CS230

Music Genre Classification with CNN Evaluation on Mel Spectrograms and MFCCs

Tom Nguyen
Department of Computer Science
Stanford University
anhn@stanford.edu

Alice Zhang
Department of Computer Science
Stanford University
alicebz@stanford.edu

Joseph Zhang
Department of Computer Science
Stanford University
josephz@stanford.edu

ABSTRACT

As genre classification becomes more integrated into modern recommendation systems, it becomes increasingly important to improve their performance by exploring newer avenues of research and how they affect accurate accuracy. In this paper, we explore the idea of explicitly combining MFCC features and spectrograms of each track to our classification CNN and evaluating its resulting performance, as compared to a CNN that only takes in spectrogram or MFCC inputs. We saw a major decrease in accuracy with this new model, in which the CNN showed a performance of 0.28 compared to a hypertuned spectrogram CNN with 0.60 and a simple MFCC CNN with 0.54.

1 INTRODUCTION

When it comes to browsing, searching, and organizing large collections of music, classifying a song's genre is generally a convoluted task. With the rise of music recommendation and organizational systems, however, its necessity becomes increasingly crucial to acknowledge and perform. Take Spotify, Apple Music, and even Youtube as well-known examples: all three have internal systems that can recommend other songs based on a listener's interests, with genre being one of the valid descriptors to consider. Spotify even dedicates a portion of its computational power to creating playlists of specific genres to present to a listener when they search for that genre in the search bar. To further support Spotify's endeavor, one recent study in 2019 found that end users tended to prefer automatic recommendations that weighed a track's genre more than recommendations based on other sorts of similarity methods [5].

2 RELATED WORK

With this motivation in mind, we decided to explore the effects and consequences of combining MFCC features and spectrograms together rather than as separate inputs as we predicted that, since each is a different summary of the features of a specific track, combining the results from both will in theory allow the model draw more conclusions from new input feature. This in theory increases performance accuracy for genre classification. Previous research for music classification, in general, falls into one of three categories: classification based on intrinsic low-level features and computations (Deep Neural Networks, LSTMs), numerical features (random forest classifiers), or image and

CS230: Deep Learning, Fall 2022, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

spectrogram recognition (CNNs, RNNs). Several papers focused on transfer learning applications [1, 2] and utilized Deep Neural Networks trained with limited amounts of metadata and MFCC features or other low-level handpicked features. Pitfalls with these transfer learning applications identify how the amount of shared information correlate to a decrease in in performance. Thus, we aim to utilize minimal shared information between different architectures. There are also many papers that researched the use of Extreme Gradient Boosting (XGBoost) and forest classifiers to examine numerical features [3], as well as many other papers utilized spectrogram classification as a baseline, in which they created images based on audio input and classified accordingly [4]. We aim to incorporate some of these ideas as motivations. However, pitfalls with these ideas include the lack of depth with information. As these papers are limited to only looking at spectrogram data, they lack depth in evaluating further numerical representations of the data, which we will hope to accomplish by integrating models on MFCCs.

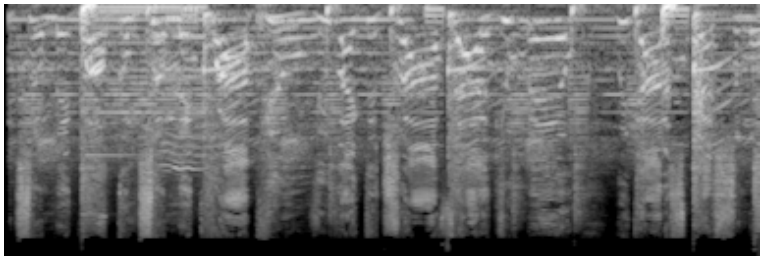
Although some of the mentioned papers consider and compare the losses across various techniques, none of them have **explicitly combined the multiple techniques** to feed into a larger system for prediction evaluation. Instead, many opt to augment the data in different ways, with the addition of various pieces of metadata being the most popular avenue of augmentation. This is where our approach comes in. To further explore the concept of the purposeful combination of multiple feature descriptors, we decided to combine MFCC features with that of their respective track's spectrogram descriptors and analyze how this combination affects classification accuracy in our resulting model.

