# Project 6 Deep learning for scene recognition

### Due date: 23:59 Sunday December 2nd (2018)

# 0. Acknowledgements

We would like to thank James Tompkin at Brown and James Hays at Georgia Tech for using their material for this project.

# 1. Instructions

Most instructions are the same as before. Here we only describe different points.

1. Generate a zip or tgz package, and upload to coursys (<https://coursys.sfu.ca/2018fa-cmpt-822-g1/>). The package must contain the following in the following layout. **data** folder is large for this project. Please do not include data folder.
   * {SFUID}/
     + {SFUID}.pdf (your write-up, the main document for us to look and grade)
     + code/
       - hyperparameters.py
       - run.py
       - vgg\_model.py
       - your\_model.py
2. Project 6 has 16 points.

# 2. Overview

This is a new coding lab with state-of-the-art deep learning software library TensorFlow. Please expect a few bumps in the mechanics. TensorFlow code with TensorPack functions will look very different from MATLAB, and much of this project is about familiarizing yourself with these systems. TensorFlow is the most popular deep learning library in industry, making revolutions everywhere. Note that CNN training takes time and you will need to wait for many hours. So, start early. You can do other work while waiting for training to be done.

In the project, we will design and train convolutional neural networks (CNNs) for scene recognition using the TensorFlow system. Remember the scene recognition task with bag of words in lab 4. The bag of words model can achieve 50 to 70% accuracy on 15-way scene classification task, which will be used in this lab. We seek to achieve better performance with deep learning. There will be 2 tasks: 1) training a deep network from scratch; and 2) fine-tuning a pre-trained network.

# 3. Programming

### 3.1. TensorFlow installation [ 6pts ]

Install TensorFlow and TensorPack, and familiarize yourself with the stencil code. This will take time. Take it slow and learn to follow the code flow.

Follow the TensorFlow Website (<https://www.tensorflow.org/install/>) and TensorPack website (<https://github.com/ppwwyyxx/tensorpack>) for the installation.

For example, the following steps might work on Windows through PowerShell:

1. Install Python 3.6 (<https://www.python.org/downloads/release/python-363/>)
2. Use Python 3’s package manager pip3 to install TensorFlow:
   1. >pip3 install --upgrade tensorflow
3. Use Python 3’s package manager pip3 to install TensorPack:
   1. >pip3 install --upgrade tensorpack
4. Test the install
   1. >python (or python3)
   2. >>> import tensorflow as tf
   3. >>> hello = tf.constant(‘Hello, TensorFlow!’)
   4. >>> sess = tf.Session()
   5. >>> print(sess.run(hello))

You will need to train a large VGG model. If you have a relatively old laptop with slow CPU, each training might take a few hours and the debugging will be very time consuming. And if you do not code it right (w/ bugs), you might wait for 12 hours and see nothing. Look for a recent computer with a good CPU for training. CSIL lab has machines with Geforce GTX 1050Ti, but they do not have enough GPU memory to run the second task.

If your computer has NVIDIA GPU and want to use it to speed up processing, this requires a bit more set up, but refer to the TensorFlow documentation for the information. When stuck, please reach out for helps and ask questions.

Here are some recommended tutorials on TensorFlow.

1. Getting started with TensorFlow (<https://www.tensorflow.org/get_started/get_started>)
2. MNIST tutorials: Slower pace with more explanations (<https://www.tensorflow.org/get_started/mnist/beginners>) or faster pace with no underlying explanation (<https://www.tensorflow.org/get_started/mnist/pros>).
3. TensorBoard (https://www.tensorflow.org/get\_started/summaries\_and\_tensorboard)

The following is an outline of the stencil code:

1. **run.py**: The top level function for data loading and network training. This is what you will run, for instance  
     
   $> python run.py --task 1 --gpu -1  
     
   Arguments are explained at the top of the file, plus you can inspect them in \_\_main\_\_. If you run this starter code unmodified, then it will train a simple network that achieves ~40% accuracy after 30 epochs—somewhat better than nearest neighbor baseline, but not as good as HOG + bag of words + linear SVM.
2. **parameters.py**: Contains all the tweakable parameters; feel free to add your own if you wish.
3. Then we have two 'models' for your network:
   1. **your\_model.py**: This is your model for Task 1, to be trained from scratch.
   2. **vgg\_model.py**: This is the VGG-16 model for Task 2, to be fine-tuned.

In the write-up, please describe your experimental setup, what OS, how you installed, if you are enabling GPU and etc. There are no specific results to be reported directly in this step. As long as you report some results (even very bad ones) in the following step(s), we know that you completed the installation and can grade this step.

### 3.2. Training from scratch

We will train a CNN to recognize scenes with the provided architecture and a dataset of 1,500 training examples. This isn’t really enough data to gain high accuracy given the number of parameters. Therefore, we will try to

1. Add standardization (feature normalization): run.py, class Scene15.  
     
   We talked about a few normalization strategies in class, for example, subtracting the mean or dividing by the standard deviation of image intensities. Try these combinations.
2. Add dropout regularization: your\_model.py, function \_build\_graph().  
     
   We talked about the technique in class with sample code, but TensorFlow offers an easy way of enabling this. You do not have to implement the dropout mechanism. Read TensorFlow documentation and enable this.
3. Add data augmentation to ‘fake’ more data: run.py, function get\_data().  
     
   Look at the reference <https://tensorpack.readthedocs.io/modules/dataflow.imgaug.html> and try augmentation modules.
4. Change the architecture to be deeper: your\_model.py, function \_build\_graph().  
     
   Try to add more layers of similar forms. This website should be a useful reference <https://www.tensorflow.org/tutorials/layers>.

TODO:

1. [3 pts] Add the above improvements and achieve at least 50% test accuracy. Report the best test error you could achieve and describe what you did to achieve the result.
2. [2 pts] Use TensorBoard to visualize how loss/errors change during training. In the write-up, include the screenshots of the graphs. TensorBoard is a locally-hosted Web-based interface to assess models. Use TensorBoard to visualize your loss and error. Each training session is saved as a set of logs and model weights (which you can then use on new examples).

* &> tensorboard --logdir=train\_log/run
* Load a Web browser and navigate to <http://localhost:6006/>
* Explore!

Training for this part might take 20-30 minutes--40 seconds per epoch on a laptop CPU.

The system writes out data regularly, e.g., training performance data and network weights. The weights especially can quickly eat up laptop hard drives or filesystem quota. You can set None to save\_checkpoints\_steps and save\_checkpoints\_secs inside Config to disable these saving operations when not necessary.

### 3.3. Fine-tuning VGG

We will fine-tune the VGG-F pre-trained CNN to recognize scenes, where the CNN was pre-trained on ImageNet. For this, begin by downloading the VGG-16.npy model (<https://drive.google.com/open?id=1nKUJZtIMrV-taIcg769OrQT5CwDU2ccb>) and placing it in your code/ directory. Training for this part will take many hours: two hours per epoch on a laptop CPU. Leave it to run overnight.

Note that tf.stop\_gradient() is needed for fine-tuning VGG through only the last layers (up until the call). Training the whole VGG takes a lot longer. You can use this call to only fine-tune the last layers, which would significantly speed up the training process.

TODO:

1. [3 pts] Achieve at least 85% test accuracy. You may need to tweak hyper parameters.
2. [2 pts] Use TensorBoard to visualize how loss/errors change during training. In the write-up, include the screenshots of the graphs.