# MUSIC GENRE CLASSIFICATION USING A CONVOLUTIONAL NEURAL NETWORK WITH MEL SPECTOGRAMS

**Mathew Vallejo**
McGill University
mathew.vallejo@mail.mcgill.ca

## ABSTRACT

This paper explores the implementation and evaluation of convolutional neural networks (CNNs) for music genre classification in the Python coding environment. It compares the efficiency and accuracy of two separate CNN models for a multilabel classification task using the GTZAN dataset. The two models are constructed in the Tensorflow (Keras) and Pytorch deep learning frameworks respectively, and take mel spectograms of the GTZAN audio as input data for training and evaluation. The results highlight the contributions of architectural choices in the networks as well as the limitations of deep learning on small datasets.

## 1. INTRODUCTION

The GTZAN dataset originally developed by Tzanetakis and Cook [1] will be used. The dataset is a collection of 1000 audio files of 30 seconds in length, with equal distribution across 10 distinct musical genres. The included genres consist of blues, classical, country, disco, hip hop, jazz, metal, pop, reggae, and rock. All tracks are 22,050Hz (sample rate), mono (channel count), 16-bit (bit-depth) audio files stored in the .wav file format.

Early work in audio classification relied on statistical methods of analyzing low-level audio information such as loudness, pitch, etc. [2]. Dannenberg *et al.* [3] proposed what is likely the first machine learning approach to musical style recognition, but the experiment was limited to trumpet improvisation that had been translated to MIDI for computer music understanding. These experiments, although successful, did not delve into the complex musicological considerations of genre definition. Pachet and Cazaly [4] postured that a new taxonomy was necessary for defining music genres to accompany the massive influx of audio data happening due to the booming digital audio industry at the turn of the century. They discuss how traditional genre taxonomy was fueled by marketing needs and record store organization, leaving a tremendous amount of room for subjective classification—an issue that continues to challenge modern classification models.

A move away from low-level audio features for classification can be seen in the early 2000's, such as the implementation of the multi-layer perceptron (MLP) by Costa *et al.* [5] that could account for that lack of "non-audio" information by uncovering complex relationships in the data. Around the same time, a fair amount of attention was being paid to image classification in the broader machine learning field. Such models were leveraged in the context of genre classification by using spectral imaging of audio signals as model inputs as opposed to statistical data. Costa *et al.* [6] developed the first example of this approach using the Latin Music Database (LMD) and support vector machines (SVM) for classification. Their model showed a notable increase in classification accuracy compared to state-of-the-art models from the most recent MIREX (Musical Information Retrieval Evaluation eXchange) competition at that time. Dieleman and Schrauwen [7] show how visual representations of audio (in this case mel spectrograms) can outperform raw audio inputs using CNNs for feature learning and classification. Costa et al. [8] also discuss the use of such spectrograms for classification purposes and how they can outperform models trained with acoustic features. One concern they raise about the acoustic feature approach is the possibility of model performance being affected by the feature engineering itself and not the model. Their review of the representation learning capabilities of CNNs (the ability to extract patterns from raw data) claims that they could reveal hidden features and relationships in the data and provides a comprehensive overview of the setup required for image-based genre classification with CNNs. Other research goes into more depth manually engineering the input data by separating spectrograms into percussive and harmonic elements [9] and shows that CNNs have useful characteristics both as stand-alone genre classification models as well as part of larger multimodal models [10]. In a more recent take on the genre classification task, Campobello et al. [11] utilizes the Brain Project architecture—a hybridization of genetic programming (evolving algorithms) and neural networks. Another example of a modern approach can be found in Song et al. [12] which utilizes scattering coefficients for musical image generation as opposed to more traditional methods of spectral computation.

