Music Genre Classification with CNNs

Manuel Rodríguez Rodríguez, Farshid Alizadeh-Shabdiz

Boston University, Boston MA 02215, USA
mrdgz2@bu.edu, alizadeh@bu.edu

Abstract. The goal of this project is to be able to detect the genre of a song by training a Convolutional Neural Network (CNN).

To do this, we will use three datasets. First, we will use the **GTZAN dataset** which consists of 900 **30-second, 22kHz, Mono, 16-bit .WAV files**, belonging to 9 different genres from 2000-2001 to build and test our model architecture.

After that, we will expand the GTZAN dataset with two new genres sourced completely from Spotify Playlists.

Finally, we will create our own expanded, modern, bigger, dataset using **Spotify Playlists** to be able to classify and detect more genres.

Keywords: mel spectrogram, music genre recognition, cnn music genre, spotify, gtzan. python, keras

1 Project Description

The goal of this project is to be able to **detect the genre of a song**, using Mel spectrograms, by training a **Convolutional Neural Network (CNN) with Keras using Python.** For this purpose, we will work on **three different datasets**.

First of all, we will begin our work on the **GTZAN dataset** which consists of 900 **30-second, 22kHz, Mono, 16-bit .WAV files**, belonging to 9 different genres from 2000-2001 to **build and test our model architecture**:

- Blues
- Pop
- Rock
- Classical
- Country
- Disco
- Hip-hop
- Metal
- Reggae

From then on, we will **expand the GTZAN dataset with two new genres** created from scratch using Spotify Playlists and see how the model adapts to using these two new genres.

In particular, the two genres that we will include will be 'Rap' and 'Brazilian Funk'. 'Rap' because it is one of the **biggest music genres** out there and it would be interesting to build a genre from scratch using Spotify and see how it performs versus the original GTZAN dataset.

Brazilian Funk for two purposes: first and foremost, it is a wink to **my sweet Brazilian girlfriend**. Secondly, it is a genre that is usually used to **remix popular songs** and it will be an interesting observation to compare our model with the original song versus the remixed, Brazilian Funk, version of the song. The inspiration behind this was **Rihanna's Brazilian Funk interpretation** of her hit 'Rude Boy' at this year's **Super Bowl**.

The inspiration behind this paper came from Kunal Vaidya's post [1] on **TowardsDataScience**. From the original paper came the idea of transforming the GTZAN dataset into Mel Spectrograms to be able to treat the audio files as images to use with CNNs and to split each audio file into 10 3-sec clips.

However, I felt like little documentation was provided and that the project had a lot more to offer and investigate. For this reason, in this paper we aim to:

- Create a completely new dataset sourced from Spotify Playlists with more genres and bigger size.
- Optimize, test and compare different model architectures.
- Focus on analyzing how the model performs in 'original' versus 'remixes' songs.
- Provide easy-to-access, reproducible and readable source code and documentation on the whole process.

Most importantly however, we hope to use this project as a learning opportunity to improve our knowledge on Machine Learning, Artificial Intelligence and Neural Networks.

1.1 What is a Mel Spectrogram?

To train our CNN, we need to transform the audio into an image that the network will understand. To do this, we will need to generate a Mel spectrogram, which manages to capture a lot of details about the audio.

To explain what a Mel spectrogram is, we first have to understand the different steps involved in the process of generating it [2].

First, **we take 22.050 samples (22kHz) per second of the variations in air pressure of the song**. This generates the following waveform, which everyone is used to, where the x-axis is the time, and the y-axis is the amplitude. The higher the amplitude, the louder the song is at that point in time.

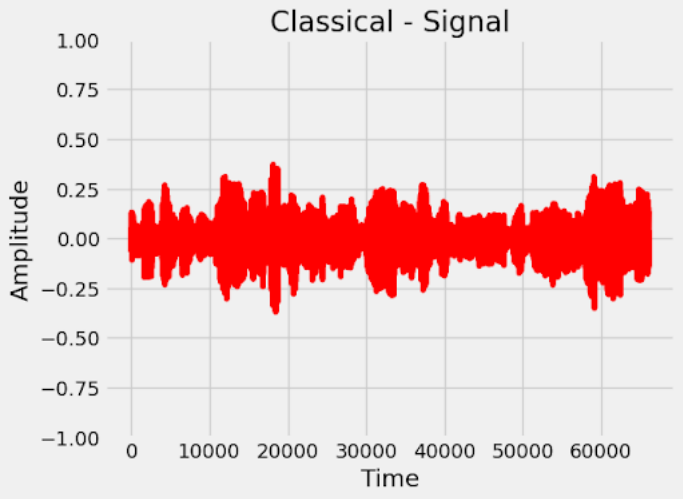

Fig. 1. Signal waveform of a 'Classical' genre sample from the dataset.

This waveform belongs to a song in the 'Classical' genre. Here is the waveform for a song in the 'Rock' genre, for comparison:

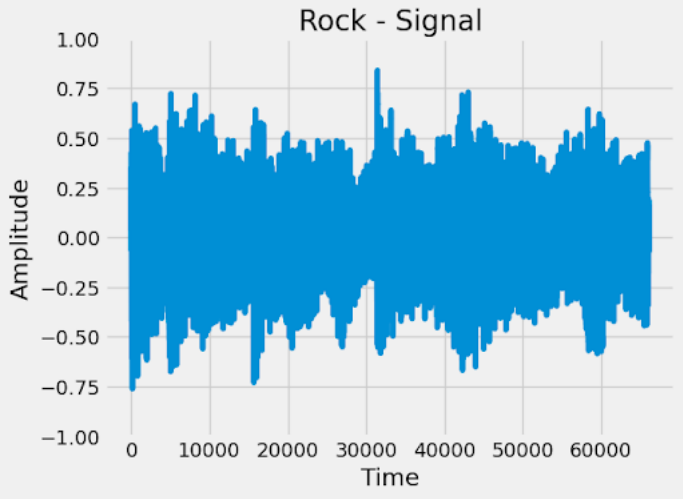

Fig. 2. Signal waveform of a 'Rock' genre sample from the dataset.

