Music Genre Classification of Audio Signals

Hakan Tekgul
University of Illinois Urbana-Champaign
Urbana, Illinois
tekgul2@illinois.edu

Raimi Shah
University of Illinois Urbana-Champaign
Urbana, Illinois
rsshah2@illinois.edu

ABSTRACT

ABSTRACT HERE!

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; **Reliability**;

KEYWORDS

Fault Tolerance, Reliability, Instruction Criticality, Embedded Systems

ACM Reference Format:
Hakan Tekgul and Raimi Shah. 2018. Music Genre Classification of Audio Signals. In *Proceedings of Machine Learning for Signal Processing Conference (CS 598 PS Fall 18').* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the past decade, with the introduction of technology that can store huge amounts of data, a large amount of musical data is increasingly available to public on different application platforms such as Spotify. As the number of musical data in our phones and Internet keeps increasing, there is a need to characterize each music track so that finding a specific song in a large archive of music would not be a problem. Musical genres are commonly used to describe and characterize songs for music information retrieval. Pachet suggests that genre of music can be the best general information for music content description [1]. Hence, a system that can classify musical genres can solve the problem of locating a specific sound track on any device.

The only problem with musical genre classification is the fact that the definition of genre is very subjective by its nature and there exists thousands of genres or sub-genres. It is also important to note that, the definition of music genre tends to change with time, as what we call Rock song today is very different from the rock songs twenty years ago. Even though musical genres are subjective, there are certain features that can easily distinguish between different genres. By using features such as distribution of frequency or the number of beats, it is possible to classify main genres of music. For classification of musical genres, various approaches have been proposed. Unfortunately, most of these approaches have been proven to show accuracy around 60-70%. Therefore, new approaches that can maximize classification accuracy must be considered.

Hence, we try to improve the classification accuracy of music genre classification of audio signals in this work. Specifically, we use a wide range of machine learning algorithms, including k-Nearest Neighbor (k-NN) [12], k-Means Clustering [14], Support Vector Machines [9], Gaussian Mixture Models [1] and different types of Neural Networks to classify the following 5 genres: metal, classical, blues, pop, country.

Our main goal in this study is to maximize the classification accuracy of 5 genres and compare different methods of machine learning for classification of audio signals. We use state-of-the-art machine learning platforms such as PyTorch [11] to introduce deep learning into our project. We experiment with different neural network architectures and types of neural network. Moreover, we use Mel Frequency Cepstral Coefficients (MFCC) [3] to extract useful information from musical data as recommended by past work in this field. To summarize, we make three main contributions in this paper:

- We experiment with a wide range of machine learning algorithms and state their classification accuracy for 5 different genres.
- We propose a method for feature extraction and audio processing that is dependent on both MFCC and PCA. We also discuss the significance of such methods.
- We report experimental data that describe the overall effectiveness of our classification methods by including confusion matrices.

——»> ADD a paragraph that states experimental results briefly!!!! «<———

2 RELATED WORK

The development of music genre classification has been increasing rapidly in the past decade. Many approaches have been proposed that build different models for genre classification. Some approaches concentrate on the processing of audio signals, whereas some approaches try to combine audio signals with lyrics from each musical track to increase accuracy. Some of the related work to our project is presented below.

Firstly, Tzanetakis and Cook [13] introduced different feautres to organize musical tracks into a genre by using k-NN and Gaussian Mixture Model (GMM) methods. Three different feature sets for speaking to tumbrel surface, rythmic substance and pitch substance of music signse were suggested. They also introduced a dataset for music genre classification (GTZAN Dataset [13]), which is widely used today in many project, including ours.

Furthermore, Aucouturier and Pachet [1] used GMM and utilized Monte Carlo procedures for evaluation of KL divergence, which

CS 598 PS Fall 18', December 2018, Champaign

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

was used in a k-NN classifier. They conveyed some significant component sets for musical information retrieval that we use in our work, specifically the MFCC.

Apart from models such as GMM or k-NN, Feng [2] proposed an approach that uses Restricted Boltzmann machine algorithm to build deep belief neural networks. By generating more dataset from the original limited music tracks, he shows great improvement in the classification accuracy and describes the significance of neural networks for music genre classification.

Xing et. al. [16] proposes a similar approach that uses convolutional neural networks. By combining max and average pooling to provide more more statistical information to higher neaural networks and applying residual connections, Xing et. al. [16] improves the classification accuracy on the GTZAN data set greatly. Li, Chan and Chun [7] recommend a very similar technique to concentrate musical example included in audio signals by using convolutional neural networks. They present their revelation of the perfect parameter set and best work on CNN for music genre classification.

