MIE324 Project Report

Music Genre Classification

Robert Adragna

Yuan Hong (Bill) Sun

Word count: 1981

1. Introduction

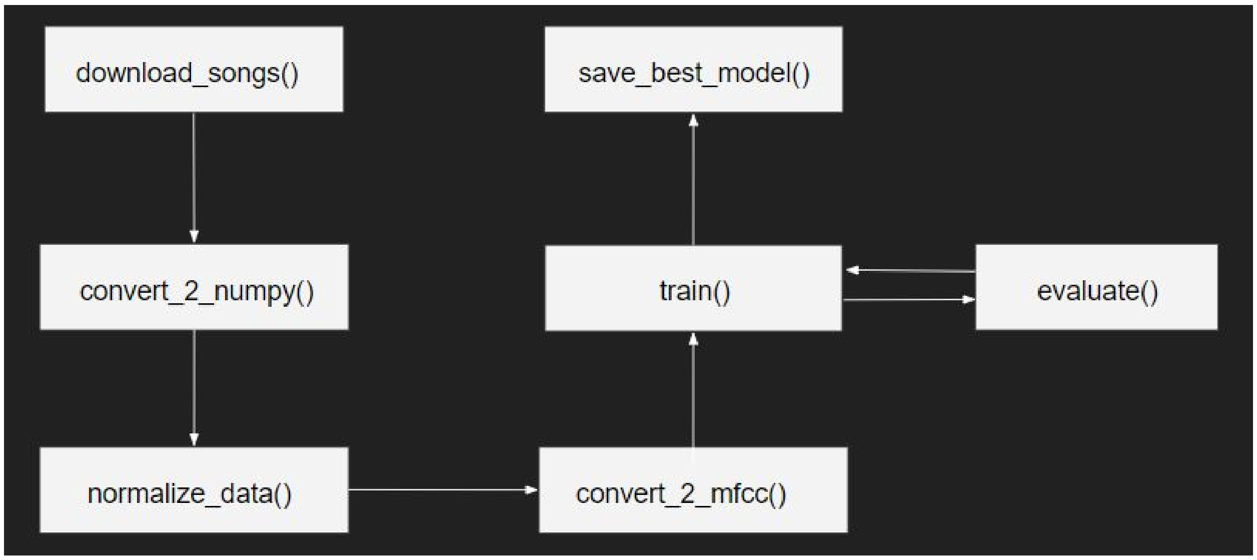
A music genre classifier is a software program that predicts the genre of a piece of music in audio format. These devices are used for tasks such as automatically tagging music for distributors such as Spotify and Billboard and determining appropriate background music for events.

Currently, genre classification is performed manually by humans applying their personal understanding of music. This task has not yet been automated by conventional algorithmic approaches since the distinctions between music genres are relatively subjective and ill-defined. However, the ambiguity of genre classification makes machine intelligence well-suited to this task. Given enough audio data, of which large amounts can be easily harvested from freely available music online, machine learning can observe and make predictions using these ill-defined patterns.

The goal of this project is to build a proof-of-concept music genre classifier using a deep learning approach that can correctly predict the genre and confidence level of Western music from four candidate genres (classical, jazz, rap, rock).

