Assignment 1.

TAs: **Lerrel** (lerrelp@cs.cmu.edu) & **Senthil** (spurushw@andrew.cmu.edu)

In this assignment, you need to implement the Fully Convolutional Networks (FCN) in Torch for surface normal estimation. This may be a challenging depending on your background. So please start **working on this early.**

Here are the list of contents:

A. Setup on Latedays
B. Assignment setup
C. Datasets and Background
D. Finetuning FCN using ImageNet pretrained AlexNet (50%)
E. Training FCN from scratch (50%)

The codes, scripts and other files needed for the assignment are in assignment.tar.gz, which can be downloaded from:

https://www.dropbox.com/s/12f7yy0mms9u4w1/assignment1_16824.tar.gz?dl=0

For students who are using their own machines, all the other models and data you need can be downloaded from here:

http://ladoga.graphics.cs.cmu.edu/xiaolonw/assignment2/

After finishing the tasks, you need to submit a link to pdf and a link to a folder containing your code(online viewable so do not tar or zip). The pdf should contain a report with training curves and answers to questions asked in section D and E. The folder to code should contain all the code you have edited or written from scratch for the assignment.

Before you submit your assignments, make sure everything is in there because you may not be allowed multiple submissions.
SEND YOUR SUBMISSIONS TO: https://goo.gl/forms/Z6AYtS0U0W1cTG5j2
Do not email your submissions to the instructors or TAs.

Please start early, you won't have time or machines to train all these networks if you are late!

Due date: Midnight March 6th

If you have any questions about torch, please read the slides I used in the tutorial first:
https://www.dropbox.com/s/jgygdt51o7gfi4y/torch_tutorial.pptx?dl=0

A. Setup on Latedays

Everyone in the class has access to our latedays cluster with your andrew ID. For example, if your andrew ID is “andrewid”, you can log in latedays by using the following command:

```
ssh andrewid@latedays.andrew.cmu.edu
```

The password is the same one you used for your andrew ID.

After logging in, you are in the root node of the latedays cluster. **You are NOT allowed to use the root node for computation (using MATLAB or running Torch jobs are all forbidden).** This is very important. If anyone breaks the root node down, WE WILL FIND YOU.

First copy the assignment helper code from dropbox or from my home directory /home/lerrelp/assignment1_16824.tar.gz to your home. Untar the file by

```
tar -xvzf assignment1_16824.tar.gz
```

This should leave you with the folder assignment1 in your home directory. Now copy the bashrc provided to your bashrc

```
cp ~/assignment1/bashrc_torch_latedays ~/.bashrc
source ~/.bashrc
```

All the jobs should be submitted through qsub. Please do **NOT** ssh the compute nodes to run the jobs. You can use the following scripts in assignment1 folder to run the jobs:

- assignment1/qsub_scripts/latedays_starter.sh
- assignment1/qsub_scripts/latedays_driver_train.sh
- assignment1/qsub_scripts/latedays_killer.sh

For more information on qsub, see: https://docs.google.com/presentation/d/1Jdm_-0f9UzLDm554NJfCXZ4OQwOY078m9ZmysMmClEc/edit?usp=sharing

As an example, if you want to run train_3dnormal_pretrain.lua in the folder:

```
/home/andrewid/assignment1/scripts_fcn_assignment
```

You will first need to modify the script latedays_driver_train.sh in line 19 as:

```
cd /home/andrewid/assignment1/scripts_fcn_assignment
```

then you can run the command as:

```
./latedays_starter.sh train_3dnormal_pretrain.lua
```

During the experiments, you will need to see what the program prints out, i.e., the logs. You can modify line 30 in latedays_starter.sh to specify where the place you store the logs. It should be “/home/${andrewid}/STOutputs”. Before start training and testing anything, first try to print “Hello World!” using the scripts and find the outputs in the logs.

To kill the jobs you are running, you can use the command to find your job id first:

```
showq | grep andrewid
```

and you will see something like this:

```
28324              andrewid    Running    8  3:06:10:12  Sun Feb  7 22:05:32
```

you can try to stop the job by:

```
./latedays_killer.sh 28324
```

Please note that everyone is only allowed to submit one job at a time. Each machine has 24 cores and each job by default is using 8 cores, so that there can be 4 jobs running in one machine. **Please do not increase the core numbers** of your job and each job should not use more than **3GB GPU** memory (Training AlexNet with batchsize=100 costs around 1.7 GB).
Every student only has 2GB hard disk to save your models and data, please use it carefully.

