CS2470 Project Report

1. Title: Summarizes the main idea of your project. Clever/catchy acronyms and/or puns are welcome.

   CONVOLUTIONAL RECURRENT NEURAL NETWORKS FOR MUSIC CLASSIFICATION

2. Authors: The names of all your group members. Take credit for your awesome work!

   MIAOLIN MIN (mmin2)
   QUANJIA ROU (jquan4)
   YUHANG YANG (yyang102)
   ZEMIAO ZHU (zzhu19)

3. Introduction: What problem are you trying to solve and why?

   Convolutional neural networks (CNNs) have been actively used for various music classification tasks such as music tagging, genre classification, and user-item latent feature prediction for recommendation. CRNNs take advantage of convolutional neural networks (CNNs) for local feature extraction and recurrent neural networks for temporal summarisation of the extracted features. So basically, We implemented a convolutional recurrent neural network (CRNN) for music tagging.

4. Related Work: Are you aware of any, or is there any prior work that you drew on to do your project?

   The network structure and hyper-parameters are from the paper:
   Convolutional recurrent neural networks for music classification (Keunwoo Choi ; György Fazekas ; Mark Sandler ; Kyunghyun Cho)

   Our network code is based on the source code that the author provided. However we also implemented a version using tf.layers without Keras, which adopted the format that we have been always using in the assignments of CS247.

5. Dataset: What data did you use? If you're using a standard dataset (e.g. MNIST), you can just mention that briefly. Otherwise, say something more about where your data come from (especially if there's anything interesting about how you gathered it).

The Million Song Dataset is a freely-available standard dataset of audio features and metadata for a million contemporary popular music tracks. We used 50 tags from the dataset for our classification task.

However, this dataset doesn't provide the source audio files which are necessary for our research, as we need to first extract a spectrogram out of every song and then feed it to the CRNN as input.

To gather the audio clips, we first tried the method recommended by the paper: to download the 30-second preview clips from 7digital.com. This approach failed as 7digital no longer provide public access to its API.

After doing some research, we ended up using iTunes as the audio preview downloading source. As we need to collect massive amount of data from iTunes (i.e. 220k songs = 220GB data), we wrote a data-downloading pipeline which takes care of server rejections by using timeout, and run it on 10 GCP instances simultaneously. It worked well!

6. Method: Describe your data processing, network architecture, training procedure, etc.

A. Data processing

We developed the whole pipeline of downloading and processing audios as follows:

- Download the msd_id's of every song in the dataset splitting.

- Download the file 'unique_tracks.txt' containing information of singer and song names from (in the format msd_id + singer + song name, 1000000 in total, coming from Million Song Dataset).

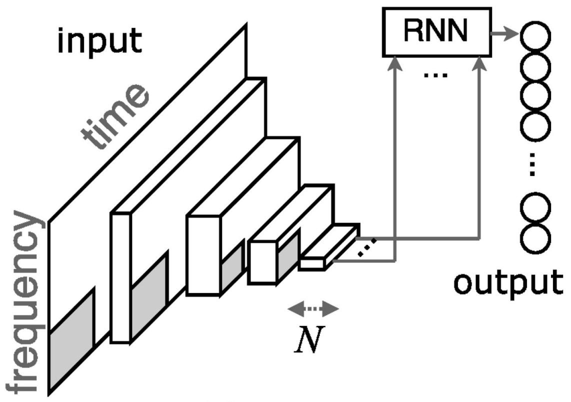
- get_song.py: Call the iTunes API to get the url for the preview clips of the songs in the subset we use. Download the corresponding audio files and store them in './AUDIO_DATA/'.

- get_spectrum.py:
    - Get the spectrum from data_preprosess.py from converting the corresponding audio in the downloaded dataset.
    - Reshape the 2d spectrum matrix to 3d matrix (with shape 96 * 1366 → 96 * 1366 * 1).
    - Stack the 3d matrix of segment size(1000) spectrum and store it (with shape 1000 * 96 * 1366 * 1).

- Store the spectrograms from all xxx.m4a files as x0-x1_spectrum.npy (from x0 to x1), each segment size. Store the segmented labels from 'train_y.npy' in the meantime.