2. Overall Software Structure

2.1. Training and Validation

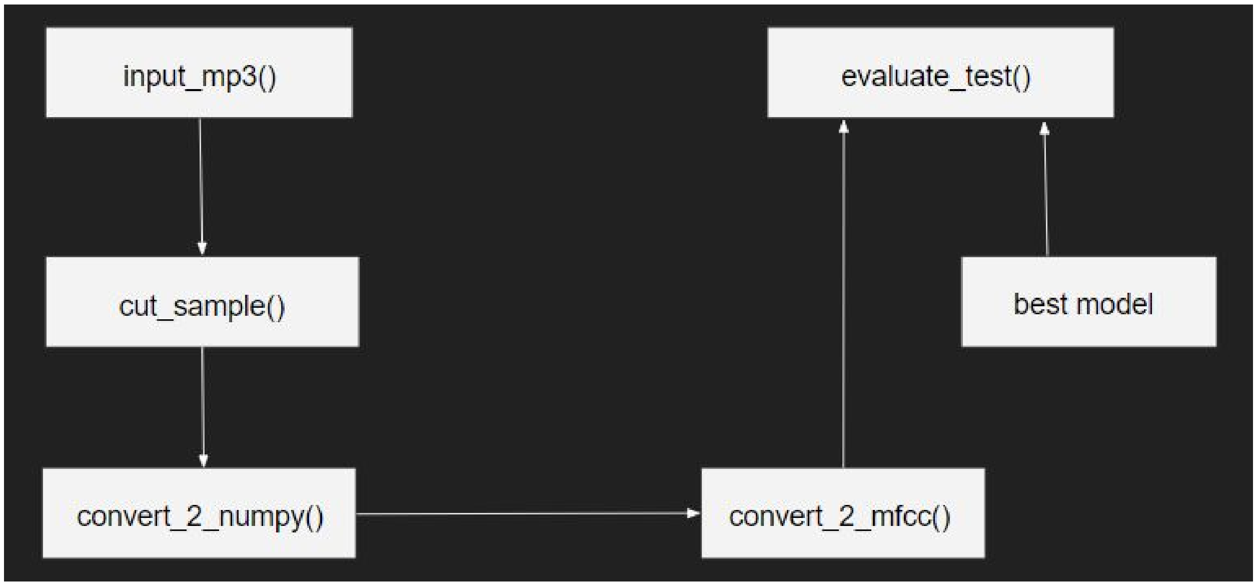


- Download song samples of different genres using the Spotify API

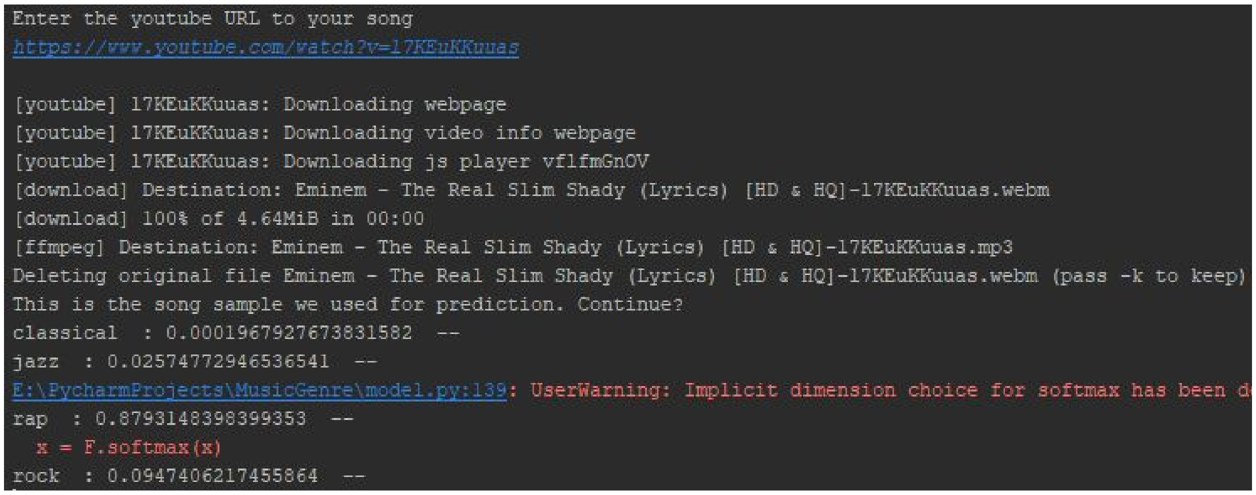
- Generate labels based on song categorization from Spotify
- Take a 10-second sample from each audio file and convert to Numpy array
- Normalize each array
- Convert raw audio arrays into time series of Mel-frequency cepstral coefficients (MFCC) for input into the RNN (will be explained in Section 3.2)
- Split data into training and validation sets

- Main neural network infrastructure
- Training, validation, and evaluation loops
- Generates a model at the end