The problem with these waveforms is that they only capture the loudness of the audio, as an aggregate of all combined frequencies.

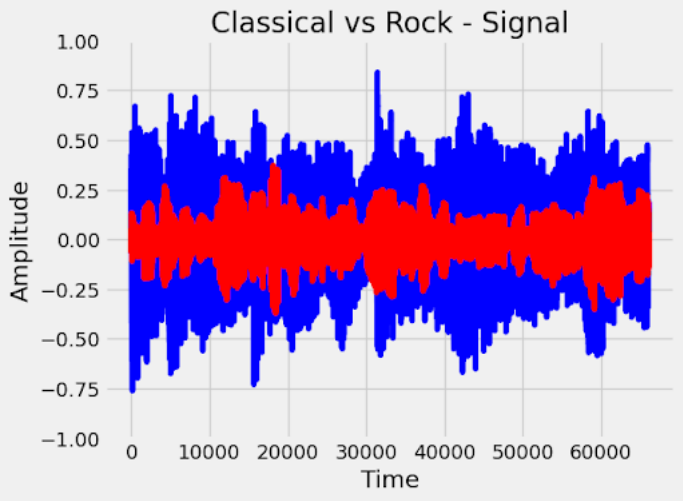

Fig. 3. Comparison of 'Classical' versus 'Rock' signal waveforms.

So yes, Rock songs tend to be louder than classical songs, but that's not nearly enough information to classify genres. To solve this and gain more information on what frequencies the song is playing at, we apply a Fourier Transformation.

The explanation of this mathematical formula escapes the purpose of this project, but the Fourier Transformation basically gets the signal for each time input and returns the decomposition of frequencies of the signal at each time point.

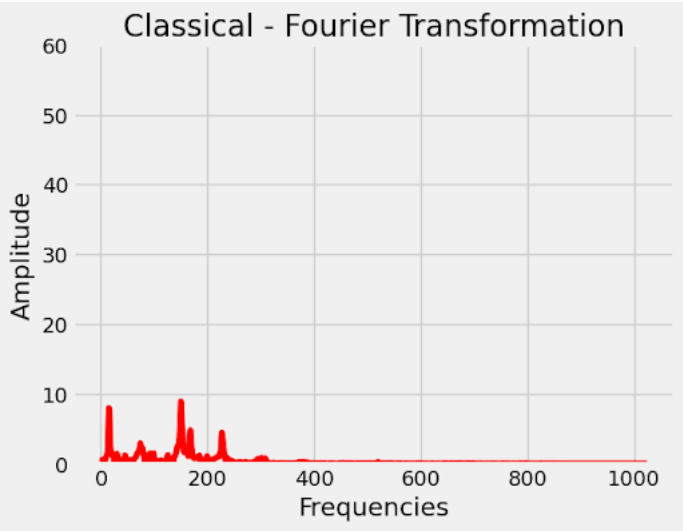

Fig. 4. The Fourier Transformation of a ‘Classical’ sample.

Here we can start to see which frequencies are being used the most and at which amplitude. And visually, compared to rock, we can begin to see differences:

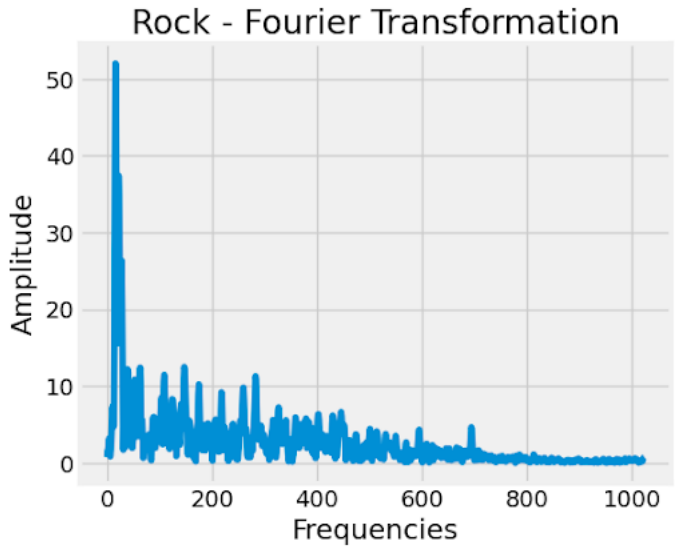

Fig. 5. The Fourier Transformation of a ‘Rock’ sample.

And the overlay:

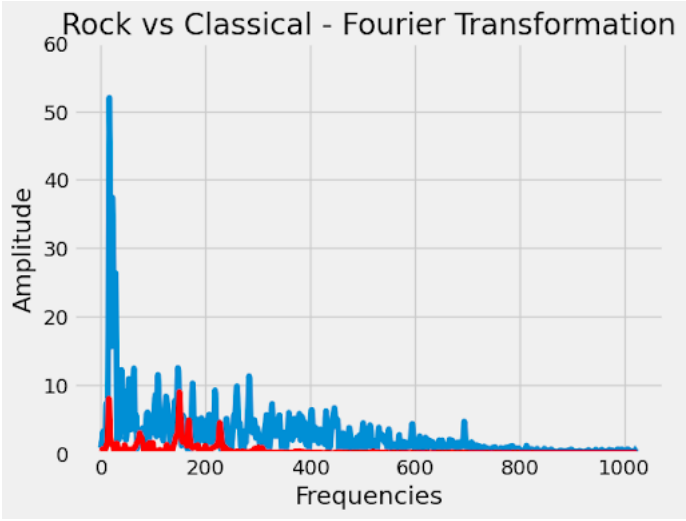

Fig. 6. Comparison of the Fourier Transformation of a ‘Classical’ versus ‘Rock’ sample.

