# *Music Generation and Genre Classification*

MAIS 202 Project Deliverable 3 Jad Hamdan

**Problem Statement:**
My project is meant to explore some of the many intersections between music and machine learning. This translated into two main problems to solve: classifying music by genre and generating music that sounds both human and pleasing.

I relied on a different dataset for each problem:

1- For the first problem, I used the GTZAN [1] dataset, comprised of 10 different folders each containing 30 second audio clips of a given genre.

2- For the second problem, I used the Nottingham folk music dataset [2]. Its data's format was crucial to my approach to music generation and, in short, massively simplified the whole process (more details later).

## Part 1: Classification by Genre:

### Initial attempt:

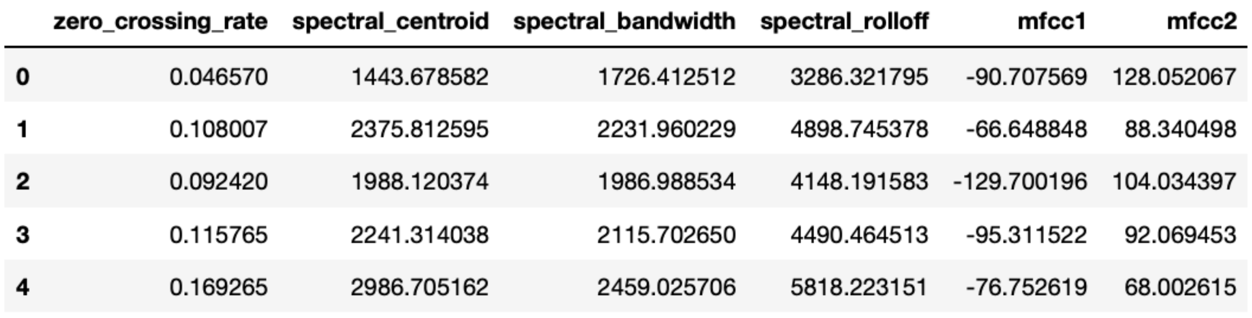
Shortly after deciding what my final project was going to be, I stumbled upon Librosa. This python library gave me various tools for the manipulation of audio files, one of which was particularly interesting to me: feature extraction.

I thus began the process by extracting the following features from all of my files...

- Zero crossing rate:
  - Rate of sign changes along signal. Usually higher when we're dealing with more percussion
- Chroma Frequencies:
  - Spectrum is projected onto 12 bins (representing the 12 semitones on an octave
- Spectral Centroid:
  - The Music's "center of mass". For example, a jazz composition will reach its climax towards the middle of the piece, which is where its spectral centroid will be located
- Spectral Bandwidth:
  - Interval in which the signal is at an intensity that's greater than half its maximum value
- Spectral Rolloff:
  - Frequency below which a specified percentage of our total spectral energy lies

- Mel Frequency Cepstral Coefficients:
  - Describe overall shape of spectral envelope

… and stored said data in a csv file [3].



| | zero_crossing_rate | spectral_centroid | spectral_bandwidth | spectral_rolloff | mfcc1 | mfcc2 |
|---|---|---|---|---|---|---|
| **0** | 0.046570 | 1443.678582 | 1726.412512 | 3286.321795 | -90.707569 | 128.052067 |
| **1** | 0.108007 | 2375.812595 | 2231.960229 | 4898.745378 | -66.648848 | 88.340498 |
| **2** | 0.092420 | 1988.120374 | 1986.988534 | 4148.191583 | -129.700196 | 104.034397 |
| **3** | 0.115765 | 2241.314038 | 2115.702650 | 4490.464513 | -95.311522 | 92.069453 |
| **4** | 0.169265 | 2986.705162 | 2459.025706 | 5818.223151 | -76.752619 | 68.002615 |

Fig [3]: CSV file representing all the extracted features for each sample.

This allowed me to use traditional classification algorithms on my dataset with SciKit Learn:

1- Random Forest: collection of decision trees (bagging method), where each tree learns what features to split on via slightly different, randomized subsets of the original feature space.

2- Linear Support Vector Machine: classifies by finding a hyperplane that has the maximum margin (distance) between data points of two classes.

3- Boosting (AdaBoost): sequentially built classifier by feeding the output of one classifier into the next, re-weighting the training examples from the previous classifier.