Finally, Smaragdis and Whitman [15] presents a very interesting musical style identification scheme based on simultaneous classification of auditory and textual data. They combine musical and cultural features of audio tracks for intelligent style detection. They suggest that addition of cultural attributes in feature space improves the proper classification of acoustically dissimilar music within the same style.

–» Add a paragraph that compares our work to above «– As compared to these works,...

3 PROPOSED APPROACH

3.1 Musical Dataset

For musical data, Marsyas is an open source software framework for Music Information Retrieval with the GTZAN Genre Collection Database, which has 10 genres and each genre has 100 30-second audio tracks. All the tracks are 22050 Hz Mono 16-bit audio files in .au format.

For this project, we chose five distinct genres; classical, metal, blues, pop, country. Hence, our dataset was 500 songs total, from which we used 80% for training and 20% for testing. We chose five very distinct genres as previous works [7] suggest more than 5 genres can decrease accuracy a lot and introduces many problems.

3.2 Feature Extraction: Mel Frequency Cepstral Coefficients (MFCC)

Previous works [3] on music classification and processing of audio signals directed us to use MFCC (Mel Frequency Cepstral Coefficients) as a method for feature extraction so that time domain waveforms can be represented in the frequency domain in a mel-scale. For the process of MFCC, we first computed the spectogram of each waveform by using Fast Fourier Transform and a Hamming Window. Then, we mapped each frequency to mel scale, as mel scale is the best scale for human ears. The mel spectogram of a song from each genre is shown on Figures 1 through 5, so that the difference between each genre can be visualized. After computing the mel-spectograms of each song, we applied discrete cosine transform (DCT) and then removed the very high frequency values from our data. At the end, we had an MFCC array of each song, where

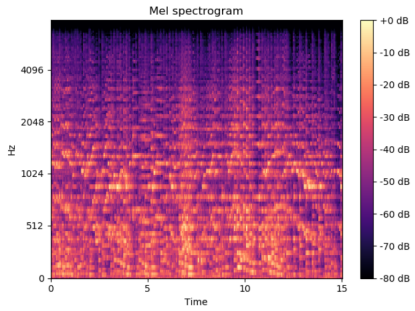

Figure 1: Classical

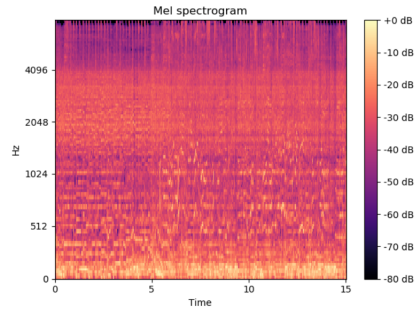

Figure 2: Metal

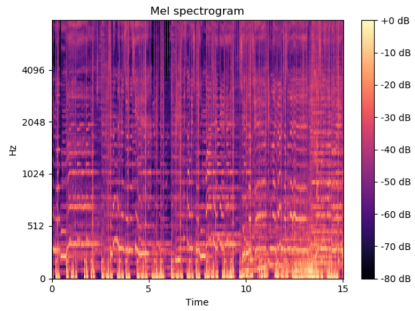

Figure 3: Pop

we stacked all them together, created appropriate labels for each genre and constructed our training and testing datasets. As stated, we used 80% for training and 20% for testing our classifiers.

3.3 Dimensionality Reduction with Principal Component Analysis (PCA)

After feature extraction and construction of final dataset, we thought of using dimensionality reduction before putting out data through classifiers. Since Principal Component Analysis (PCA) is a well-known and effective method for reduction of dimensions, we used PCA on our dataset. A realistic choice for number of reduced dimensions is to visualize the data with different PCA values and then pick the minimum dimension that can keep at least 95% of the