3 DATASET

We are using the medium sized dataset from the Free Music Archive (FMA) dataset (https://github.com/mdeff/fma) which contains 25,000 tracks of 30s with 16 unbalanced genres. Albeit useful for the majority of our programming, the most notable catch to this dataset was the incredible discrepancy between the number of songs in each of the 16 genres, in which we underestimated its imbalance and therefore prolonged our process of hypertuning. For example, the rock genre had roughly 6000 songs whereas the Soul-RnB only had around 154 songs. Some mp3 files were also found to be corrupted and had to be removed during preprocessing, leaving 24,985 songs.

From this dataset, we computed grayscale Mel-spectrograms and MFCC features during data preprocessing. Despite most relevant research using colored Mel spectrogram images as their classification inputs, we decided to explore the use of grayscale spectrograms as a computationally less expensive alternative to colored spectrograms and evaluate its accuracy under faster conditions (in which we predicted grayscale to be an effective alternative since research showed that a typical neural network only needed raw data over Mel-spectrogram inputs. As many of these resulting spectrograms resulted in different dimensions, we specifically plotted the spectrograms with 384 time steps, a window of 128 bins, and 512 samples per time step in order to properly consolidate the data for training (giving us spectrogram dimensions of 128 x 385). To match these new spectrogram dimensions, we subsequently also used 512 samples per time step and 384 time steps for each MFCC vector.

Our Mel-spectrograms were computed using Fast Fourier Transform (FFT) converted to decibels and mapped onto the Mel scale. We have provided an example input below, where the x-axis is the number of time steps and the y-axis is the window we specified:



This dataset had precomputed MFCCs that we implemented into our baseline CNN, but to combine a track's MFCC with its spectrogram as intended we needed to create new MFCCs with dimensions that were broadcastable with our spectrogram dimensions. In order to make our own MFCC features, then, we converted an audio track using the Mel scale, took a logarithm of this Mel representation, and fed it through a Discrete Cosine Transformation, creating a spectrum over Mel frequencies as

opposed to time (which were how we created our spectrogram representations). Since we did not make a classifier using just MFCC features, we decided to save these as .npy files that were to be concatenated with our Mel-spectrograms, each of which had dimensions of 20 x 385. Both feature descriptors were produced using `librosa`. Due to the amount of data we had, we decided to use an 80/10/10 ratio to split the data into a training, test, and validation set respectively.

4 METHODS

Our main method was utilizing a convolution neural network to extract visual features from various images of extracted spectrogram and MFCC features. From our literature review, our novelty comes in the 2 forms, the usage of both MFCC and Mel-spectrograms as input features, and the usage of grayscale Mel-spectrograms instead of colored spectrograms. We had found literature on using MFCC and Mel-spectrograms separately but not together as input features for a combined input. Additionally, we found a paper on utilizing gray-spectrograms instead of colored spectrograms, but in the paper they used an RNN architecture rather than a CNN. In terms of our specific hyperparameter tuning and neural network construction this is discussed in the later section of the report.

5 EXPERIMENTS

5.1 BASELINE

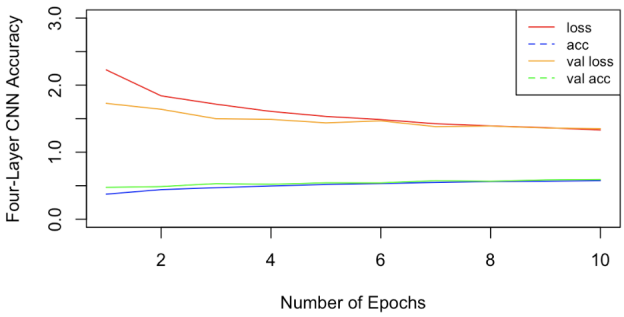
We developed four baseline models for initial classification, which all took in MFCC inputs: one completely random model with completely random one-hot vector predictions, one pseudo-random model with pseud-random one-hot vectors predictions based on the weighted genre distribution of the training set, one simple, fully connected neural network layer pair utilizing categorical cross entropy loss and an Adam optimizer, and one simple convolutional neural network (CNN) utilizing stochastic gradient descent with momentum and categorical cross entropy loss.

The first baseline ignored the training set altogether while the second baseline only examined the genre distribution and ignored any track features. Our third and fourth baselines, albeit focused on determining a relationship between training features and genre, did not have the capacity to develop more complex boundaries or relationships without more layers. After running these baselines, we correspondingly received accuracies of 0.06 with the completely random assignments, 0.17 with the pseudo-random assignments, 0.54 with the fully connected layer pair, and 0.50 with the CNN.