From this, we generate the spectrogram by applying the Fourier Transformation at every point in time, obtaining the decomposition of frequencies. We also convert things to log-scale so that it's easier to visualize, as numbers can get very big, and we apply a color gradient.

This is the result, 'Classical' spectrogram:

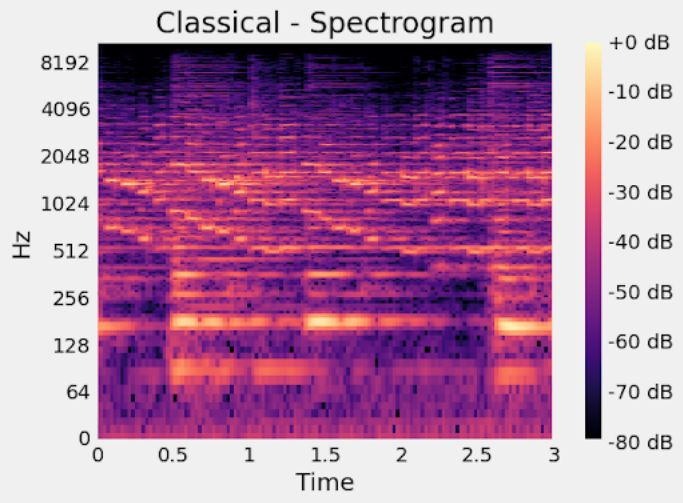

Fig. 7. The spectrogram of a 'Classical' sample.

Versus a 'Rock' spectrogram:

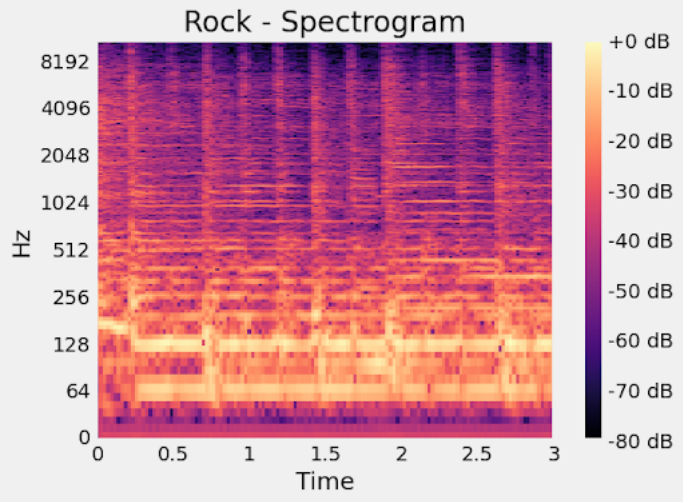

Fig. 8. The spectrogram of a 'Rock' sample.

But this generates a spectrogram, where for each unit of time we have the amplitude in Decibels of each frequency. We're still missing the *Mel* in Mel Spectrogram.

The human ear has trouble differentiating sounds in higher frequencies. For example, we are very good at telling the difference between a sound at 767Hz versus 1200Hz. However, we are bad at doing the same thing when it's the same distance in higher frequencies, like 10.000Hz and 10.433Hz.

For this reason, the Mel Scale was introduced in 1937 to scale sounds so that each point in this scale would sound equally distant to the listener.

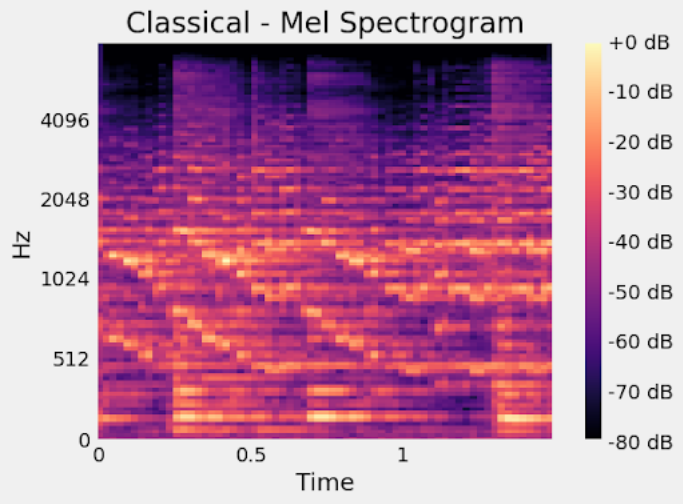

Fig. 9. The Mel Spectrogram of a 'Classical' sample.

And the 'Rock' Mel spectrogram:

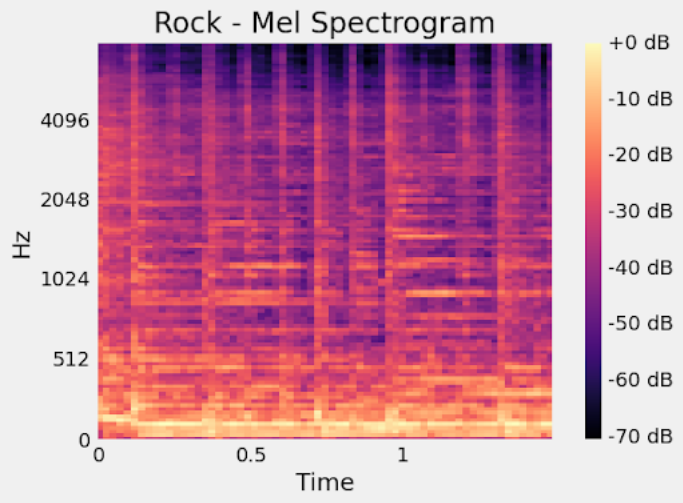

Fig. 10. The Mel Spectrogram of a 'Rock' sample.

As we can see, there are a lot of differences between a Rock Mel spectrogram and a Classical Mel spectrogram, and **these are the patterns on which we will train the CNN model to pick up on for each genre**.

2 Datasets

As we mentioned in the project description, one of the important innovations of this paper was the creation and testing of various datasets. In the following chapters we will go into detail on how we generated each dataset.