2.2. Testing infrastructure and user interface



- User picks a song on YouTube and copies the link to the program
- Program downloads the song in mp3 format and cuts out a 10-second sample at a location representative of its genre
- Converts audio into normalized Numpy array and then MFCC
- Takes the model file, and computes the predicted confidence level of each of the 4 genres (shown below)



```
Enter the youtube URL to your song
https://www.youtube.com/watch?v=17KEuKKuuas

[youtube] 17KEuKKuuas: Downloading webpage
[youtube] 17KEuKKuuas: Downloading video info webpage
[youtube] 17KEuKKuuas: Downloading js player vflfmGnOV
[download] Destination: Eminem - The Real Slim Shady (Lyrics) [HD & HQ]-17KEuKKuuas.webm
[download] 100% of 4.64MiB in 00:00
[ffmpeg] Destination: Eminem - The Real Slim Shady (Lyrics) [HD & HQ]-17KEuKKuuas.mp3
Deleting original file Eminem - The Real Slim Shady (Lyrics) [HD & HQ]-17KEuKKuuas.webm (pass -k to keep)
This is the song sample we used for prediction. Continue?
classical  : 0.0001967927673831582  --
jazz  : 0.02574772946536541  --
E:\PycharmProjects\MusicGenre\model.py:139: UserWarning: Implicit dimension choice for softmax has been d
rap  : 0.8793148398399353  --
  x = F.softmax(x)
rock  : 0.0947406217455864  --
```

3. Data

3.1. Sources of data

Our data is comprised of a total of 4640 10-second audio clips of music from four genres: jazz, classical, rap and rock. There are an equal number of songs of each genre to ensure a balanced dataset. The data was selected from songs featured on Spotify's genre-specific playlists and downloaded using the Spotify API.

3.2. Preprocessing

First, we normalized the audio data for each song to remove differences in the baseline volume at which different pieces are recorded, which does not affect their genres.

Raw audio is difficult to work with since it contains too many data points (22,500 per second) to be computationally feasible for most neural networks. It would take too long to train, and the data's detail would make pattern recognition difficult without a prohibitively large model. Thus, we tested two more compact forms of data representation: Fourier-transform coefficients and Mel-frequency cepstral coefficients (MFCC).

Performing Fourier transform involves breaking the audio sample into small segments (~0.1 seconds), and taking a Fast Fourier Transform (FFT) of each segment. The resulting Fourier coefficients vectors were stacked along the time axis to form a time-series matrix of Fourier coefficients, which can be treated like an “image” when training. The FFT was performed using a Numpy function [1].

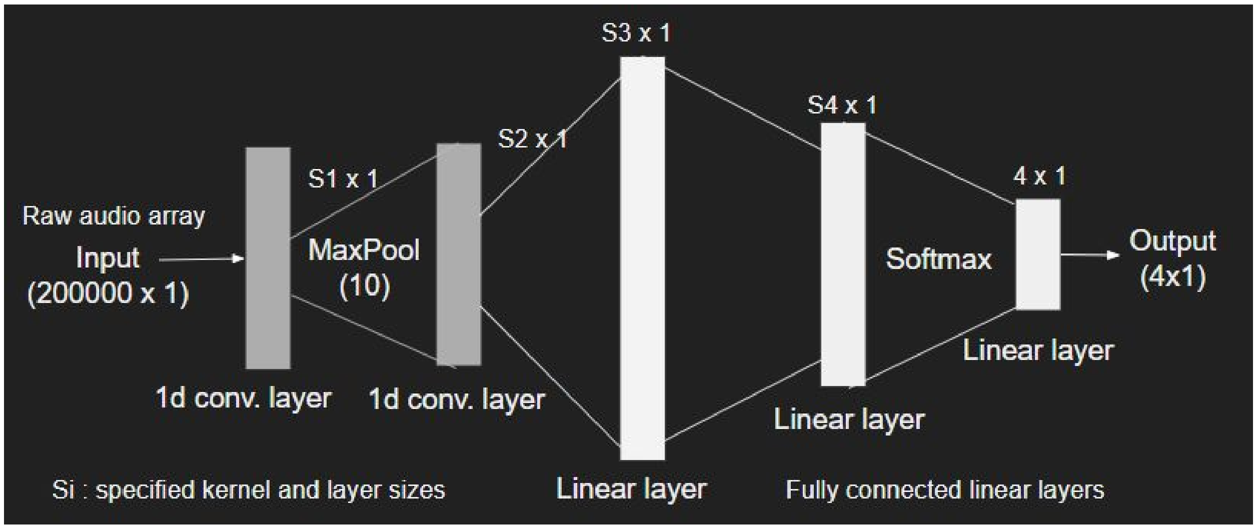
Performing the MFCC transform involves the following steps [2]:

