Project 1

- The project is out of 100 points (and worth 15% of your final grade)
- The project is to be completed by groups of two students or at most three students
- Submit your assignment through the Assignments in MyCourses
  - A .pdf file for the write-up, named project1.pdf.
  - A .zip file containing your code, trained parameter values, and a README explaining (briefly!) how to run your trained classifiers, named project1.zip.
  - Code must be able to run on servers for the course.
  - To be fair to other students, please use only 1 GPU on the server at a time.

Grading

- 40% Project report (criteria: correctness, completeness, and clarity; appropriate references and reference formatting)
- 10% Accuracy of the best network
- 15% Implementation and design of an early, simpler network
- 35% Implementation and design for your best network

Task

For this project, we use the NSynth dataset created by researchers from the Magenta project at Google Brain (https://magenta.tensorflow.org/datasets/nsynth). Your task is to classify single note recordings of instruments into one of 10 instrument family names (e.g., pipe organ, jazz organ, and synth organ recordings are all from the family “organ” in the dataset).

There are 11 instrument families (e.g., mallet, organ, piano, strings, guitar, etc.), and 3 instrument source types (acoustic instrument sample, electronic instrument sample, or synthesized sample). The test set does not include any samples for the ‘synth lead’ class, so please remove these from the training and validation sets.

You will create at least two convolutional neural networks, with the goal of recognizing these instrument families with high accuracy. You are welcome to use built-ins for the Tensorflow or Keras libraries, and to use layers/nodes of whatever type you like (e.g., dropout layers, sigmoids, ReLU, tanh, etc.), or build your own.

Input Data (Recordings)

Data is available on the CS machines at:

/home/fac/cmh/tfrecord

Each recording is 4 seconds long; the last second is just the release of the note (e.g., after lifting a piano key). The sample rate is 16kHz (16,000 samples/second; 64,000 samples over the four seconds). To relate this back to images, a 256x256 greyscale image has 65,536 pixels (i.e., not a large image, but a lot of input nodes for a network). This is a rather large input layer, and you will probably want to find a way to reduce the input size. Please keep in mind that you need twice as many samples per second as any frequency you wish to represent. So, for example, to capture a 60Hz frequency (pitch) without noise, you need a minimum of 120Hz (120 samples per second).

The data is stored in a TensorFlow format, as well as JSON files + .wav files for each sample. The JSON and .wav files provide a more intuitive way to browse through the data; the TensorFlow data is in a database format designed for rapid lookup.

Implementation

Use python and Keras/Tensorflow to create your network. Submit your code and weights with a README in project1.zip. You are welcome to use other Python libraries (e.g., Pandas, Matplotlib Scitkitlearn, etc.) for the project.

Visualizations you will Need to Create

- Learning curves for training and validation data
- Visualization of a 1-D audio waveform (sequence of values in [-1,1] representing the volume at each time)
- For each class, waveform for samples where the correct class probability is very high, or very low.
- For each class, waveforms for samples near the decision boundary, where the probability for the correct class is slightly higher/lower than the other classes.
- Confusion matrix, visualizing the frequency of correct classifications and mis-classifications.

Project Report (max. 6 pages, 11pt font)

In the write-up for the project (project1.pdf), include the following.

- **Overview**. A description of your architecture, and summary of your main findings.

- **Training**. Describe your criterion functions, optimization model, and training regimen used for your networks.
- **Results**. Report the results for two networks: 1) your best network, and 2) an earlier network that you created that differs in some non-trivial way. Include the visualizations listed above for both trained networks. In both cases, select the network with the initialization that produced the best test performance.
- **Discussion**. Discuss the difference in results between your two networks. In what ways were your attempts to improve the 'earlier' network successful? What do the visualizations tell us about the behavior of the trained classifiers, and their ability to discriminate between the instrument families?
- **References**. Throughout the report, you should refer to the sources that informed the design of your networks. In this section, list those references. Use ACM or IEEE format for references. This should include the appropriate citation for any base network that you have adapted.

Resources to Consult

- NSynth documentation
- Ch. 11 of the Deep Learning book ('Practical Methodology')
- Tensorflow documentation
- Research papers and other texts on deep learning.

Notes on department machines

You have been given access to the GPU machines weasley, granger, and lovegood, which should run your code much faster than non-GPU-based machines. One important concern is that tensorflow by default tries to reserve all GPUs it can find. The easiest way to ensure this doesn't happen is to set an environment variable. You can do this, e.g., in your python code via

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="0"
```

Where "0" is the name of a free GPU. Your program will then use only this GPU. You can also do this directly in your shell, of course. Use the shell command

```
nvidia-smi
```

to see a list of GPUs.

The data is already loaded on these machines. They are at:

```
/home/fac/cmh/tfrecord
```