py
```

And the file will be copied to the directory where you called `sftp` from.

**Using command line text editors.** All command line text editors have a somewhat steep learning curve. The two most popular are `emacs` (my choice) and `vim`. Both have non-standard shortcuts and modes, if you have never used them before. Another choice is `nano`, which may be easier to use initially. You are also welcome to write code on your own machine and send it over, though constantly `scp`-ing may get annoying due to the 2-step authentication.

**IMPORTANT: closing the command line.** Oftentimes, you may want to start your code, and then turn off your machine. That's possible, but you need to be careful. If you just close your command-line interface, your session will be interrupted and all progress will be lost. To avoid this, you should use `tmux`, which lets you start a session and detach from it without killing it. Using `tmux` is quite easy. Once you `ssh` to `dcshen`, you just type

```
tmux
```

This will put you in a new session. Note you will need to activate your `conda` environment again (see Getting started). After that, just run your code as normal (with `srun`). To detach from the session, just type `Ctrl+b` and then `d`. After this, you can safely close your terminal. If you want to get back to the session, `ssh` to `dcshen` again and type

```
tmux attach
```

This will bring you back. Note that `tmux` does not keep your entire print history, so keep that in mind. If you want to scroll up, type `Ctrl+b`, followed by `[`. To exit scrolling, press `q`. To exit the `tmux`, type `exit`.

## Getting Started

To get started, you need to initialize your conda environment. This only needs to be done once, the first time you ssh. Navigate to the `~/scratch-shared/AI-TOOLS/AI-CONDA/bin` folder and run the conda init command:

```
cd ~/scratch-shared/AI-TOOLS/AI-CONDA/bin
./conda init
```

This will add conda to your path. Restart your connection after this.

Every time you wish to run your code, you need to activate the shared conda environment, which contains the pytorch installation, among other things. To activate it, just type (from any folder):

```
conda activate pytorch-env-shared
```

Now you're ready to run your training code. Keep in mind that the `train_buildings.py` script will crash initially because the CNN class is empty.

## Grading

The warm-up task is to get 70% test accuracy on CIFAR10. I have provided you with a very small CNN (1 convolutional and 1 fully-connected layer) that you can also use as an example to build your big CNN. Try to understand why your accuracy doesn't increase after a certain point and how you can improve it. This will be worth a total of 10 points. **Your small CNN cannot have more than 4 hidden layers total.**

The main task is to achieve 75% test accuracy on the building dataset (as measured on the hidden test set). You will lose 5 percentage points for each point below 75. You are allowed to use any number of fully-connected and convolutional layers, as well as batch normalization, dropout and weight decay. You are also allowed to manipulate the training data in any way you see fit – I have normalized the data and converted it to pytorch floats to get you started (in fact, my normalization is borrowed from the CIFAR10, so it may not be perfect). **The only constraint is that your network cannot have more than 20 million parameters.**

The two best performing teams (on the hidden test set) will receive an extra 10 points.

Once you are done with your training, please provide short answers to the questions below:

- 1) How many parameters does your network have? Please provide your calculations, layer by layer. You can ignore batch normalization parameters – just count the convolution and fully connected ones.
- 2) How did you improve the CIFAR10 accuracy? What was the issue?

## Hints and Tips

- Start early. Training on the big dataset is very slow, on the order of 10-15 minutes per epoch. Once you have set up a good architecture, you should expect to reach 75% within 15-20 epochs, which will require at least 5-6 hours of training.
- Use tmux. It will make your life much easier when you need to work on other stuff.
- AiMOS is sometimes down and sometimes fully occupied. Do not leave stuff for the last day because it is very likely that AiMOS will not cooperate.
- Save your model often. I have provided a small example of how to save and load your code. You may want to keep track of how many epochs each model has been trained for.
- Start small. You might want to create a very small training set (e.g., 100 examples), to debug your architecture. Also use CIFAR10 as an example as much as possible.
- You will probably need to shrink the images dramatically. I have provided a resize factor to make your life easier. Keep in mind that ImageNet images, for example, are  $224 \times 224 \times 3$ , so you probably want to aim for that ballpark. You can also plot your rescaled images to see what they look like.
- Check out the AlexNet architecture online for an inspiration. Note that dimensions will not match. Also you probably don't need huge fully-connected layers (or max pool).
- Beware big fully connected layers. Chances are you will run out of memory (even on AiMOS!) the first time you switch from a convolutional to a fully connected layer. Always keep track of how many elements each layer has (and how many weights).
- Check out the `torchvision.transforms` for various ways to augment your training (but not testing!) data.
- You may want to have a small validation set to tune hyper-parameters, but keep in mind that the training set is not that big. You are probably fine tuning on the test set in this assignment.
- When you use `srn`, you should use set the `flush` flag in the print function; otherwise, your print statement may be buffered and will not show up in real time. For example, to print a variable `var`, call `print(var, flush=True)`.

## Submission

Please use LMS to submit a zip file containing 1) your training **.py** code, along with instructions on how to run it, 2) a small test **.py** that we can use on a given test set, 3) your trained model (in **.pth** file), and 4) a **.pdf** file containing your answers to the above questions. The deadline is **11:59 pm, Thursday, Nov. 3**.

**Important:** 3 days after the deadline (but not before), I will ask you to put your code and trained models in the barn-shared folder, so that we don't have to move files back and forth when we grade.