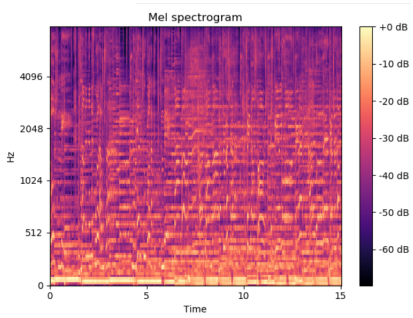

Figure 4: Country

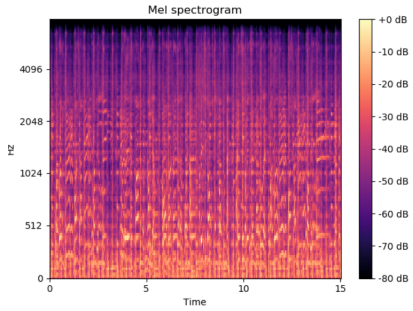

Figure 5: Blues

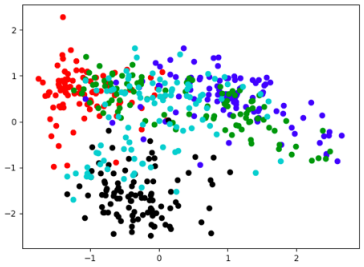

Figure 6: 2-d pca

significant components. The visualization of our data with respect to different PCA dimensions and the corresponding eigenvectors is shown on Figure 6. Note that figures 7 and 8 also present a visualization of each genre in 2 and 3 dimensions. After extensive analysis of the visualization, we reduced the dimensionality to 16. Even though 16 dimensions performed very well on classifiers such as k-NN or SVM, we had to use much bigger dimensions for our neural network, since neural networks need much more data in practice.

3.4 Machine Learning Algorithms

3.4.1 ***K-Nearest Neighbor (K-NN).*** The first algorithm we used is the very famous and effective k-closest neighbors algorithm. k-NN is a non-linear algorithm that can detect direct or indirect spread of data. It is very effective for huge amounts of data. The

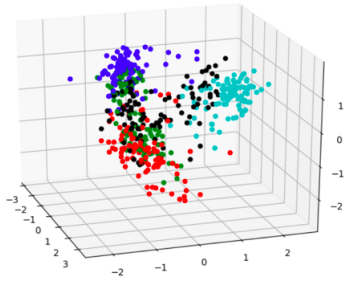

Figure 7: 3-d pca

only downside of k-NN is the fact that it makes hard decisions and might produce low classification accuracy. The most significant computation in this algorithm is measuring the distance between different points in our dataset. We used Euclidean distance for k-NN, which produced good results. After our experiments, we also found that k=5 produced the best results.

3.4.2 ***Support Vector Machines (SVM).*** The second technique we used is the support vector machine (SVM), which is a directed organization method that discovers the extreme boundary splitting two classes of information [9].

3.4.3 ***Gaussian Mixture Models (GMM).***

3.4.4 ***K-Means Clustering.***

3.4.5 ***Simple 3-layer Neural Network.***

3.4.6 ***Convolutional Neural Network (CNN).***

3.4.7 ***Super-Classifier (SC).***

4 EXPERIMENTAL RESULTS

4.1 Experiment Setup

The experimental setup for computation of classification accuracies were quite simple. After extracting features of our data through MFCC and applying PCA, we saved our training and testing datasets into a file, so that we do not have to do all the computations again. Then, we wrote a script that loads the training and testing datasets, and then puts the training data as an input to each of our classifiers with the corresponding genre labels. After training on each classifier, we computed predictions of each song and compared each label with its groun truth. Finally, we outputted the classification accuracy of each classifier and their confusion matrices, which are shown in Figure 9 through 16.

4.2 Classification Accuracy

5 CONCLUSION

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

6 EXPERIMENTAL EVALUATION

6.1 Setup

We tested our reliable code generation algorithm on different benchmarks from Media Bench [6] and MiBench [4]. The set of benchmark

Table 1: Individual instruction type rankings are presented where the first instruction type is the most significant type for Silent Data Corruption (or crash). The ranking values next to the instruction types should be considered for computing the instruction criticality. Instruction types that are not listed are treated as the least significant ranking.

Crash Rank	SDC Rank
1. *allocate*	8. *load*
2. *getelementptr*	7. *add/mul*
3. *load*	6. *call*
4. *add/mul*	5. *allocate*
5. *call*	4. *br/icmp*
6. *br/icmp*	3. *getelementptr*
7. *ret*	2. *store*
8. *store*	1. *ret*

codes used in our experiments are given in Table 2. The third column of this table explains the functionality implemented by each benchmark. The next two columns give the number of basic blocks and code size in kilobytes, respectively. The last column gives the dynamic nymber of instructions executed.

We collected statistics for a number of different applications in each benchmark and compared the SDC and Crash rates with non-modified source codes. After implementing our code generation algorithm, we again used LLFI [8] on different benchmark applications where we first injected faults in a random manner without any modification. Then, for different overhead limits and different α values, we injected faults similarly to our proposed approach. In order to get accurate results, at least a thousand fault injections were conducted on the source code using a random number generator. All experiments are repeated multiple times and the average values of those experiments were reported.