5.2 HYPERPARAMETER and MODEL ARCHITECTURE TUNING MEL-SPECTROGRAM CNN

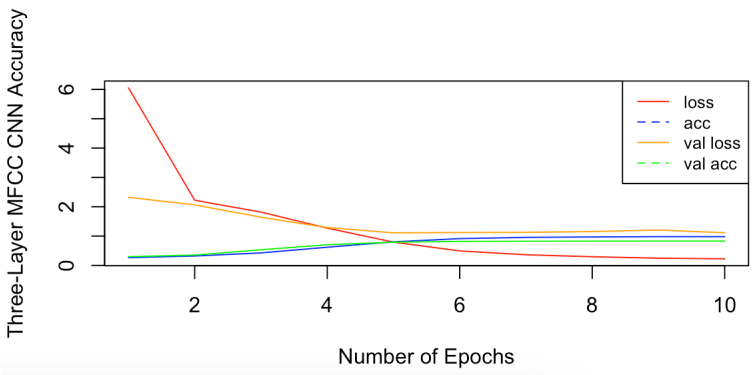
To begin, we first created a CNN on the grayscale Mel-spectograms. The model first exhibited strange behavior where the testing accuracy from subsequent epochs would start from accuracies of greater than 10% increase to over 30% and then proceed to decrease to under 5%. We realized that the input dimensions for the CNN were flipped, so after fixing that issue we achieved much more reasonable results.

We began by creating a 2-layer CNN with 8 and then 16 filters each with a RELU activation function followed by a single linear layer with a softmax activation function, a learning rate = .01, and an Adams optimizer. With 10 epochs this achieved a testing accuracy of roughly 95% and a training accuracy of roughly 48%, exhibiting high variance. To address this variance issue we decided to add more layers but each subsequent layer would have a higher number of filters to reduce the amount of trainable parameters that the last linear layer had. We also implemented regularization with a dropout layer. Throughout this process we also used the validation to test the learning rate and found that .0001 was optimal. We ended up creating a 4-Layer CNN with 8, 16, 64, and 128 filters respectively, followed by an outer linear softmax layer. Before we fed the data into the linear layer, we included a dropout layer to help with the regularization. This gave us a much higher results with a training accuracy of 57% and a testing accuracy of 56%.

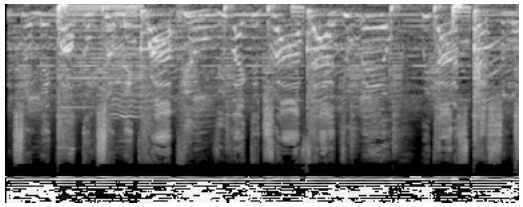


5.3 HYPERPARAMETER and MODEL ARCHITECTURE TUNING MFCC FEATURES CNN

The input data was first cleaned of corrupted files before being split into a 80/10/10 division of training, dev, and test sets. We used an Adam optimizer with a learning rate of 0.001 and measured categorical crossentropy loss. We created a three-layer CNN followed by a linear layer. The first Conv2D layer has a filter of 2 and a kernel size of (2,2). We then utilize a Conv2D layer with a filter size of 32 and a kernel size of (2, 2). We then expand to use a Conv2D layer with a filter size of 48 and a kernel size of (2, 2). We also use kernel regularizers and bias regularizers at 0.01, allowing us to prevent overfitting to the training set and correspondingly variance. Afterwards, we implement a max pooling layer with a size of (2,2). We then flatten the data before passing it into a softmax activation layer. We then explored hyperparameter tuning of the model as follows. We started with an initial one-layer model and gradually expanded the amount of layers we included. However, we started to realize that extensively increasing the number of layers forced our data to slow in both training speed and accuracy, which we theorized was a potential reflection of the vanishing gradients issue. In considering grid search for hyperparameters, we realized that the various hyperparameters would have different importances, and thus, we focused on tuning the filters and kernel sizes, which we theorized would map to the high level features that MFCCs reflected. At first, we started out with smaller filters and were able to get very high training accuracy but low testing accuracy. We thus applied weight decay as a regularizing factor and increased the number of filters to improve model robustness. Although we weren't able to fully eliminate the spread between the training accuracy and validation accuracies, in using mentioned hyperparameter tuning, we were able to dramatically increase the validation accuracy for a final value of 0.831 (compared to a training accuracy at 0.98).



