Music Genre Detection From Audio

Irtiza Khan ikhan12@horizon.csueastbay.edu
Computer Science 497
California State University, East Bay
Hayward, CA 94542, USA

Editor: Irtiza Khan

Abstract

Over the years, deep learning has found its footing in society to accomplish a variety of useful tasks. What if music fans and listeners could have their playlists of songs curated to them by genre. In this paper, we will be attempting to use deep learning to find a way to detect the music genre of songs from their audio data. We'll be decomposing audio data into its separated forms, analyzing them, and then combining them back into a rich visual format. From there, we will be applying a deep learning model on it. In the end, we will have a music genre detection program.

Keywords: Deep Learning, Convolutional Neural Network, Music Genre Classification, Spectrogram

1. Introduction

1.1 Problem Definition

The problem this paper attempts to solve revolves around music genre classification. More specifically, we are trying to classify songs (from audio data) into its respective musical genre. To maintain a level of simplicity, there are 11 different genres this project attempts to classify music into: rock, reggae, pop, metal, jazz, hip-hop, electronic, disco, country, classical, and blues.

1.2 Background

There are a vast number of ways to tackle this one problem. Common methods include machine learning techniques such as K-Nearest Neighbors classification and K-means clustering. Information can be extracted from wave-forms for feature engineering, and statistically combined to feed into a Machine Learning algorithm. However, we will be focusing on achieving this through deep learning instead. Within deep learning itself, there are various different neural networks that can be applied here including, but not limited to convolutional neural networks, Recurrent Neural Networks, Long Short-Term Memory, or even a combination of these.

1.3 Motivation

The motivation here is to take a sea of songs and organize them into musical categories to improve things such as music discovery. This allows for tools that can split incredibly large playlists into smaller, more digestible, and searchable playlists. So, listeners with playlists consisting of a large array of musical genres, can have an algorithm curate their songs into smaller, more sizable playlists according to their respective genres.

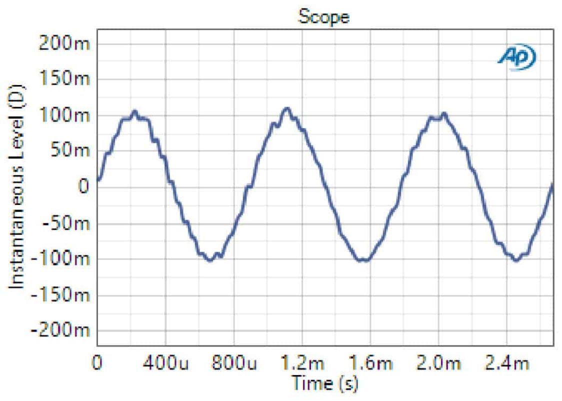
2. Approaches

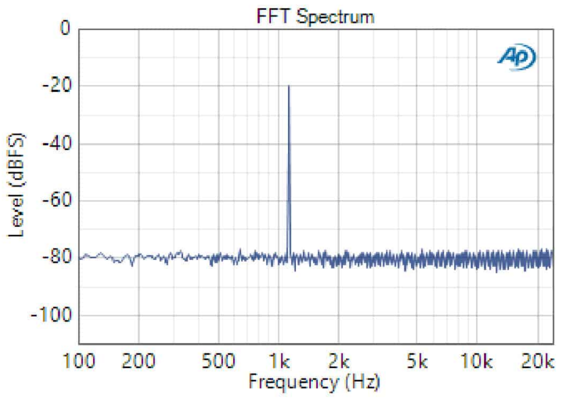
Our goal is to apply a deep learning algorithm on our data so that we can build a solution for our problem. In this approach, a convolutional neural network was used. While convolutional neural networks have predominantly been the solution to image classification problems, it can be extended to audio data as well.