For each benchmark code in our experimental suite, we performed experiments with 3 different versions, which can be summarized as follows.
1)*BASE*: The base execution does not employ any optimization where we injected randomized faults into unmodified LLVM IR code and collected data.
2)*ICBR*: This is our instruction criticality based reliability (ICBR) enhancement approach where we applied our reliable code generation algorithm and collected both SDC and Crash results.
3)*FTP*: Finally, we conducted fault injection experiments on fully tolerant and protected (FTP) programs so that we could see how close we are to a fully fault-tolerant system with a given α and overhead.

Finally, Table 3 lists the base simulation parameters used in our experiments. Unless stated otherwise, our results are collected using these parameters. We use α as 0.5 for the default value since we want to keep the significance of SDC and Crash the same. Note that this value can easily be changed by the user. We also set the overhead limit as 70% to compare and prove the usefulness of our formula. Finally, we use 0.55 as the default value for DDC after our analysis and normalization of our IC formula.

Table 2: Benchmarks used in our experiments and their characteristics.

Benchmark	Source	Type	Number of Basic Blocks	Code Size (KB)	Instr Count (mil)
btcnt	MiBench [4]	Automative	138	98	688.3
btstrng	MiBench [4]	Automative	56	48.9	327.3
FFT	MiBench [4]	Telecomm	44	69.2	238.89
qsort	MiBench [4]	Automative	78	72.3	513.8
adpcm	MediaBench [6]	Compression	22	8	1.2
gsm	MediaBench [6]	Telecomm	98	438	7.09
jpeg	MediaBench [6]	Decompression	112	488.8	18.65
rasta	MediaBench [6]	Feature Extraction	189	269	24.86

REFERENCES

[1] Jean-Julien Aucouturier. 2003. Representing Musical Genre: A State of the Art. *Journal of New Music Research* 32 (03 2003), 83–93. https://doi.org/10.1076/jnmr.32.1.83.16801

[2] Tao Feng. 2016. Deep learning for music genre classification. *Pattern Recognition Class Paper* (2016).

[3] Z. Fu, G. Lu, K. M. Ting, and D. Zhang. 2011. A Survey of Audio-Based Music Classification and Annotation. *IEEE Transactions on Multimedia* 13, 2 (April 2011), 303–319. https://doi.org/10.1109/TMM.2010.2098858

[4] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*. 3–14. https://doi.org/10.1109/WWC.2001.990739

[5] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization (CGO '04)*. IEEE Computer Society, Washington, DC, USA, 75–. http://dl.acm.org/citation.cfm?id=977395.977673

[6] Chunho Lee, M. Potkonjak, and W. H. Mangione-Smith. 1997. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of 30th Annual International Symposium on Microarchitecture*. 330–335. https://doi.org/10.1109/MICRO.1997.645830

[7] Tom L. H. Li, Antoni B. Chan, and Andy HW. Chun. 2010. Automatic Musical Pattern Feature Extraction Using Convolutional Neural Network.

[8] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman. 2015. LLFI: An Intermediate Code-Level Fault Injection Tool for Hardware Faults. In *2015 IEEE International Conference on Software Quality, Reliability and Security*. 11–16. https://doi.org/10.1109/QRS.2015.13

[9] Michael I. Mandel, Graham E. Poliner, and Daniel P. W. Ellis. 2006. Support vector machine active learning for music retrieval. *Multimedia Systems* 12 (2006), 3–13.

[10] N. Oh, P. P. Shirvani, and E. J. McCluskey. 2002. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability* 51, 1 (Mar 2002), 63–75. https://doi.org/10.1109/24.994913

[11] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[12] L. E. Peterson. 2009. K-nearest neighbor. *Scholarpedia* 4, 2 (2009), 1883. https://doi.org/10.4249/scholarpedia.1883 revision #137311.

[13] G. Tzanetakis and P. Cook. 2002. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing* 10, 5 (July 2002), 293–302. https://doi.org/10.1109/TSA.2002.800560

[14] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. 2001. Constrained K-means Clustering with Background Knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 577–584. http://dl.acm.org/citation.cfm?id=645530.655669

[15] Brian Whitman and Paris Smaragdis. 2002. Combining Musical and Cultural Features for Intelligent Style Detection. In *ISMIR*.

[16] Weibin Zhang, Wenkang Lei, Xiangmin Xu, and Xiaofeng Xing. 2016. Improved Music Genre Classification with Convolutional Neural Networks. In *INTERSPEECH*.