2.1 GTZAN Dataset

As mentioned previously, we used the GTZAN dataset [3], which consists of 900 **30-second, 22kHz, Mono, 16-bit .WAV files**, belonging to 9 different genres from **2000-2001**:

- Blues
- Pop
- Rock
- Classical
- Country
- Disco
- Hip-Hop
- Metal
- Reggae

This means that each genre only had 100 samples to train on (minus validation and testing). For this reason, we decided to split each 30-second clip into 10 different 3-second clips, resulting in 1000 samples per class. We use the following code to split the WAV files:

```
import os
from pydub import AudioSegment

def process_wav(song, tag):
    i = 0
    to_return = []

    total_duration = len(AudioSegment.from_wav(song))

    intervals =  int(total_duration / 3000)

    for w in range(0, intervals):
        i = i + 1
        t1 = 3 * (w) * 1000
        t2 = 3 * (w + 1) * 1000
        newAudio = AudioSegment.from_wav(song)
        new = newAudio[t1:t2]
        new.export(str(tag) + '_' + str(w) + '.wav', format = 'wav')
        to_return.append(str(w) + '.wav')

    return to_return, intervals

audio_files = os.listdir()

i = 0
for file in audio_files:
    #Split file into name and extension
    name, ext = os.path.splitext(file)
    if ext == '.wav':
       process_wav(file, i)
       i += 1
```

For each of the 9000 samples, we generated the corresponding Mel spectrogram using the method suggested here [2], splitting the dataset 80-10-10. To generate the Mel spectrogram, we used the following code:

```
import os
import librosa
import matplotlib.pyplot as plt

def generate_spectogram_for_song(song):
```

```
    x, sr = librosa.load(song, sr = None)
    X = librosa.stft(x)
    Xdb = librosa.amplitude_to_db(abs(X))
    librosa.display.specshow(Xdb, sr = sr, x_axis = 'time', y_axis = 'log')
    plt.axis('off')
    plt.savefig(song.replace('.wav', '.jpg'), bbox_inches = 'tight')
    plt.cla()

    return song.replace('.wav', '.jpg')

audio_files = os.listdir()

for file in audio_files:
    #Split the file into name and extension
    name, ext = os.path.splitext(file)
    if ext == '.wav':
       generate_spectogram_for_song(file)
```

So, this meant 7200 samples used for training, 900 used for validation and 900 used for testing. All of this is included under the 'original_dataset' folder.

2.2 GTZAN + Rap & Brazilian Funk

On top of that, we wanted to expand and introduce our own genres to see if the model would be able to work with them.

For this reason, we downloaded **two Spotify Playlists** for the following genres: 'Rap' and 'Brazilian Funk' in MP3 format, converted them to the correct **WAV, 22kHz, Mono, 16-bit format**, generated the corresponding Mel Spectrogram and added them to the dataset.

To download the Spotify Playlists, we used a Python library named 'SpotDL' [4]. The usage is simple:

```
spotdl 'https://open.spotify.com/playlist/37i9dQZF1DX38t16fuNXJJ?si=8f37db54aa854c1d'
```

We can use this command straight from terminal and it will download the songs in the playlist in MP3 format to the current folder.

For the sourcing of the Rap genre, we used the playlist 'Best Rap Songs of the 2000' [5]. For the Brazilian Funk playlist [6], credit due to Brazilian native Sofia Léo Moreira.

To convert MP3 to the correct WAV format we did this:

```
import os
```

```
from pydub import AudioSegment

genres = ['rap', 'brazilian_funk']

for genre in genres:
    path = 'dataset/' + genre + "/"
    os.chdir(path)
    audio_files = os.listdir()
    for file in audio_files:
        name, ext = os.path.splitext(file)
        if ext == '.mp3':
            sound = AudioSegment.from_mp3(file)
            sound = sound.set_channels(1)
            sound = sound.set_frame_rate(22050)
            sound.export('{0}.wav'.format(name), format = 'wav')
    os.chdir('..')
    os.chdir('..')
```

The expanded dataset can be found under the ‘custom_dataset’ folder.

2.3 Spotify Dataset (3-Seconds)

The next step was to build our own complete dataset from scratch using Spotify playlists. We chose 14 genres, with more samples per set each when compared to the GTZAN dataset.

The **only genre we left out was Brazilian Funk**, because there weren’t as many samples available as with the others and leaving it in would unbalance the training/testing/validation structure. Here is the list of the genres:

- Classic [7]
- Country [8]
- EDM [9]
- Flamenco [10]
- Hip-Hop [11]
- Indie [12]
- Jazz [13]
- K-Pop [14]
- Latin [15]
- Metal [16]
- Pop [17]
- Rap [18]
- Reggae [19]

- Rock [20]

New inclusions are:

- **'Latin'**, which has become very popular worldwide since the 2000s.
- **'K-Pop'**, a very famous genre in South Korea that has managed to cross borders and make it into the Western mainstream.
- **'Jazz'**, a huge genre that did not see representation in the GTZAN dataset.
- **'EDM'**, was kind of a substitute for 'Dance' in the GTZAN dataset. Is more modern and comprises more of the electronic music scene.
- **'Flamenco'**, as a Spanish student, it was basically my obligation to include a very typical Spanish dance genre.
- **'Indie'**

Naturally, the genres did not have the same number of samples, so we limited it to the one with the least number of songs. This ended up resulting in **3200 Mel spectrograms per genre**, split accordingly at 80% training, 10% testing, 10% validation. This was an **increase of 3x compared to the GTZAN dataset**.

During this part of the project, we realized, that we could also **test 9-second clips of the songs versus 3-second clips**, in an attempt to give the model more information per input (each Mel spectrogram). On this note, we re-created the previous dataset but with 9-second clips, which we will explain in the next chapter.

2.4 Spotify Dataset (9-Seconds)

Our thought process behind creating this alternate version of the Spotify dataset was that 3 seconds isn't that much to clearly differentiate one genre from another. A silence, a drop, a bridge… any of those elements are pretty much unanimous to any genre and could be captured in a 3-second clip.