2.1 Fourier Transformation

To apply a convolutional neural network on audio data, we have to be able to first visualize and represent all the rich information stored inside of the audio data. The first step is to decompose our sound wavelength into its spectral components -much like splitting a food dish into its ingredients. This is achieved with the Fourier transformation, given by the following equation:

$$s = A_1 sin(2\pi \mathrm{f}_1 t + \varphi_1) + \mathrm{A}_2 sin(2\pi \mathrm{f}_2 t + \varphi_2)$$

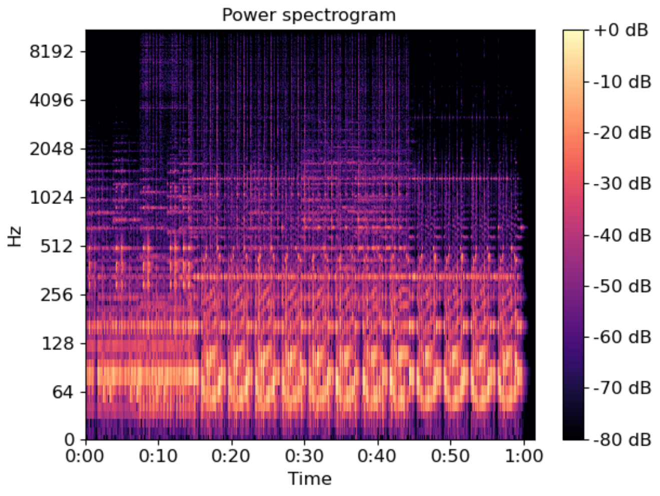




2.2 Short Term Fourier Transformation

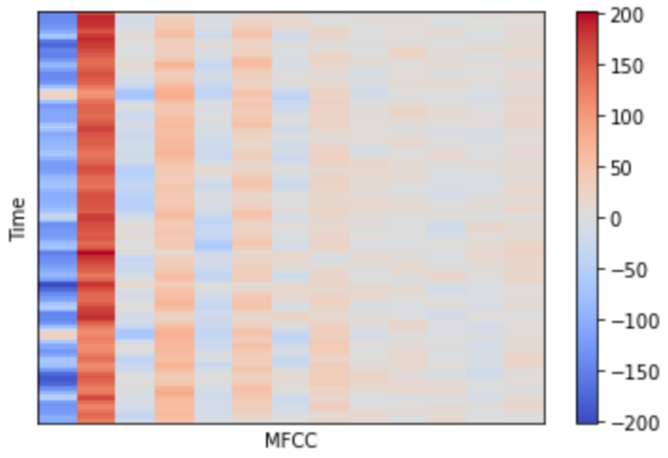
When the Fourier transformation is applied to our audio wavelength, our wave graph (of amplitude and frequency) is transformed into a power spectrum graph (of magnitude and frequency). Thus, the output of a Fourier transformation does not preserve temporal data, an important feature of audio. So we have to take the Fourier Transformation at several time intervals, and combine them together into what's known as a Short Time Fourier Transformation. Ultimately, we are left with a 3 dimensional spectrogram containing magnitude, frequency, and time.

*Something to note is, we are using a Fast Fourier Transformation (FFT) algorithm to compute the Fourier transformation faster.



2.3 Mel Frequency Cepstral Coefficients

We still have one more thing to capture from our audio data: the timbre/texture of a sound. Both a piano and violin can play the same chord (frequency) at the same volume (amplitude) for a set interval (time). So what differentiates these two types of sounds? It's the timbre. Mel Frequency Cepstral Coefficients (MFCCs) can capture this data by creating a vector of coefficients at each frame.



2.4 Convolutional Neural Network

We are finally ready to apply our convolutional neural network to our audio image. First let's recap CNNs. CNNs attempt to emulate the human visual system by extracting and recognizing different features that make up an image (distinguishing different edges and shapes). Kernels or filters are a grid of weights that make up a convolutional layer. These filters are essentially feature detectors extracted by the algorithm and applied iteratively across the entire image.

Architecture of the convolutional network depends on the grid size (odd number of height and width in pixels), stride (step size of sliding the kernel on the image in pixels), depth (number of channels, i.e. RGB), and number of kernels (a convolutional layer has multiple kernels). Each kernel produces a single 2D array. The second part of the CNN model includes pooling, which is simply the downsampling of the image.

Now that we grasp how convolutional neural networks work, it's time to learn how to apply it to audio data. As previously mentioned, our MFCC spectrogram is the image the CNN model will be applied over. Thus, the grid size of height and width in pixels is represented by time and frequency in amplitude.