B. Assignment Setup

Xiaolong (Last year's TA) has installed torch in his directory, we are directly using it with the bashrc in assignment1

For students who are using their own machines:
A new layer called "ReArrange" is written in c and cuda (codes are in the folder of "ReArrange" in assignment.tar.gz). If you are going to use it, you need to compile your package nn and cunn with them. Otherwise, ignore it. You can also just download nn and cunn from Xiaolong's torch and compile them.
(nn: /home/xiaolonw/torch/extra/nn and cunn: /home/xiaolonw/torch/extra/cunn)
If you have questions on how to compile, please **read the torch slides**.

To submit jobs, you are going to use qsub scripts.

C. Datasets and Background

We are using the NYUv2 dataset, which contains around 200K training RGBD images. We convert the depth images into surface normal images. In a surface normal map, each pixel is represented by the direction XYZ. The l2-norm for surface normal of each pixel should be 1. For simplicity, we rescale the normals into 0-255 and save them as a RGB image (We use blue -> X; green -> Y; red -> Z).

- The image list:

Training list: /scratch/16824/3d/list/trainlist_rand.txt
Testing list: /scratch/16824/3d/list/testLabels.txt

- Data:

Training images: /scratch/16824/3d/imgs
Training normals: /scratch/16824/3d/normals_high
Testing images: /scratch/16824/3d/croptest
Testing groundtruths: /scratch/16824/3d/testNormals

- ImageNet pre-trained AlexNet:

/scratch/16824/3d/models/AlexNet

- Regression as classification:

Instead of using regression loss, we are doing classification here. Regression loss might work for ImageNet pre-trained model, however, it works very poorly when I train the network from scratch.

We have a codebook of 40 codewords, we quantize the normals into 40 classes using these codewords. In more details, given a 3-dimension normal vector V, we perform dot product between V and the codewords and get 40 scores, then we select the codeword which gives the highest score. The corresponding index is the label for the pixel.

Thus, for a surface normal map of size h * w * 3, we can convert it to a label map of size h * w. Each pixel label is in the scale from 1 to 40.

The codebook is here:
/scratch/16824/3d/list/codebook_40.txt

D. Finetuning FCN using ImageNet pretrained AlexNet

In this task, you need to train a FCN given a ImageNet pre-trained AlexNet model. Following the FCN paper, we first resize the input image into 512 * 512 and the output for the last convolutional layer will be 16 * 16 resolution. Note that in the FCN paper, the authors add a bilinear upsampling layer (deconv) to resize the output to the size as input image. However, for fast training, we do not use this upsampling in the assignment. We are going to resize the labels to 16 * 16 and use them to backprop. In practice, the performance is very close to the "standard" approach.

There is a small problem in torch that there is no "multisoftmax" layer. (There is actually one in the nnx package, but I think their implementation is wrong. If you don't think so, please have a try). It seems the current LogSoftmax layer only support input of 1-d and 2-d tensor. Thus given the last convolution output tensor with dimension as (batchsize, 40, 16, 16), we need to reshape it into (batchsize * 16 * 16, 40), i.e., taking each output pixel as an individual sample. To do so, we cannot directly apply the "Reshape" or "View" layer. Why is that? Well, please explain why in your report.

I have written a layer called "ReArrange" for you, which is very easy to use, after the last convolutional layer, you can do:

```
model_FCN:add(nn.SpatialConvolution(4096, opt.classnum, 1, 1, 1, 1))
model_FCN:add(nn.ReArrange())
model_FCN:add(nn.LogSoftMax())
```

To train this FCN, you will need to fill in some "TODO" in the codes in assignment.tar.gz . I am going to explain each lua file in the folder "scripts_fcn_assignment" as following:

The data part:

- dataset.lua: This file implement how to sample the images and labels from the dataset, you do not need to change anything in this code.
- donkey.lua: This file is also about reading data, following dataset.lua. You need to implement the functions "makeData_cls" and "makeData_cls_pre", which basically quantize the surface normal into discrete labels (from 1 to 40).
- data.lua: multi threads for reading data, you do not need to do anything here.

The training part:

- fcn_train_cls.lua: you need to implement the training function, use sgd to optimize. Remember to print out the losses during training.
- train_3dnormal_pretrain.lua: the main function for training, you need to setup:
    - The labelsize: it is 16 here, but could be different in the next task.
    - The network architecture: you need to read the pre-trained network. During using it, you might want to skip the dropout layer and the last softmax layer in the pre-trained network.
    - The loss function
    - Activate the data functions, training functions.
    - Convert the model and loss function into cuda
- sanitize.lua: It is a function remove the unnecessary things before saving the model. To save nngraph model, I have tried something hacky. You don't need to do anything here.