1. Take the Fourier transform of each segment of audio
2. Map the powers of the spectrum obtained above onto the Mel-scale, using triangular overlapping windows
3. Take the logarithms of the powers at each of the Mel-frequencies
4. Take the discrete cosine-transform of the list of Mel-log powers
5. The MFCCs are the amplitudes of the resulting spectrum

This process was implemented using the Librosa library (a toolset designed for sound processing) [3]. We tried MFCC because according to [2], it is an industry standard for audio processing, and the author noted that it leads to significant accuracy improvements.

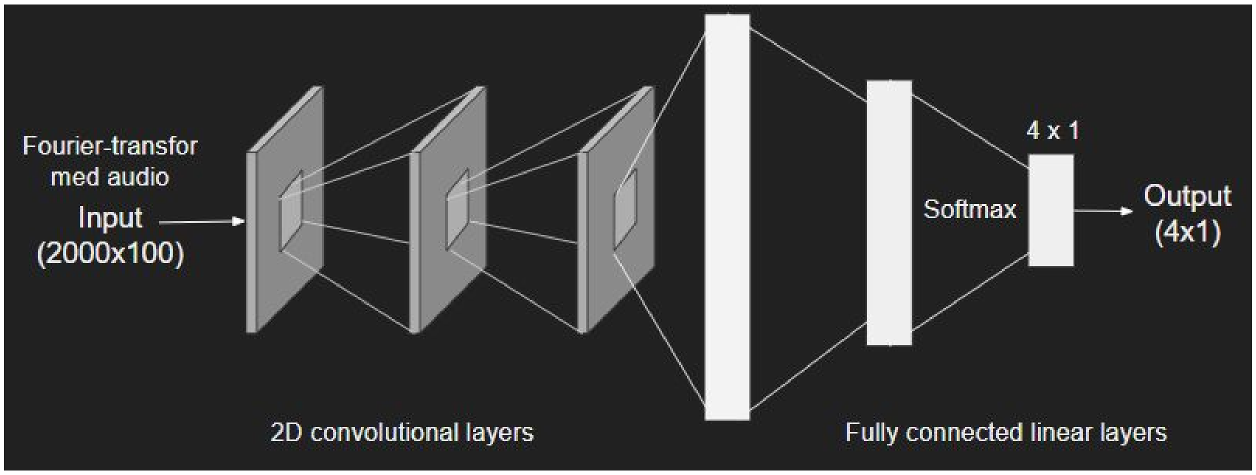
4. Machine Learning Models

4.1. 1D CNN



We initially attempted training on normalized audio data using a 1D-CNN with 2 layers and 3 fully-connected linear layers. However, the challenges mentioned in Section 3.2 were insurmountable. At first, since each audio vector was 200,000 elements long and we theorized that genre-specific ‘features’ of music are at least 0.1 seconds long, we made the kernel sizes very large (>1000) so that they could capture relevant musical features. However, at these sizes our Google Cloud instance would use up extremely large amounts of memory (> 80 Gb) and take prohibitively long to train. We attempted using smaller kernel sizes, using fewer kernels, and reducing the number of layers to make the model computationally feasible. However, all of these configurations led to underfitting: both test and validation accuracies remained near equilibrium regardless of training time.