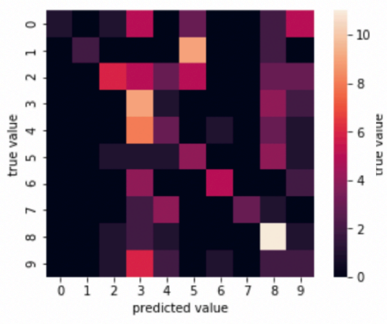
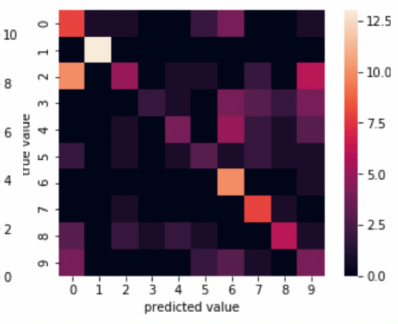
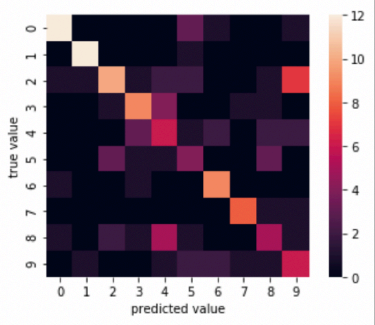
| Algorithm | AdaBoost | SVM | Random Forest |
|---|---|---|---|
| Confusion Matrices |  |  |  |
| Accuracy | 0.34 | 0.42 | 0.54 |

Fig [4]: Confusion Matrices and Accuracy for 3 models

This led to rather disappointing results. As seen by the table in Fig. 4 [4], my preliminary results were barely reaching 60%, with Random Forest being the most successful algorithm of the three and reaching a mere 54%.
On a more positive note, a traditional feed forward network trained on the same data achieved an accuracy of 67%. Albeit better, this result is still far from sufficient.
I hence came to the conclusion that to achieve better results, I'd most likely have to change my approach.

## Second Attempt:

I figured out what to do next upon finding an interesting paper by Z. Nasrullah and Y. Zhao of the university of Toronto [5]. The paper describes an attempt to classify music by its artist (a task more or less similar to mine), and while doing so explains the problem with my previous attempt. "Past research favored vector summaries of frequency content in an audio window, specifically Mel-frequency cepstral coefficients, because they summarize both key and timbral information in an audio track. The problem with this and other established approaches is that, in summarizing a short duration of audio into a vector, temporal structure is lost."
To solve that issue, the paper cites recent works by Choi et al. and Cakir et al. where audio spectrograms are used in conjunction with deep learning to produce superior results.

Spectrograms are a representation of frequency content over time, found by taking the absolute value of the Short-Time Fourier Transform of a signal (see Figure [6] for an example). The latter can be calculated with the following mathematical formula:

$$X(m, w) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-jwn}$$

*Where x[n] and w[m-n] describe the signal and window function.*

Thankfully, the previously mentioned Librosa library takes care of this for us and makes extracting a spectrogram from an audio file relatively simple. After that, each audio file in my dataset was effectively "summarized" by an image.

We thus have an image classification problem to solve, for which Convolutional Neural Networks (CNNs) are the de facto algorithm to use. I thus defined a model with a very simple architecture [7] and trained it, hoping to see improvement.

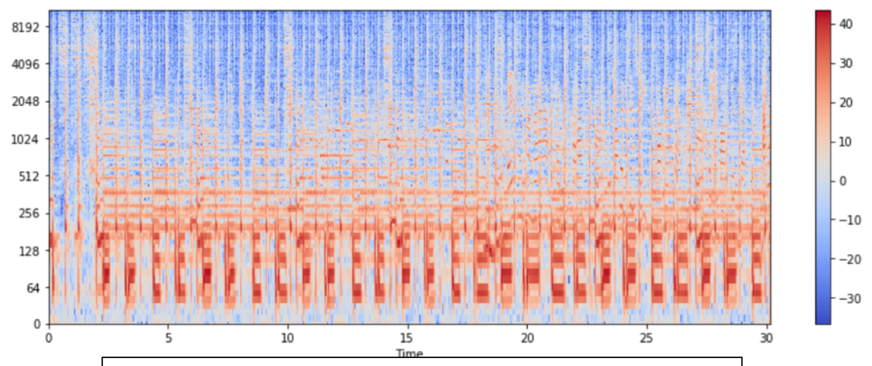
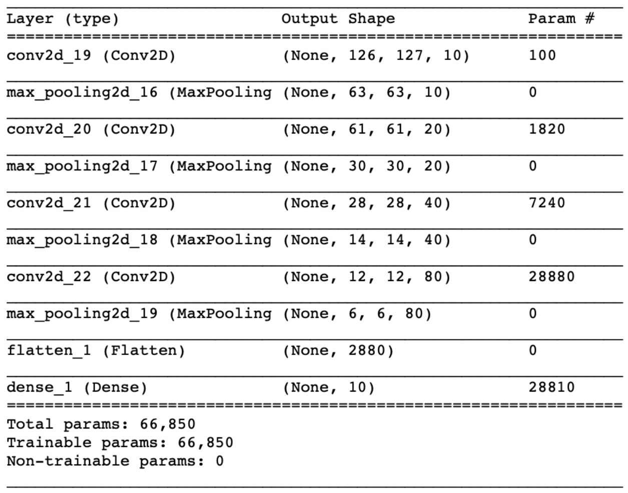