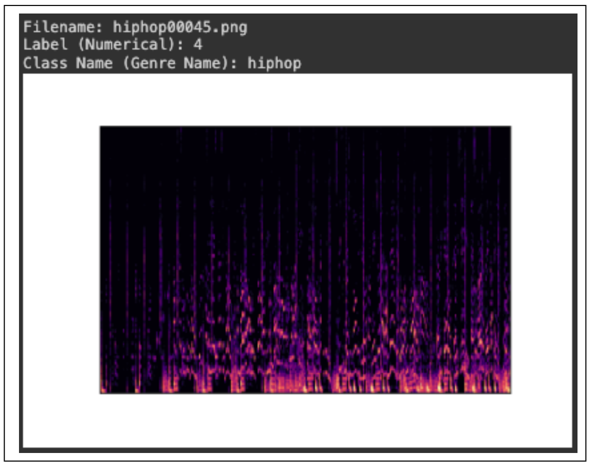


**Figure 1**. Example of mel spectogram from dataset representing a sample from the hip hop genre with its numerical label. X-axis represents time, Y-axis represents frequencies (axes are unlabeled for classification task).

## 2. MODELS

Convolutional Neural Networks (CNNs) are machine learning models that are particularly adept at image classification tasks. In their application for music genre classification, this means that audio tracks must first be converted into a visual representation of the frequency domain to be processed as input data by a CNN. This experiment will use mel spectograms (Figure 1) generated by the Librosa audio processing library [13] in the Python environment, though there are numerous transforms that have been used to accomplish this goal. Mel spectrograms are chosen as inputs due to their ability to represent important perceptual components of the audio time-frequency distribution.

A thorough review of the GTZAN dataset conducted by Sturm [14] has called attention to some potential flaws in the dataset including variable audio quality between samples that has not been annotated, artist repetition, and the subjectivity of the labeled genres for many tracks. Although accounting for these considerations is largely outside the scope of this experiment, they are worth noting as potentially contributing factors to experimental results.

Once the image-based representations of the audio data are in place, they are fed into the CNN model. One of the key capabilities of CNNs is their ability to discern relationships and features in the data relevant to the classification objective on their own. This process is known as representation learning. To examine the experimental models' learning capabilities, initial model structures will be set and hyperparameters will be adjusted as needed based on test results.

### 2.1 PyTorch Model

For the PyTorch implementation, a model was constructed from scratch using the PyTorch framework. This process includes all data loading and cleaning, as well as clearly and explicitly defining the model's object structure. Following the object-oriented programming style of Python, PyTorch allows the user to define the network's convolutional and fully connected layers, activation function, and pooling parameters along with the flow of the network's forward pass and architecture of the training and evaluation loops. K-fold validation was used during both training and evaluation to measure its effect on loss reduction and overall prediction accuracy.

Aside from the architectural and training choices, the data fed into this model was converted into tensor format prior to being fed into the network. This was primarily to investigate the usability and intuitiveness of tensor data in comparison to more common data types such as basic matrix arrays. For this model, the tensors were left as the same size as their input dimensions for ease of use.

The structure of the model includes two convolutional layers, rectified linear unit activation (ReLU), max-pooling, flattening, and a fully connected layer for final classification.

### 2.2 Tensorflow (Keras) Model

The Tensorflow implementation utilizes the Keras library, a high-level application programming interface (API) that facilitates simple user commands built on a Tensorflow back end for processing. This was chosen as a comparison to the PyTorch implementation as it represents a largely pre-optimized framework that can be taken advantage of by a wide variety of users. Its use offers insight into confounding elements that could arise in CNN performance from user-error in more flexible architectures, as well as those from the dataset itself due to increased confidence in model structure and data flow.

The structure of this model is the same as that of the PyTorch implementation with a few key differences. Although layer format is the same, K-fold validation is replaced with a dropout stage prior to flattening for better model performance. The model also receives the data as NumPy arrays as opposed to tensors.

## 3. IMPLEMENTATION AND RESULTS

### 3.1 Data Preprocessing

The raw mel spectogram images are loaded into the code as images with dimensions of 3x432x288 (channels, width, height). The data is normalized across pixels for better performance, and transformed into tensor data for the PyTorch implementation. Data is left as a NumPy array for use with Keras. Target labels are imported as a list, and converted to categorical labels in numerical format through one-hot encoding.