2.5 Preparing MFCCs for CNN

We are taking 51,200 numbers of Fourier Transformations at 512 intervals. Thus, the number of Short Term Fourier Transformation is 100 (number of fourier transformations / number of intervals = 51200/500=100). At each time window, we will be using 13 MFCCs. Since audio data is essentially in grayscale, it only has one channel. Therefore, they only have a depth of 1.

In summary, MFCCs: 13, Hop Length: 512 samples, Number of samples in audio file: 51,200 samples, Data Shape: 100 x 13 x 1

3. Experimental Setup

3.1 Data Used

The dataset used in this project comes from two different sources. One dataset was collected from Kaggle, and the other was collected personally.

3.1.1 Kaggle Data Set

The kaggle dataset consisted of 1,000 30-second sound clips categorized into 10 genres (100 songs per genre). They were saved as '.WAV' files. This was not a perfect dataset for a couple of reasons. To begin with the data was collected from 2000 to 2001, so the selection of songs fell into a very small time frame and was not representative of the different music styles across generations. The other shortcoming was simply not being a large enough dataset to create an accurate model. This will be covered later in the paper.

3.1.2 Personally Collected Data Set

The personally collected data set consisted of 10 hours of roughly 1,200 30-second clips categorized into 11 genres. This data required a lot of collecting, cleaning, and processing.

3.1.3 Data collection

The first was collecting the data. A list of popular one to two hour compilations of popular songs for each genre over the past decade had been aggregated from Youtube. Each video was converted into a '.WAV' file to match the Kaggle data already collected.

3.1.4 Data Processing

The next step was processing the data. There were two portions to the processing: compression and splitting. Since '.WAV' files are uncompressed by nature, the file sizes were unnecessarily big. So, every file was compressed through a compression software so as to be one third of its original size. Now that our file sizes are manageable, we can begin splitting our 1 hour long song compilations into 30 second chunks through python scripts.

3.1.5 Data Cleaning

After processing our data, we began cleaning. Some of the files during compression and conversion have been improperly transformed. Some files became corrupted and some were not cut into equal 30-second segments. They were manually removed.

3.1.6 Data Preparing

Finally, we have our two homogeneous datasets. We can now combine and organize all the songs pertaining to the same genre in their respective files. Now, in training the model with this data we virtually split the 30 second

3.2 Hyper-parameters

In order to have a deep learning model that performed well, we needed to make sure it was tuned properly and using appropriate data. By far the most impactful thing to improve the accuracy of this model was the addition of the new dataset. However, there were additional changes to the hyperparameters that contributed to many other gains.

3.2.1 Train, Validation, and Test sets

The test set was made up of 10% of the training set, and the validation 10% of the training set was designated for the test set. Another 10% of the training set was used for the validation set.

3.2.2 Convolutional Layers

The Convolutional Neural Network uses 3 convolutional layers. The output of these convolutional layers is flattened and then fed into a dense layer. This dense layer is followed by the output layer.

The test set was made up of 10% of the training set, and the validation 10% of the training set was designated for the test set. Another 10% of the training set was used for the validation set.