Fig [6]: Spectrogram for a Disco Sample

The model is comprised of a convolutional layer with a filter of size (3,3), followed by a Max Pooling layer with a pool size of (2,2). This is repeated three times before adding a flatter layer and finally a dense layer (output) layer.

*This method proved to be much more effective!* After a mere 10 EPOCHS, I reached 78% validation accuracy (Fig [8])!

Attempting to surpass 90%, I defined a much more complicated model with considerably more parameters to train. The model is currently training and its results will be reported as soon as the training ends



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 126, 127, 10)      100
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 63, 63, 10)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 61, 61, 20)        1820
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 30, 30, 20)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 28, 28, 40)        7240
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 14, 14, 40)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 12, 12, 80)        28880
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 6, 6, 80)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2880)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                28810
=================================================================
Total params: 66,850
Trainable params: 66,850
Non-trainable params: 0
_________________________________________________________________
```

Fig [7]: Simple CNN Model Architecture

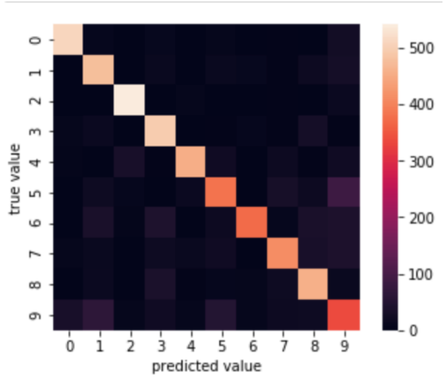
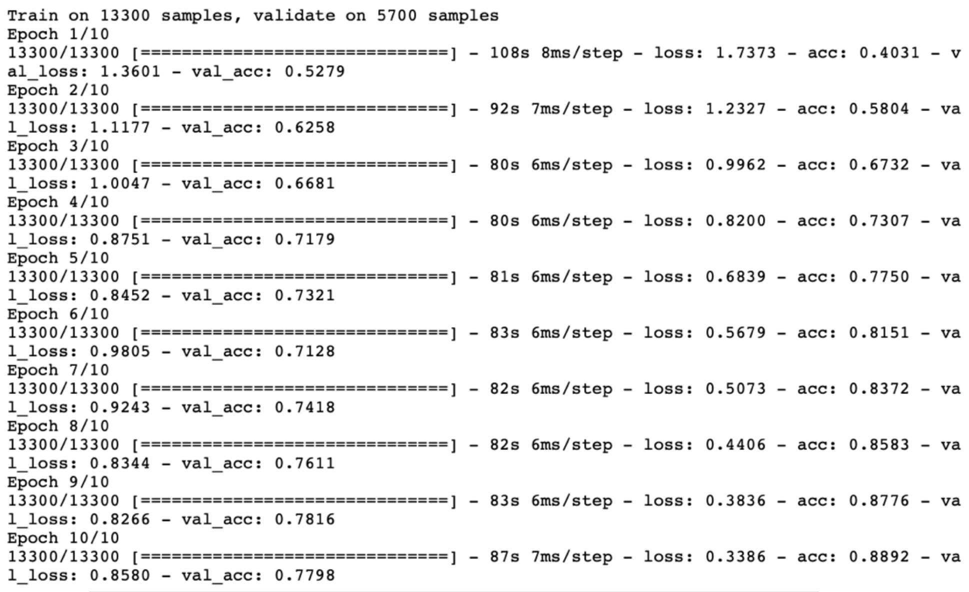


Fig [8.a]: Early Results (Confusion Matrix)