To avoid generating this ambiguity in the training of the model, we decided to merge each pair of three consecutive Mel spectrograms into just one. The ending number of samples was 1000 per class. This is the same number as the samples per class in the GTZAN dataset, but with vastly more information per sample.

This is a quick comparison of a 3-second Mel Spectrogram versus a 9-second Mel Spectrogram:

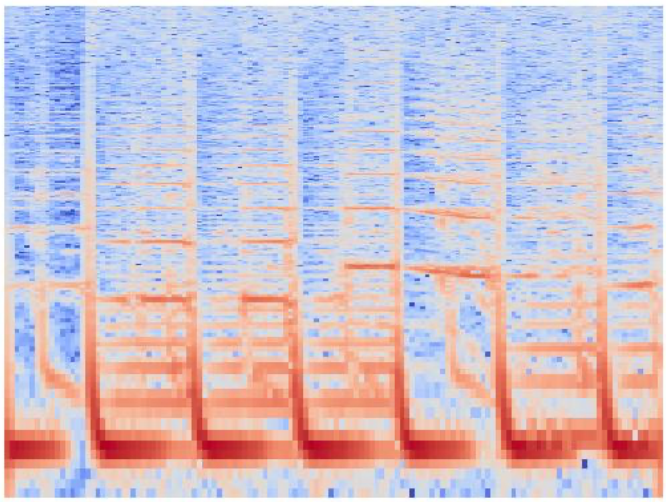

Fig. 11. 3-second Mel spectrogram of 'EDM' file 0_33.jpg.

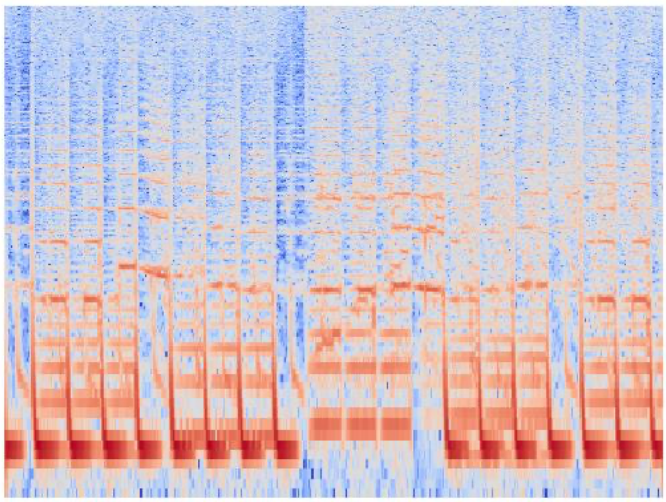

Fig. 12. 9-second Mel spectrogram of 'EDM' files 0_33.jpg, 0_34.jpg, 0_35.jpg.

We will analyze in the results chapter how the final model fared in this alternate dataset.

3 Model Building

In regards to building the Convolutional Neural Network, we tried many different approaches, all based on sequences of Conv2D + MaxPooling layers.

First, we are going to use the GTZAN dataset to perform quick tests and choose our baseline model. Once we have that, we will try out many different architectures on the Spotify 3-Second dataset and compare them after 10 epochs. The idea is to find the right direction to work towards.

On the ones that perform well, we will increase the training to 20 epochs and keep filtering out models until there is a clear winner for the Spotify 3-Second dataset, which is the end goal of this project. Here are some of the architectures we're going to try based on the baseline model (we are re-using the model architecture, but we're re-training on the new Spotify dataset!):

- **Baseline GTZAN model**. After multiple tests on the base GTZAN model, we found this one to get the best accuracy, which consists of: 5 layers of Conv2D + MaxPooling with 3x3 kernel and increasing filters (8, 16, 32, 64, 128), with Dropout of 0.3, on ReLu activation and Adam (lr = 0.001). Reduce on Plateau learning rate scheduler with patience = 1 and factor of 0.8. For following architectures, only the changes to this baseline model will be mentioned.
- **Less filters**. 5 layers of Conv2D + MaxPooling of increasing filters (4, 8, 16, 32, 64) with Dropout.
- **Less layers**.4 layers of Conv2D + MaxPooling of increasing filters (8, 16, 32, 64) with Dropout.
- **Inverted pyramid**. 5 layers of Conv2D + MaxPooling of decreasing filters (128, 64, 32, 16, 8) with Dropout.
- **5x5 Kernel**. 5 layers of Conv2D + MaxPooling with kernel 5x5 of increasing filters (4, 8, 16, 32, 64) with Dropout.
- **L2 Regularization**. 5 layers of Conv2D + MaxPooling with L2 Regularization of increasing filters (4, 8, 16, 32, 64) with Dropout.
- **Batch Normalization**. 5 layers of Conv2D + MaxPooling with Batch Normalization of increasing filters (4, 8, 16, 32, 64) with Dropout.
- **Batch Normalization + 5x5 Kernel**. 5 layers of Conv2D + MaxPooling with 5x5 Kernel and Batch Normalization of increasing filters (4, 8, 16, 32, 64) with Dropout.

After 10 epochs, these were the results:

Table 1. Model Building results after 10 epochs.

Model	val_loss	val_accuracy
Baseline Model	1.1015	0.6891
Less filters	1.0308	0.6781
Less layers	1.6548	0.6574
Inverted pyramid	1.2200	0.5955
5x5 Kernel	1.0895	0.6866
L2 Regularization	1.2620	0.6196
Batch Normalization	0.9483	0.7272

BN + 5x5 Kernel	0.8536	0.7496

As we can see, the baseline model we got from the GTZAN dataset performs well after re-training for this dataset. Less filters, Batch Normalization, 5x5 Kernel and BN + 5x5 Kernel also yielded very good results, so we will go ahead with these 5 and train them for 20 epochs.

Table 2. Model Building results after 20 epochs.