### 3.2 PyTorch Implementation

The PyTorch model was initialized with adaptive moment estimation (Adam) optimization and the cross-entropy loss function. Results show performance with a batch size of 10 with hyperparameters learning rate and weight decay set to 0.001 and 0.0001 respectively.

| K-Fold | Test Accuracy |
|---|---|
| 1 | 0.435644 |
| 2 | 0.465347 |
| 3 | 0.415842 |
| 4 | 0.485149 |
| 5 | 0.425743 |

**Table 1**. Test accuracy across K-folds in PyTorch implementation.

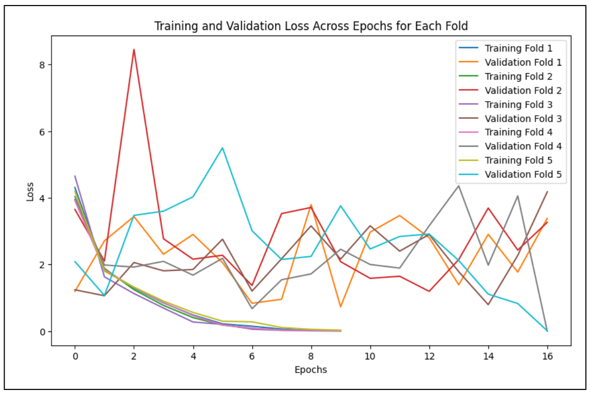


**Figure 2**. View of training and validation loss for each fold *k* across epochs for PyTorch implementation.

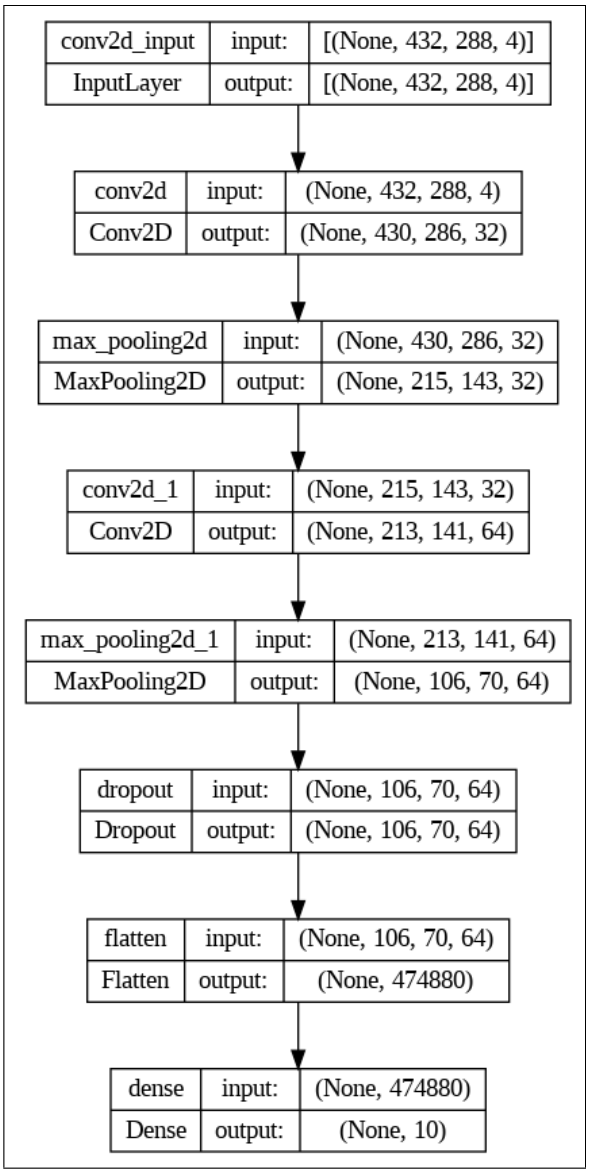
Test accuracy was lower than expected across all K-folds (Table 1), indicating potential problems with the evaluation loop. It was, however, consistent across folds.