```
Train on 13300 samples, validate on 5700 samples
Epoch 1/10
13300/13300 [==============================] - 108s 8ms/step - loss: 1.7373 - acc: 0.4031 - v
al_loss: 1.3601 - val_acc: 0.5279
Epoch 2/10
13300/13300 [==============================] - 92s 7ms/step - loss: 1.2327 - acc: 0.5804 - va
l_loss: 1.1177 - val_acc: 0.6258
Epoch 3/10
13300/13300 [==============================] - 80s 6ms/step - loss: 0.9962 - acc: 0.6732 - va
l_loss: 1.0047 - val_acc: 0.6681
Epoch 4/10
13300/13300 [==============================] - 80s 6ms/step - loss: 0.8200 - acc: 0.7307 - va
l_loss: 0.8751 - val_acc: 0.7179
Epoch 5/10
13300/13300 [==============================] - 81s 6ms/step - loss: 0.6839 - acc: 0.7750 - va
l_loss: 0.8452 - val_acc: 0.7321
Epoch 6/10
13300/13300 [==============================] - 83s 6ms/step - loss: 0.5679 - acc: 0.8151 - va
l_loss: 0.9805 - val_acc: 0.7128
Epoch 7/10
13300/13300 [==============================] - 82s 6ms/step - loss: 0.5073 - acc: 0.8372 - va
l_loss: 0.9243 - val_acc: 0.7418
Epoch 8/10
13300/13300 [==============================] - 82s 6ms/step - loss: 0.4406 - acc: 0.8583 - va
l_loss: 0.8344 - val_acc: 0.7611
Epoch 9/10
13300/13300 [==============================] - 83s 6ms/step - loss: 0.3836 - acc: 0.8776 - va
l_loss: 0.8266 - val_acc: 0.7816
Epoch 10/10
13300/13300 [==============================] - 87s 7ms/step - loss: 0.3386 - acc: 0.8892 - va
l_loss: 0.8580 - val_acc: 0.7798
```

Fig [8.b]: Early Results (Validation Accuracies)

With classification out of the way, it was time to focus on the second part of my project: Music Generation.

## Part 2: Music Generation

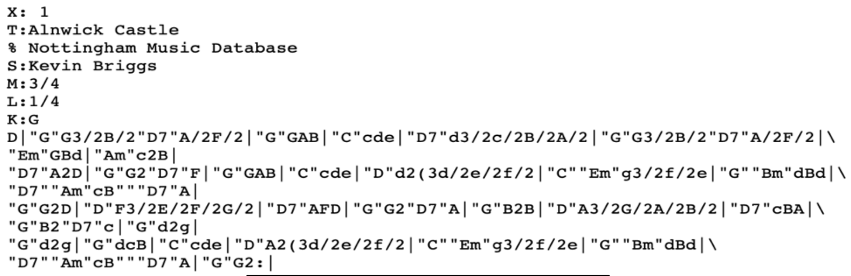
At first, I had no clue how to approach the problem. I found many different ways of tackling that problem using all forms of data online. Some used simpler techniques while others made use of particularly complicated models, such as C-RNN-GANs.

I decided to focus on single instrument music to simplify the task and try to find an "easy way out" by using data whose format made everything easier. I found that with ABC Notation, the way music is written in the Nottingham Music dataset I mentioned previously.

The notation is simple as seen by the following song, "Alnwick Castle" (Fig [9]):

Vertical bars divide musical measures. Single letters represent different keys while letters in between quotations represent a chord.

Each song starts with the same header, that specifies its title, artist, key, etc.