Model	val_loss	val_accuracy
Baseline Model	1.5396	0.7103 *(+2.12%)*
Less filters	1.2909	0.6971 *(+1.9%)*
5x5 Kernel	1.4281	0.7076 *(+2.1%)*
Batch Normalization	1.1969	0.7520 *(+2.48%)*
BN + 5x5 Kernel	0.9646	0.7737 *(+2.41%)*

From here, we can see that clearly BN + 5x5 Kernel yields the best results so we will go ahead with it and fine-tune it using the **hyper-parameter testing framework** Optuna [21]. The parameters we're going to be focusing on are the Adam optimizer learning rate and the dropout value.

After **running 37 trials with Optuna**, for 10 epochs each, these were the results:

- **Dropout**: 0.2
- **Learning Rate**: 0.0015
- ***val_loss****: 0.8661*
- ***val_accuracy****: 0.7549*

We get some good accuracy and loss measurements after just 10 epochs. So now, all there is left to do is let the model run for 100 epochs, with Early Stopping.

Early Stopping did end up triggering after 40 epochs. In the following graph we can observe both the 'loss' and 'accuracy' curves:

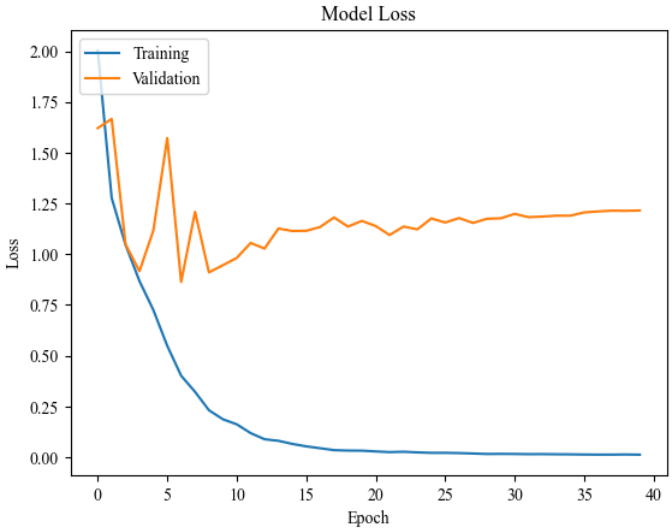

Fig. 13. Final CNN architecture model, training and validation loss.

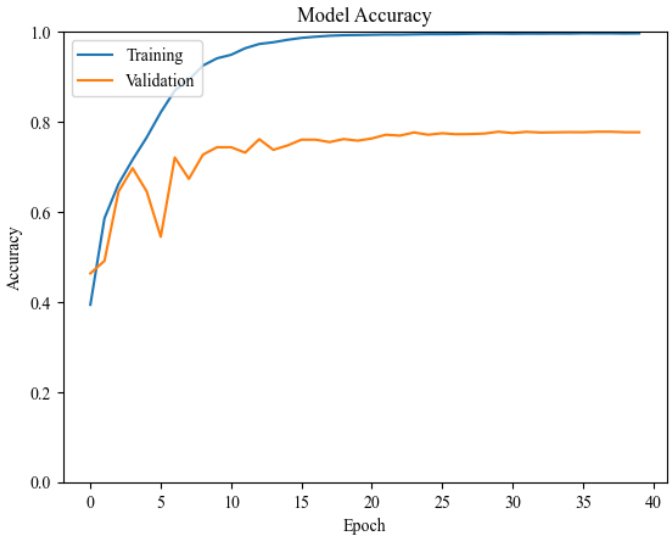

Fig. 14. Final CNN architecture model, training and validation accuracy.

We can clearly see that the training loss and accuracy are pretty much inversely correlated, with the training loss going down and accuracy going up. Interestingly, and it was something that we have noticed throughout the whole phase of model building, the **validation loss tends to stagnate**, or even go up, with **validation accuracy also increasing**, which doesn't seem very logical. It would be interesting to find the reason behind this.

In summary, the configuration that achieved the best accuracy was the following:

1. Conv2D (8) + MaxPooling with 5x5 Kernel
2. Conv2D (16) + MaxPooling with 5x5 Kernel
3. Conv2D (32) + MaxPooling with 5x5 Kernel
4. Conv2D (64) + MaxPooling with 5x5 Kernel
5. Conv2D (128) + MaxPooling with 5x5 Kernel
6. Dropout of 0.2
7. Softmax

All the convolutional layers have Batch Normalization, ReLu activation function and we are using Adam optimizer with learning rate of 0.0015.

The model's visual structure we can distinguish here:

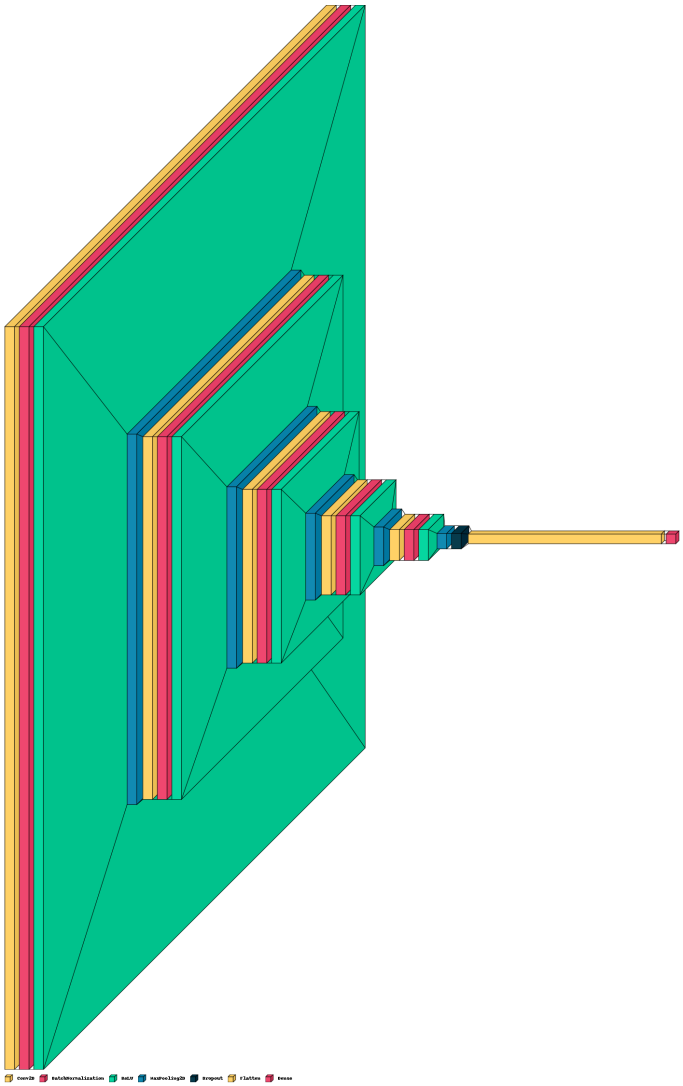

Fig. 15. Visual representation of the CNN.

Or here:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 511, 385, 8)       608

 batch_normalization (BatchN  (None, 511, 385, 8)      32
 ormalization)

 re_lu (ReLU)                (None, 511, 385, 8)       0

 max_pooling2d (MaxPooling2D  (None, 255, 192, 8)      0
 )

 conv2d_1 (Conv2D)           (None, 251, 188, 16)      3216

 batch_normalization_1 (Batc  (None, 251, 188, 16)     64
 hNormalization)

 re_lu_1 (ReLU)              (None, 251, 188, 16)      0

 max_pooling2d_1 (MaxPooling  (None, 125, 94, 16)      0
 2D)

 conv2d_2 (Conv2D)           (None, 121, 90, 32)       12832

 batch_normalization_2 (Batc  (None, 121, 90, 32)      128
 hNormalization)

 re_lu_2 (ReLU)              (None, 121, 90, 32)       0

 max_pooling2d_2 (MaxPooling  (None, 60, 45, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 56, 41, 64)        51264

 batch_normalization_3 (Batc  (None, 56, 41, 64)       256
 hNormalization)

 re_lu_3 (ReLU)              (None, 56, 41, 64)        0

 max_pooling2d_3 (MaxPooling  (None, 28, 20, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 24, 16, 128)       204928

 batch_normalization_4 (Batc  (None, 24, 16, 128)      512
 hNormalization)

 re_lu_4 (ReLU)              (None, 24, 16, 128)       0

 max_pooling2d_4 (MaxPooling  (None, 12, 8, 128)       0
 2D)