The model showed slightly erratic behavior in the validation and test sets despite smooth loss decay during training. This is likely due to a combination of dataset size, batch size in training, and perhaps the data preprocessing (moving data into tensors). Figure 2 shows this behavior alongside the training behavior for each K-Fold (where *k*=5).

### 3.3 Tensorflow (Keras) Implementation

This implementation was built on starter code provided as course material for the COMP 551 class at McGill University. Figure 3 shows a detailed description of the model structure in compact view.

The model is trained on a batch size of 10 samples across 30 epochs. Similarly to the PyTorch implmentation, this model is initialized with Adam optimization and the cross-entropy loss function. Model accuracy for test data in this implementation hovers around 60% across all epochs, achieving this level of accuracy within the first five epochs. Despite being a well-optimized model, the low accuracy compared to what is commonly associated with modern CNN capabilities reflects the impact dataset size has on a neural network's ability to properly learn features. The model is trained on slightly over 800 samples, which is a very low number for this type of task and limits learning. Figure 4 shows model accuracy and model loss across epochs. With larger datasets it is typical to see more convergence between training and testing data.



conv2d_input | input: | [(None, 432, 288, 4)]
InputLayer | output: | [(None, 432, 288, 4)]
conv2d | input: | (None, 432, 288, 4)
Conv2D | output: | (None, 430, 286, 32)
max_pooling2d | input: | (None, 430, 286, 32)
MaxPooling2D | output: | (None, 215, 143, 32)
conv2d_1 | input: | (None, 215, 143, 32)
Conv2D | output: | (None, 213, 141, 64)
max_pooling2d_1 | input: | (None, 213, 141, 64)
MaxPooling2D | output: | (None, 106, 70, 64)
dropout | input: | (None, 106, 70, 64)
Dropout | output: | (None, 106, 70, 64)
flatten | input: | (None, 106, 70, 64)
Flatten | output: | (None, 474880)
dense | input: | (None, 474880)
Dense | output: | (None, 10)

**Figure 3**. Compact view of Keras model structure with input and output dimensions.

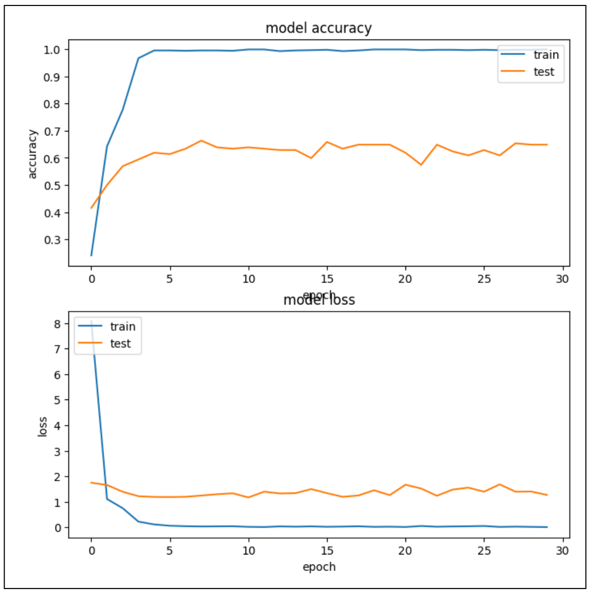
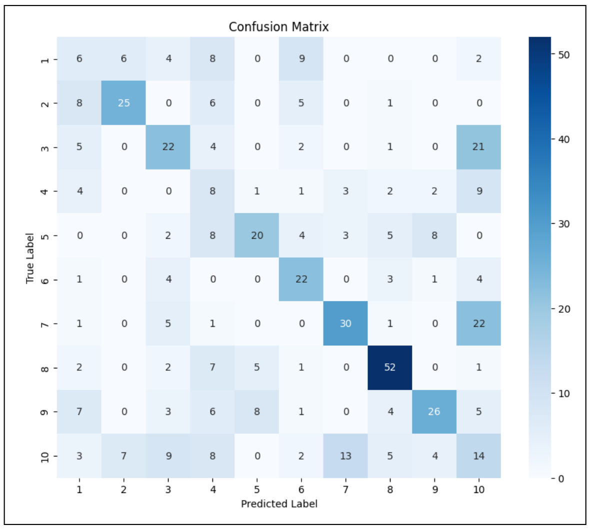