4.2. 2D CNN

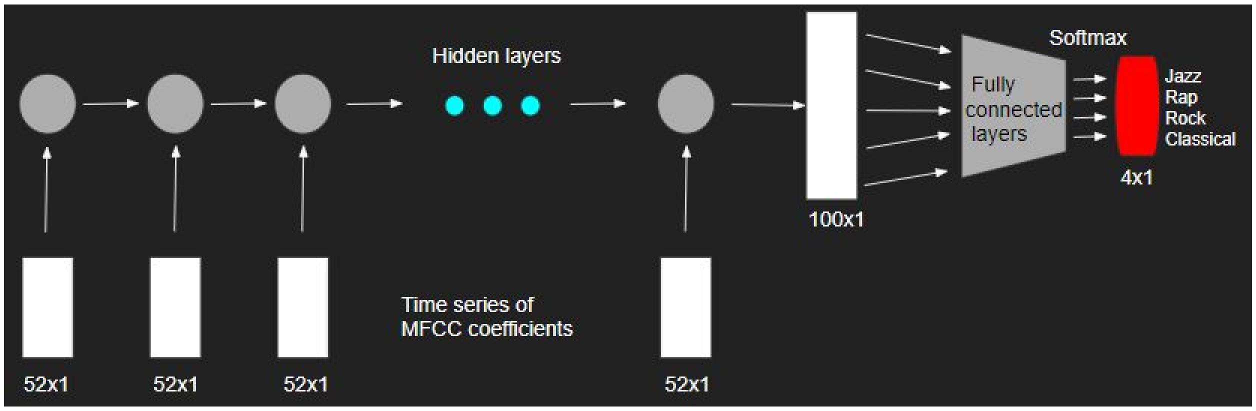


We then trained on time-series matrices of Fourier-transformed audio using a 2D-CNN, treating each matrix as an image. Following the advice of several research papers [4], we hoped that there might be patterns between subsets of Fourier coefficients across time that could be easily identified. This model contained 2-4 convolutional layers and 3-4 fully-connected linear layers. Three different kernel configurations were tested:

1. Square kernels of varying sizes
2. Thin rectangular kernels moving only along the direction of the time series designed to capture features across each segment along the time series
3. Thin rectangular kernels along the direction of audio across time to capture features along each segment

The first and second configurations resulted in underfitting as the kernels did not capture any features. Conversely, the third configuration resulted in overfitting as the training accuracy approached 80% while the validation accuracy decreased to less than equilibrium (<25%). Implementing overfitting reduction methods such as batch normalization and dropout did not improve the results.

4.3. Multi-layer GRU



In this model, MFCC-transformed data is first passed through a GRU. The GRU has an embedding dimension of 52, corresponding to the number of MFCC coefficients per time segment of audio, and a hidden dimension of 100. It is 100 layers deep, to accommodate 100 time segments included in our input. MFCC coefficients are inputted to the GRU in chronological order, and the GRU's output is fed into a single linear layer with 4 outputs and a softmax activation function. Each linear layer output represents the probability that our song is one of the four genres of interest. We hoped that the network's recurrent structure would allow us to account for the sequential patterns inherent to music data. We also hoped that the long-term memory of GRUs would allow the model to glean insight from longer patterns not perceivable by CNNs.

5. Methods and Results

5.1. Training methods

The following hyperparameters were optimized during training:

- All Models:
    - Batch size
    - Learning rate
    - Number of epochs
    - Number of final fully connected layers
- CNN Specific:
    - Kernel size
    - Number of kernels

    - Kernel stride

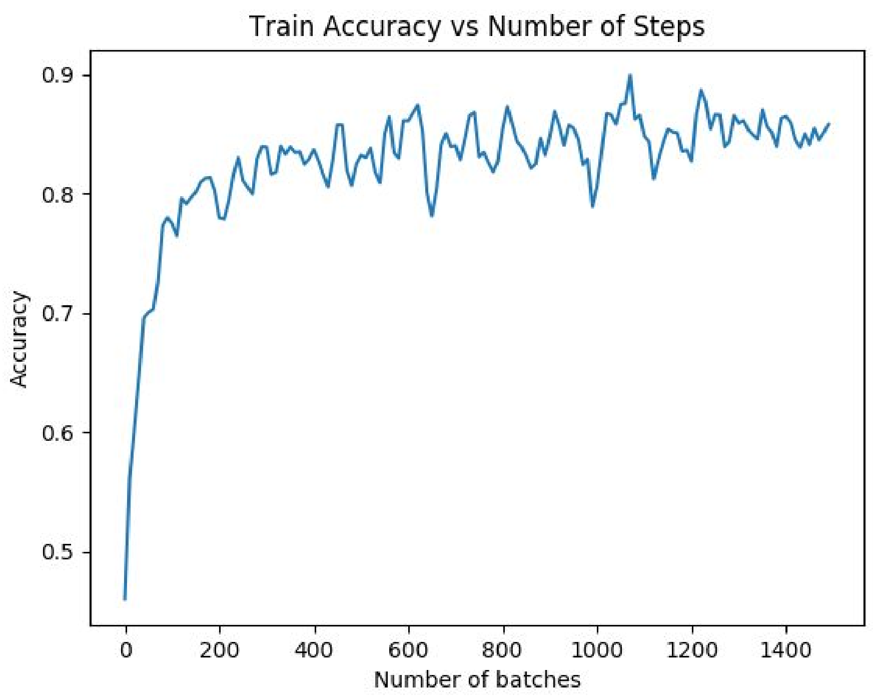
- RNN Specific:
    - Number of hidden layers
    - Hidden dimension size
    - Embedding dimension size