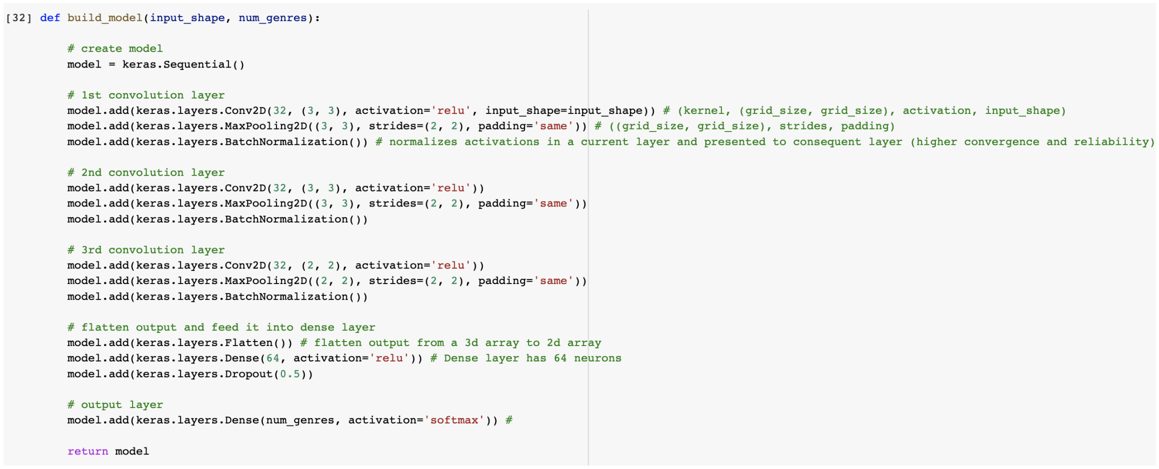
3.2.3 Dense and Output Layers

Once flattening the output of the convolutional layers, we have a dense layer made up of 64 neurons. To adjust for overfitting, the most suitable dropout seemed to be 0.5.

The output layer used an activation function of 'softmax' which provides a probability distribution on top of different categories (genres).

3.2.4 Optimizer

Adam optimizer with a learning rate of 0.0001 was used.



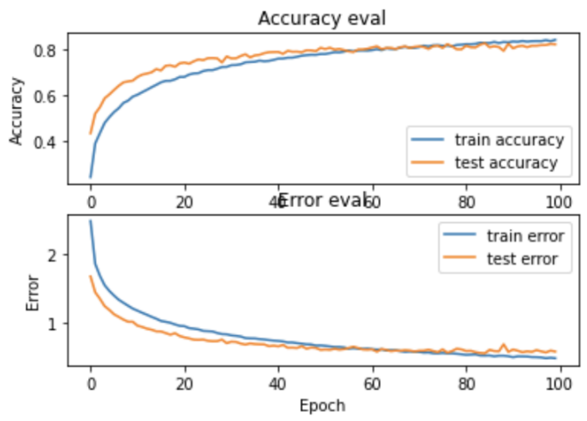
```
[32] def build_model(input_shape, num_genres):

         # create model
         model = keras.Sequential()

         # 1st convolution layer
         model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)) # (kernel, (grid_size, grid_size), activation, input_shape)
         model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')) # ((grid_size, grid_size), strides, padding)
         model.add(keras.layers.BatchNormalization()) # normalizes activations in a current layer and presented to consequent layer (higher convergence and reliability)

         # 2nd convolution layer
         model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
         model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
         model.add(keras.layers.BatchNormalization())

         # 3rd convolution layer
         model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
         model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
         model.add(keras.layers.BatchNormalization())

         # flatten output and feed it into dense layer
         model.add(keras.layers.Flatten()) # flatten output from a 3d array to 2d array
         model.add(keras.layers.Dense(64, activation='relu')) # Dense layer has 64 neurons
         model.add(keras.layers.Dropout(0.5))

         # output layer
         model.add(keras.layers.Dense(num_genres, activation='softmax')) #

         return model
```

3.2.5 Training

After some experimenting, a 100 epochs with a batch size of 64 was used to train the model. As far as I know, this gave for the most accurate result while maintaining a low overfitting problem.

3.2.6 Results

The validation accuracy achieved was 81.99% where the training set outperforms the validation set by 1.93% (training accuracy was 83.92%). The accuracy on the test set was 80.22%.



```
83/83 - 0s - loss: 0.5906 - accuracy: 0.8022 - 299ms/epoch - 4ms/step

Test accuracy: 0.8022019863128662
Target: 6 Predicted label: [6]
```

4. Conclusion

We have successfully built a deep learning model that can accurately predict the genre of a song by listening to it. It does this by breaking down the audio into the various components that make it up and represents them in a visual format. Once in a visual form, we are able to apply a convolutional neural network to it. Through some experimentation and tweaking, we are able to reach a relatively good model able to predict musical genres.

References

Peter Gilliam. ”musical fourier”. URL `https://www.youtube.com/watch?v=Eayo7pZ2g7A`.

Velardo Velardo. ”ai audio / music consultant”, a. URL `https://valeriovelardo.com`.

Velardo Velardo. ”deep learning (for audio)...”, b. URL `https://www.youtube.com/watch?v=fMqL5vckiU0&list=PL-wATfeyAMNrtbkCNsLcpoAyBBRJZVlnf`.

Velardo (a) Velardo (b) Gilliam