5.4 HYPERPARAMETER and MODEL ARCHITECTURE TUNING MFCC-SPECTROGRAM INPUTS



For our combined model, we concatenated the data from both the Mel-spectrograms and the MFCC features vector along the various time steps. We then fed this into a model with an architecture of 2 layers with 8 and 16 filters each with max pooling followed by a singular linear layer with a softmax classifier. This achieved around 28% testing accuracy and after increasing and decreasing the layers, changing the learning rate, and the number of filters in various layers, we could not get the accuracy to improve significantly.

6 RESULTS AND DISCUSSION

6.1 ANALYSIS

Both our Mel-spectrogram CNN and our MFCC features CNN performed better than our baseline from our milestone, with the Mel-spectrogram CNN performing roughly 10% better and the MFCC features CNN performing around 30% better. While both MFCC features and mel-spectrograms utilize the Mel scale, we theorized that the reason our MFCC features CNN performed better can be attributed to the architecture of the model and the MFCC features themselves. Our combined model using both Mel-spectrograms and MFCC vectors as input features did not perform well, however, even doing worst than the baseline with a measly 28%.
A lot of models during our iterations exhibited high bias with their training accuracy being higher than the testing accuracies. This indicates that there is still possible bias that needs to be address. A possible solution to this problem would have been to use the bigger FMA dataset of around 100,000 examples, however this would also contain more genres of music. A possible explanation for this would be the unbalanced distribution of tracks for each generes. As previously mentioned in our dataset section, some generes had almost double of more examples in the dataset than other generes.

6.2 IMPLEMENTATION

Our initial Dataset: https://github.com/mdeff/fma
Our github repo: https://github.com/tnguyen2002/musicClassification Video:

7 CONCLUSIONS AND FUTURE WORK

As a conclusion, we realize that classification of a genre is a complex problem that expands just beyond mathematical representations of audio files. Additionally, We could also have evaluated how differing architectures such as RNNs could also be utilized in conjunction with our model. RNNs would fit well with our objective because they are effective when working with time series data.

Altogether, we have created a model that can classify songs into genres through evaluating their MFCC and spectrogram features, but this model lacks robustness in several ways, namely, that the model only classifies songs and does not show the possible intersection of genres or individual sub-genres that the songs could fall under. Additionally, all the models exhibits lackluster testing accuracy whose models could be better tuned with more time.

8 CONTRIBUTIONS

Tom Nguyen wrote most of the code base, besides the the sequential model for MFCC CNN and code for combining .npy files, handled all of the data processing of converting to gray-scale spectrograms

and MFCC feature vectors stored as .npy file,set up the githb repo, trained both the gray-scaled mel-spectrogram and combined models, wrote the analysis, methods, dataset, parts of the conclusion, 5.2, and 5.3. Alice Zhang wrote the script, edited our video, setup AWS, wrote the code for combining .npy files, helped write the code for MFCC feature model, and help write the literature review, intro, and abstract.. Joseph Zhang helped write the MFCC input model, wrote part of the literature review, part of the conclusion and 5.3.

REFERENCES

1) Franceschi, L., Niepert, M., Pontil, M., He, X. (2019, May). Learning discrete structures for graph neural networks. In International conference on machine learning (pp. 1972-1982). PMLR.

2) crowdAI. (n.d.). Crowdai/Crowdai-musical-genre-recognition-starter-kit. GitHub. Retrieved December 9, 2022, from https://github.com/crowdAI/crowdai-musical-genre-recognition-starter-kit

3) Kim, J., Urbano, J., Liem, C.C.S. et al. One deep music representation to rule them all? A comparative analysis of different representation learning strategies. Neural Comput Applic 32, 1067–1093 (2020). https://doi.org/10.1007/s00521-019-04076-1

4) Verma, S. (2019, August 31). Understanding input output shapes in convolution neural network | Keras towardsdatascience. Retrieved November 16, 2022, from https://towardsdatascience.com/understanding-input-and-output-shapes-in-convolution-network-keras-f143923d56ca

5) Adiyansjah, Gunawan, A. A., amp; Suhartono, D. (2019). Music recommender system based on genre using convolutional recurrent neural networks. Procedia Computer Science, 157, 99–109. https://doi.org/10.1016/j.procs.2019.08.146