```
X: 1
T:Alnwick Castle
% Nottingham Music Database
S:Kevin Briggs
M:3/4
L:1/4
K:G
D|"G"G3/2B/2"D7"A/2F/2|"G"GAB|"C"cde|"D7"d3/2c/2B/2A/2|"G"G3/2B/2"D7"A/2F/2|\
"Em"GBd|"Am"c2B|
"D7"A2D|"G"G2"D7"F|"G"GAB|"C"cde|"D"d2(3d/2e/2f/2|"C""Em"g3/2f/2e|"G""Bm"dBd|\
"D7""Am"cB""D7"A|
"G"G2D|"D"F3/2E/2F/2G/2|"D7"AFD|"G"G2"D7"A|"G"B2B|"D"A3/2G/2A/2B/2|"D7"cBA|\
"G"B2"D7"c|"G"d2g|
"G"d2g|"G"dcB|"C"cde|"D"A2(3d/2e/2f/2|"C""Em"g3/2f/2e|"G""Bm"dBd|\
"D7""Am"cB""D7"A|"G"G2:|
```

Fig [9]: ABC Notation

Why does this make things easier?

*Having music presented in a text format turns this problem into a much simpler text generation problem.*

Generating text from a given corpus is a textbook application of LSTMs (Long Short-Term Memory Networks), a variant of Recurrent Neural Networks that excels at learning dependencies.

And so, I used Keras to implement a model comprised of two LSTM layers separated by Dropout layers (0.2). Finally, a dense layer with softmax activation was used to get our output values (characters).

I vectorized a single text file containing all of the dataset's songs (separated by a few carriage returns) used the latter as input along with creating two dictionaries to map to and from ABC notation to integers.

Upon training and generating samples, I noticed the model was producing various valid "sentences" but was struggling with learning the ABC notation header. EPOCH after EPOCH, I wasn't seeing any valid "song".

I thought that maybe the dataset was too large to be in a single file. The model was having trouble discerning between different songs, and so I tried significantly reducing the number of songs used as input. I thus only took the "Waltzes" subset of my dataset and used that to train the LSTM.

Upon surpassing 20 epochs, this attempt proved to be successful as I started seeing valid files and eventually cherry picked some results. The first complete result I had was a track the model name "Fyeaoon" (Fig [10]). I took said track and, using an online converter, managed to obtain sheet music and an audio file.

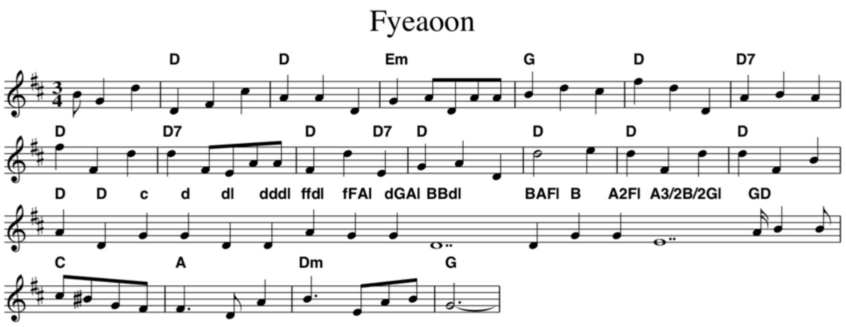


Fig [9]: Sheet Music of the LSTM's First Complete Track

To my surprise, the midi file produced from the sheet music in Fig[9] sounded rhythmically correct, human and, quite frankly, not too bad (musically speaking).
Not exactly a classical masterpiece, but definitely satisfying enough to say mission accomplished.

---

References:

1- GTZAN Dataset: http://marsyas.info/downloads/datasets.html
2- Nottingham Folk Music Dataset: http://abc.sourceforge.net/NMD/
5- Z. Nasrullah and Y. Zhao, "Music Artist Classification with Convolutional Recurrent Neural Networks".

---

## Final Demonstration Proposal: Poster Presentation

Poster Structure: 2 part poster (two different problems).
Problem: Can machine learning algorithms classify music by genre?
Hypothesis: This should be achievable using one of the many ML algorithms available today.

1- First attempt: Feature Extraction + Classification Algorithms
   a. Doesn't exactly confirm our hypothesis seeing as the results obtained were subpar.
   b. We'll use our best performing algorithm from this approach (Feed Forward Neural Net) as a baseline, making 67% accuracy the score to beat
2- Second attempt: Training Convolutional Neural Nets on Audio Spectrograms
   a. Achieves close to 80% accuracy after only a few epochs, hence confirming our hypothesis.

Problem: Can machine learning algorithms be used to create music?
Hypothesis: By using networks that excel at learning dependencies, we should be able to compose simple music.

Question: How do we go by generating audio in the first place?

Method:

1- Use a dataset that simplifies the task, turning it into a text generation problem.
2- Train and use LSTM Networks to generate our music.

Conclusions:

First Problem: Our hypothesis was confirmed and a second approach to the problem managed to defeat baseline results by a considerable amount.

Second Problem: Hypothesis also confirmed.