The testing part:

- test_normal_cls_soft.lua: Perform testing on 654 images. I have written most of the code, the only place you need to fill in is performing forward propagation given the input images. The outputs are normals saving in text files as well as jpgs for visualization.

The evaluation part:
For evaluation, you need to use the matlab codes in under the folder evaluateNYU in assignment.tar.gz

- convert_mat.m: convert the text output files to mat files. You might need to change the directory to where you save your results and the output size (16 or something else).
- evaluateNYUCanonical.m: change the directory to where you save your results in line 71 and line 53. Get the numbers and paste them to your report.

The metrics:
All the metrics are based on the angle error between the predicted pixel and the groundtruth pixel.

- Mean: the mean error over all pixels. (lower the better)
- Median: the median error over all pixels. (lower the better)
- 11.25: how many percent of pixels how error less than 11.25 degrees. (higher the better)
- 22.50: how many percent of pixels how error less than 22.50 degrees. (higher the better)
- 30.00: how many percent of pixels how error less than 30.00 degrees. (higher the better)

You can train the model for just 10 epochs or more iterations. The baselines for my implementation:

- Results after 5 epochs:
  - Mean: 30.503; Median: 26.565; 11.25: 19.647; 22.50: 42.595; 30: 55.792
- Results after 10 epochs:
  - Mean: 29.211; Median: 24.668; 11.25: 22.594; 22.50: 46.088; 30: 58.775

Things to submit:

- The lua codes you have implemented.
- The training logs (with losses in different iterations).
- The final numbers in report.
- Visualize the prediction results (10 sets), paste them in the report. You can visualize as pairs of images: input on the left and output on the right (please resize them to the same scale, so I can actually see them).

E. Training FCN from scratch

One good thing about this task is you have 200k training samples instead of hundreds of samples in semantic segmentation. You have seen FCN is very powerful in many papers given ImageNet pre-trained network. But how about training FCN from scratch? You will be surprised that there are not many people have done that. You can use the same code in the

last section, but notice that I used lr = 0.001 in the last section and lr =0.01 in this case. Implement your own architecture in train_3dnormal_scratch.lua .

The recommended architecture:
I basically follow the AlexNet architecture for the first 5 conv layers. However, it is not necessary to use 4096 filters in the 2 layers after that. Moreover, I have also replace one of the layer to deconv layer (SpatialFullConvolution) so the output will be 32 * 32 this time. And of course, BatchNormalization is always recommanded. So the detailed architecture will be:

Conv(96,11,4,68) -> BN -> RELU -> Pool(3, 2) -> Conv(256,5,1,2) -> BN -> RELU -> Pool(3, 2) -> Conv(384,3,1,1) -> BN -> RELU -> Conv(384,3,1,1) -> BN -> RELU -> Conv(256,3,1,1) -> BN -> RELU -> Pool(3, 2) -> Conv(1024,6,1,1) -> BN -> RELU -> DeConv(512,4,2,1) -> BN -> RELU -> Conv(40,3,1,1) -> ReArrange -> SoftMax

in which:

- Conv(c, k, s,pad) means the convolution layers have c filters with size k * k and stride is s, pad means padding size.
- DeConv(c, k, s) means the deconvolution layers have c filters with size k * k and stride is s;
- BN means batch normalization layer.
- Pool(k, s) means the pooling kernel size is k * k and stride is s;

You can train the model for just 30 epochs or more iterations. The baselines for my implementation:

- Results after 5 epochs:
  - Mean: 33.414; Median: 30.289; 11.25: 15.764; 22.50: 36.662; 30: 49.534
- Results after 15 epochs:
  - Mean: 30.499; Median: 25.673; 11.25: 21.816; 22.50: 44.513; 30: 56.810
- Results after 30 epochs:
  - Mean: 29.694; Median: 24.274; 11.25: 23.638; 22.50: 46.958; 30: 58.885

Note that you don't have to use my settings. If you have better architecture in mind, please use it!

Things to submit:

- The lua codes you have implemented.
- The training logs (with losses in different iterations).
- The final numbers in report.
- Visualize the prediction results (10 sets), paste them in the report. You can visualize as pairs of images: input on the left and output on the right (please resize them to the same scale, so I can actually see them).
- Visualize Conv1 filters and compare to the one in task 1.