**Figure 4**. Plots of loss and accuracy of Keras implementation across epochs.

## 4. DISCUSSION AND CONCLUSION

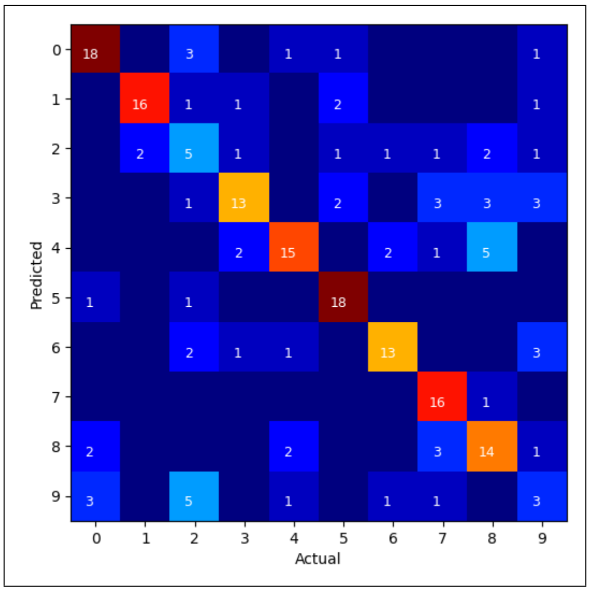
The results of these tests show us how far modern APIs like Keras have come for facilitation deep learning. The moderate success of the PyTorch implementation reflects the vast amount of variables that contribute to a successful model, which can be leveraged in the user's favor but also contribute to difficult troubleshooting when building models from the ground up in these frameworks. Further research on this topic will involve adjustment of the evaluation loop in the PyTorch implementation along with data augmentation, larger batch sizes for training, and more experimentation with hyperparameter settings.

## 5. APPENDIX



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 6 | 4 | 8 | 0 | 9 | 0 | 0 | 0 | 2 |
| 2 | 8 | 25 | 0 | 6 | 0 | 5 | 0 | 1 | 0 | 0 |
| 3 | 5 | 0 | 22 | 4 | 0 | 2 | 0 | 1 | 0 | 21 |
| 4 | 4 | 0 | 0 | 8 | 1 | 1 | 3 | 2 | 2 | 9 |
| 5 | 0 | 0 | 2 | 8 | 20 | 4 | 3 | 5 | 8 | 0 |
| 6 | 1 | 0 | 4 | 0 | 0 | 22 | 0 | 3 | 1 | 4 |
| 7 | 1 | 0 | 5 | 1 | 0 | 0 | 30 | 1 | 0 | 22 |
| 8 | 2 | 0 | 2 | 7 | 5 | 1 | 0 | 52 | 0 | 1 |
| 9 | 7 | 0 | 3 | 6 | 8 | 1 | 0 | 4 | 26 | 5 |
| 10 | 3 | 7 | 9 | 8 | 0 | 2 | 13 | 5 | 4 | 14 |

**Figure 5**. Heatmap of PyTorch implementation.



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 18 | | 3 | | 1 | 1 | | | | 1 |
| 1 | | 16 | 1 | 1 | | 2 | | | | 1 |
| 2 | | 2 | 5 | 1 | | 1 | 1 | 1 | 2 | 1 |
| 3 | | | 1 | 13 | | 2 | | 3 | 3 | 3 |
| 4 | | | | 2 | 15 | | 2 | 1 | 5 | |
| 5 | 1 | | 1 | | | 18 | | | | |
| 6 | | | 2 | 1 | 1 | | 13 | | | 3 |
| 7 | | | | | | | | 16 | 1 | |
| 8 | 2 | | | | 2 | | | 3 | 14 | 1 |
| 9 | 3 | | 5 | | 1 | | 1 | 1 | | 3 |

**Figure 6**. Heatmap of Keras implementation.

## 6. ACKNOWLEDGMENTS

The author would like to acknowledge Yue Li for their code developed for the COMP 551 course at McGill University.