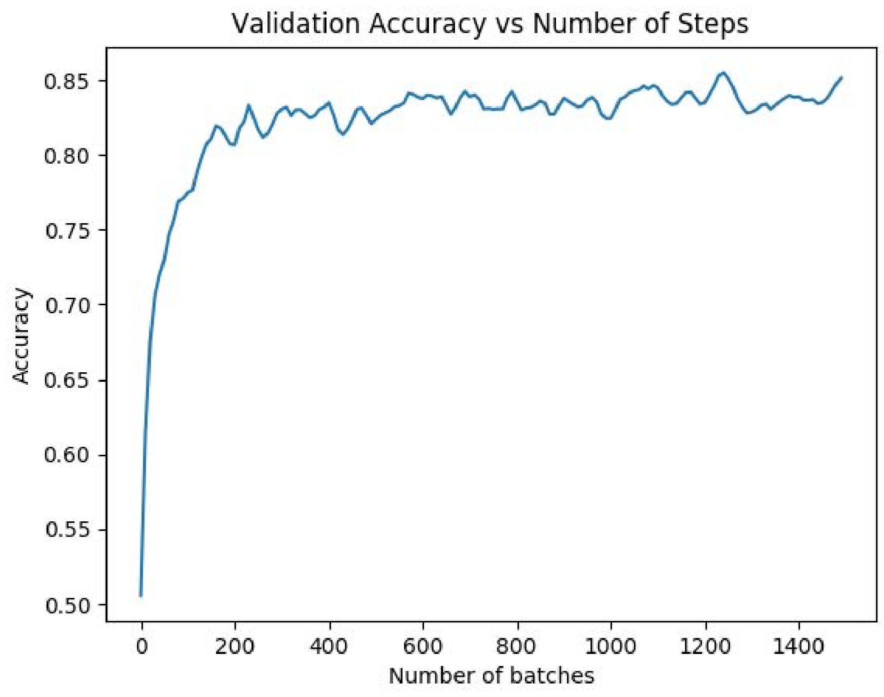
Our training methodology was a hybrid between grid search and personal intuition. We would independently perform range-constrained grid search on a limited set of hyperparameters and record the results. We would then frequently meet to assess current problems with the model and brainstorm on how to correct these issues and improve its performance. We would then iteratively re-train the model using our new ranges for hyper-parameters and architecture.

All training took place on GPUs on either our Google Cloud instance or aUToronto's (UofT's self-driving car team's) development server.

5.2. Training and validation results

For the multi-layer GRU RNN model, the training accuracy approached approximately 85% and the validation accuracy approached approximately 83% after 200 batches (2 epochs). Since these final accuracies are much greater than random chance (25%), the model is not underfitting. Moreover, since training and validation accuracies increase at roughly the same rate and final training accuracy is slightly higher than final validation accuracy, the model is not overfitting.