B. Network architecture

According to the paper, CRNN uses a 2-layer RNN with gated recurrent units (GRU) to summarise temporal patterns on the top of two-dimensional 4-layer CNNs. The inputs to the network are 96*1366 dimensional mel-frequency spectrograms, indicating there are 1366 time-frames each having 96 frequency domain features. After the CNN it's condensed to be 15*128, which is then fed into two RNN layers of 15 GRUCells as 128-dimensional embeddings. The last output of the second RNN layer is then fed into a dense layer which outputs the final logits. Batch normalization and Dropout layers are added to improve the network's performance.



(d) CRNN

Because our project is a multi-category classification problem, we used sigmoid cross entropy loss.

We first wrote the network architecture using the familiar tensorflow framework tf.layers (crnn_tensorflow.py). It is fully functional and able to be trained. Later, however, after reading and understanding the paper's source code, we decided to adopt Keras with Tensorflow as the backend in our final training procedure (crnn_keras.py). This is because Keras makes it easier in saving/reading the model, feeding large amount of data as well as visualizing the training progress.

C. Training procedure

We run crnn_keras.py on colab (on GPU) for 8 epochs on 170,000 songs with batch_size to be 10. Among them, we used a 94:6 train:validation splitting during training.

After tweaking the hyper-parameters a little bit, we found the default values from the paper to be the most effective, so we left them unmodified.

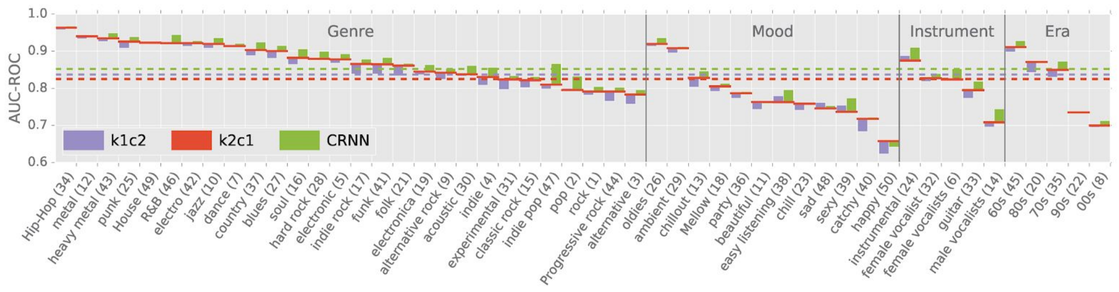
7. Results: What experiments did you run, and what were the outcomes (e.g. how well does the network perform on a held-out test set)?

In test_AUC.py, we used the same test set as is used in the paper (including 24,000 songs). Because our project is a multi-category classification problem, we calculated the AUC (area under curve of ROC curve) on the test set. The final test AUC is 0.855.

We find this result meet our expectation. It is quite close to the final AUC (~0.86) mentioned in the paper. The reason it's a bit less strong might be because we used only 170k songs as our training set (that's 20% smaller in size). Plus, we only trained for 8 epochs so there could be space of improvement.

We also wrote a tool to inspect the tag predictions for any particular song (tag_audio_conditional.py), which takes in single audio file using trained model parameters and output 10 genre tags with highest probabilities. Note that the logits directly output from the model is not accurate: we need to divide them by the prior probabilities of all tags for the final per-tag score to be accurate. This is because of a very unbalanced training data distribution among tags.

We found the top ranking tags to be convincing. Moreover, the higher the average AUC of a tag over all samples is, the more reliable that tag is, i.e. closer to human feelings.



8. Discussion: Did you learn any interesting / non-obvious lessons from doing this project? Did you try a bunch of things that didn't work before arriving at something that did? This is the place to mention such things.

The most interesting lesson we learned is that data mining plays a vital role in deep learning:

1. The collection of data takes a large portion of time:
   In this project, we spent huge effort trying to find a way to get the large amount of songs as our training set. During this process we all gained much experience about data mining, especially collecting large amount of data and storing them in order.

2. More training data improves performance.
   At first we only used 20k songs as our training set, and the final AUC is only ~0.84 which is substantially worse than the paper's result. We then realized we need more data to

improve the network’s accuracy. By using more virtual instances for parallelly downloading, we were able to successfully re-produce the best outcome.