## 7. REFERENCES

[1] G. Tzanetakis and P. Cook, "Automatic Musical Genre Classification Of Audio Signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, pp. 293–302, July 2002, doi: https://ieeexplore.ieee.org/document/1021072.

[2] E. Wold, T. Blum, D. Keislar, and J. Wheaten, "The title of the journal paper," *Content-based classification, search, and retrieval of audio*, vol. 3, pp. 27–36, 1996, doi: https://doi.org/10.1109/93.556537.

[3] R. B. Dannenberg, B. Thom, D. Watson, "A Machine Learning Approach to Musical Style Recognition," in *Intl. Conf.on Mathematics and Computing*, 1997.

[4] F. Pachet and D. Cazaly, "A Taxonomy of Musical Genres," in *RIAO '00: Content-Based Multimedia Information Access*, pp. 1238–1245, 2000, doi: https://dl.acm.org/doi/10.5555/.

[5] C. H. L. Costa, J. D. Valle, and A. L. Koerich, "Automatic Classification of Audio Data," in *2004 IEEE Intl. Conf. on Systems, Man and Cybernetics*, (The Hague, Netherlands), pp. 562–567, 2004, doi: https://doi.org/10.1109/ICSMC.2004.1398359.

[6] Y. M. G. Costa, L.S. Oliveira, A.L. Koerich, and F. Gouyon, "Music genre recognition using spectrograms," in *18th Intl. Conf. on Systems, Signals and Image Processing*, (Sarajevo, Bosnia and Herzegovina), pp. 1–4, 2011, doi: https://doi.org/10.1109/ICSMC.2004.1398359.

[7] S. Dieleman and B. Schrauwen, "End-to-End Learning for Music Audio," in *2014 IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, (Florence, Italy), pp. 6964–6968, 2014, doi: https://doi.org/10.1109/ICASSP.2014.6854950.

[8] Y. M. G. Costa, L. S. Oliveira, and C. N. Silla, "An Evaluation of Convolutional Neural Networks for Music Classification Using Spectrograms," *Applied Soft Computing*, vol. 52, pp. 28–38, March 2017.

[9] G. Gwardys and D. Grzywczak, "Deep Image Features in Music Information Retrieval," *Intl. Journal of Electronics and Telecommunications*, vol. 60, pp. 321–326, 2014, doi: https://doi.org/10.2478/eletel-2014-0042.

[10] S. Oramas, F. Barbieri, O. Nieto, and X. Serra, "Multimodal Deep Learning for Music Genre Classification," *Transactions of the International Society for Music Information Retrieval*, vol. 1, pp. 4–21, 2018, doi: https://doi.org/10.5334/tismir.10.

[11] G. Campobello, D. Dell'Aquila, M. Russo, and A. Segreto, "Neuro-Genetic Programming for Multigenre Classification of Music Content," *Applied Soft Computing*, vol. 94, September 2020, doi: https://doi.org/10.1016/j.asoc.2020.106488.

[12] G. Song, Z. Wang, F. Han, S. Ding, and X. Gu, "Music Auto- Tagging Using Scattering Transform and Convolutional Neural Network with Self-Attention," *Applied Soft Computing*, vol. 96, November 2020, doi: https://doi.org/10.1016/j.asoc.2020.106702.

[13] B. McFee *et al.*, "Librosa: Audio and Music Signal Analysis in Python," in *Proc. of the 14th Python in Science Conference*, (Austin, Texas), pp. 18–24, 2015, doi: https://doi.org/10.25080/Majora-7b98e3ed-003.

[14] B. L. Sturm, "The GTZAN Dataset: Its Contents, Its Faults, Their Effects on Evaluation, and Its Future Use," *Journal of New Music Research*, vol. 43, pp. 147–172, 2014, doi: https://doi.org/10.1080/09298215.2014.894533.