5.3. Test results

Our test set consisted of audio samples of 160 songs from each of our four genres of interest (640 total) selected from Spotify. The results are as follows:

Genre	Classical	Jazz	Rap	Rock	**Recall**
Classical	107	49	3	1	66.9%
Jazz	33	78	10	39	48.8%
Rap	0	4	150	6	93.8%
Rock	2	7	23	128	80.0%
Precision	75.4%	56.7%	80.6%	73.5%	

Overall accuracy: 72.3%

Test set accuracy is approximately 10% lower than validation accuracy, implying that our model may have overfitted slightly. Alternatively, it could imply that our test data was flawed. This is plausible since we forgot to create a test set from our originally collected data, and only did it after training. We selected test set songs that were not included in our training or validation data, which forced us to collect these songs from rather obscure Spotify playlists. These playlists may have included songs from esoteric sub-genres not adequately present in the training / validation data, reducing model accuracy. Despite this potential flaw, our model still performs very well - over 9% better than the current academic standard for 4-genre classification of 63.75% [2].

Our model is able to accurately recognize rap and rock music, with recall values of 93.8% and 80% respectively. However, it is much less adept at recognizing classical and jazz music, with recall values of respectively 66.9% and 48.8%. Rap, rock and classical music have roughly similar precision values near 75%, while jazz has a much lower precision of 56.7%.

These relative differences in performance between genres may be explained by the qualitative differences in their distinctiveness. Rap and rock music tend to have very distinctive sounds and vocal stylings, such as autotune and the electric guitar, and thus are easier for the model to differentiate. Conversely, classical and jazz music are both typically instrumental and use similar instruments, causing the model to easily confuse them and thus cause worse performance with both genres. This partially explains our results; however, since jazz is also often misclassified as rock, there clearly must be other reasons for genre-specific performance discrepancies.

Furthermore, the model is able to recognize the ‘ambiguity’ of a song’s genre through its level of confidence in its predictions. For instance, the model predicts that Eminem’s “Killshot”, which has a distinctively hip-hop sound, is rap with a confidence of 91%. Conversely, the model predicts that Linkin Park’s “Numb”, considered a hybrid of rock and rap, is rock with a confidence of 70% and rap with a confidence of 30%

6. Ethical Issues

One ethical issue resulting from automatic genre classification is that it may be used to regulate or censor certain types of music. Music from different genres has been used throughout history as a medium for publicly sharing socio-political-cultural messages. These messages may go against the interests of those in power. With a genre classification system, music containing offending messages can be easily identified and this information can be used to suppress or encourage its popularity and exposure according to the interests of those in power. This threatens the democratic right to freedom of expression.

We are also aware of the potential of our automatic classifier to replace jobs in the music industry which involve music classification (for example, playlist curators at Spotify). However, given the small number of workers in the music industry, we do not believe that this technology will meaningfully change employment patterns.

7. Key Learnings and Reflections

- Data is key!
    - We were forced to take the ‘pop’ category out of our classification system since results were very poor. We did not have enough data to generalize the genre’s diverse patterns, and many songs Spotify tags as ‘pop’ blend into other genres (like rap). Better quantity and quality of data from this genre would have resolved this issue.

- Machine learning = trial-and-error + intuition
    - Sometimes the best way to achieve success is to try something new. This mindset encouraged us to try using the RNN

    - Intuition about new solutions may guide which ones to try. For instance, we knew that RNNs could easily process sequential data and thus might work well for song data.

- Sometimes smaller is better
    - Our large CNN models were very ineffective, but smaller RNN models worked extremely well

References

[1] "Discrete Fourier Transform (numpy.fft) — NumPy v1.15 Manual", Docs.scipy.org, 2018. [Online]. Available: https://docs.scipy.org/doc/numpy-1.15.0/reference/routines.fft.html. [Accessed: 01- Dec- 2018].

[2] "mlachmish/MusicGenreClassification", GitHub, 2018. [Online]. Available: https://github.com/mlachmish/MusicGenreClassification. [Accessed: 01- Dec- 2018].

[3] "Core IO and DSP — librosa 0.6.2 documentation", Librosa.github.io, 2018. [Online]. Available: https://librosa.github.io/librosa/core.html. [Accessed: 01- Dec- 2018].

[4] "Machine Learning with Signal Processing Techniques", Ahmet Taspinar, 2018. [Online]. Available: http://ataspinar.com/2018/04/04/machine-learning-with-signal-processing-techniques/. [Accessed: 01- Dec- 2018].