 dropout (Dropout)           (None, 12, 8, 128)        0

 flatten (Flatten)           (None, 12288)             0

 dense (Dense)               (None, 14)                172046

=================================================================
Total params: 445,886
Trainable params: 445,390
Non-trainable params: 496
```

Fig. 16. Keras Summary of our CNN.

We will now discuss the results of this model in the following chapter.

4 Results

4.1 GTZAN Dataset

The original GTZAN of 9 genres did very well on training using the baseline model outline before, reaching near 100% accuracy after 20 epochs. However, it lagged a little bit in testing/validation.

- **Training**: 99%
- **Testing**: 85%
- **Validation**: 87%

Here we can see the results per genre, with **Country, Hip Hop, Pop and Rock** lagging a little bit behind the rest. Genres in music are a very subjective matter, so an accuracy of 100% is not to be expected, and anything in the range of 75-100% should be considered as successful.

Table 3. Accuracy per genre for the original GTZAN dataset.

Genre	Accuracy
Blues	97%
Classical	95%
Country	78%
Disco	86%
Hip-Hop	81%
Metal	97%
Pop	81%
Reggae	88%
Rock	71%

We can dive deeper to see which genres are being confused with each other with a Confusion Matrix:

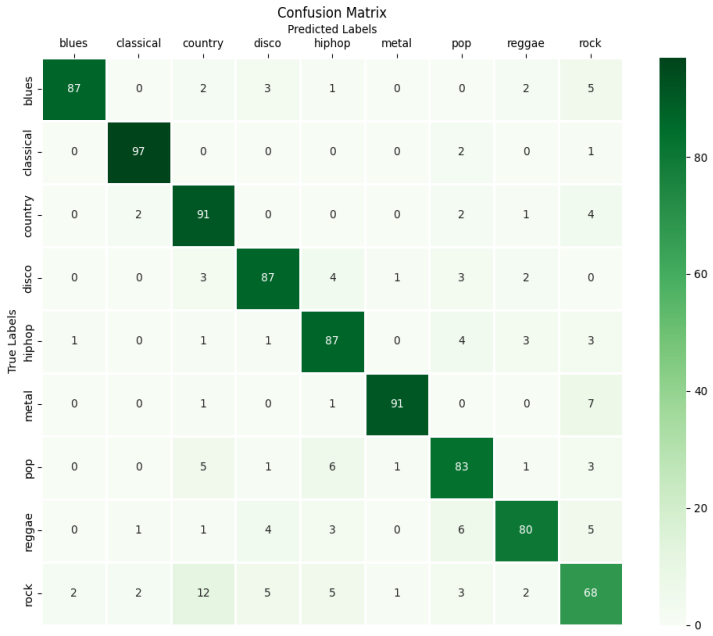

Fig. 17. Confusion Matrix for the Original Dataset.

Some interesting results we can observe in this image, like 'Rock' being mixed up with 'Country', but 'Country' is not being mixed with 'Rock'. 'Classical' is far and away the easiest genre to classify, as it is hard to get it mixed up with any other. 'Metal' is also confused with 'Rock', but 'Rock' isn't being confused with 'Metal'. All these little pieces of information can guide us in the right direction towards higher accuracies.

Finally, we will try some very popular songs from each genre, that have not been included in the dataset, to see how it would fare in the real world. First, a simple test, we will use one of Beethoven's symphonies:

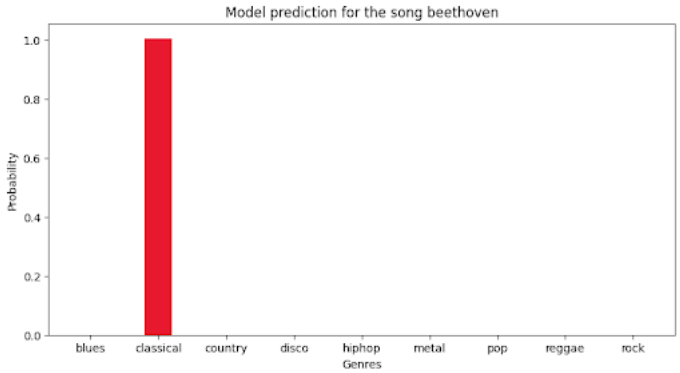

Fig. 18. Model prediction for Beethoven's 5th Symphony.

As expected, no problems classifying it as 'Classical'.

Another test, Coldplay's 'Viva la Vida':

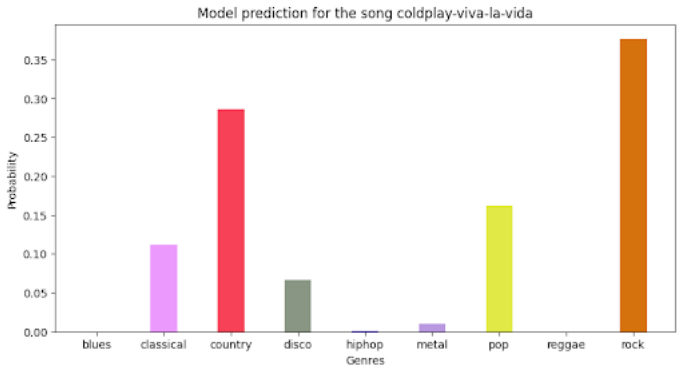

Fig. 19. Model Prediction for the song Viva la Vida by Coldplay.

Viva la Vida is an interesting song, because although it is regarded as 'Rock' it has a lot of elements from other genres, and our model does a great job at identifying most of them. From 'Pop' to 'Classical', Viva la Vida is more a mix of genres, and we can determine our model to be successful in navigating all of them.

Finally, we have a special case we want to test. Apart from her pregnancy, Rihanna shocked everyone at this year's Super Bowl half-time show with one of her hit songs 'Rude Boy'. But it wasn't the original version of the song, it was the Brazilian Funk remix.

So we thought it would be interesting to see how our model performs with the original song versus the remix. First, the original version:

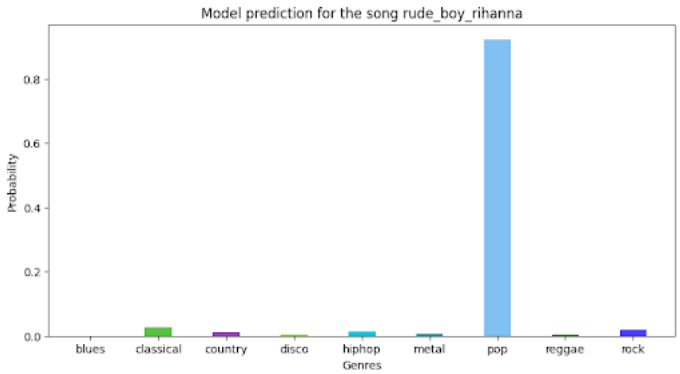

Fig. 20. Model Prediction for Rihanna's hit 'Rude Boy'.

Correctly classifies it as 'Pop'. How will it do with the Brazilian Funk remix?

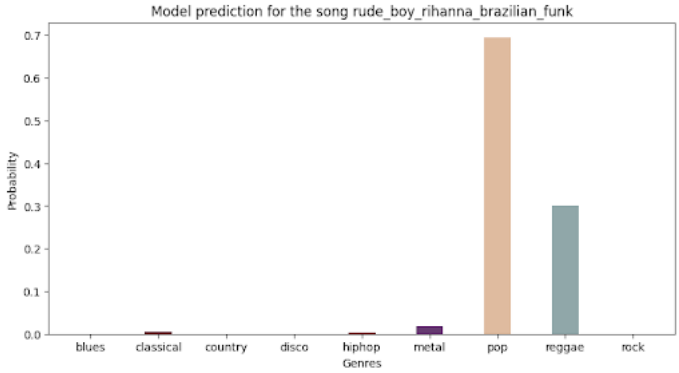

Fig. 21. Model Prediction for the Brazilian Funk remix of Rihanna's hit 'Rude Boy'.

It still classifies it as 'Pop', but it has been able to detect that it is not 100% Pop anymore, and can identify some 'Reggae' in it, which, without having been trained to classify Brazilian Funk yet, is a pretty good approximation.

On the custom dataset we included Brazilian Funk and Rap, so let's see how that worked out.

4.2 Custom GTZAN Dataset

Results using the Custom Dataset are surprisingly good. Even when using 11 genres (2 more on top of the GTZAN dataset, which were sourced from Spotify) the model still performs great. Better than the original dataset, possibly due to having more genres to distinguish from:

- **Training**: 99%
- **Testing**: 88%
- **Validation**: 87%

If we look at the accuracy per class, we can see that both Brazilian Funk and Rap integrated perfectly, both with more than 90% accuracy. Country and Rock are still lagging, so more training data would do them good.

Table 4. Accuracy per genre for the custom dataset.

Genre	Accuracy
Blues	87%
Brazilian Funk	94%

Classical	97%
Country	78%
Disco	87%
Hip-Hop	84%
Metal	96%
Pop	94%
Rap	92%
Reggae	81%
Rock	73%

The confusion matrix:

Fig. 22. Confusion Matrix for the Custom Dataset.

And now we perform the same test on popular songs as with the original dataset. First, Beethoven:

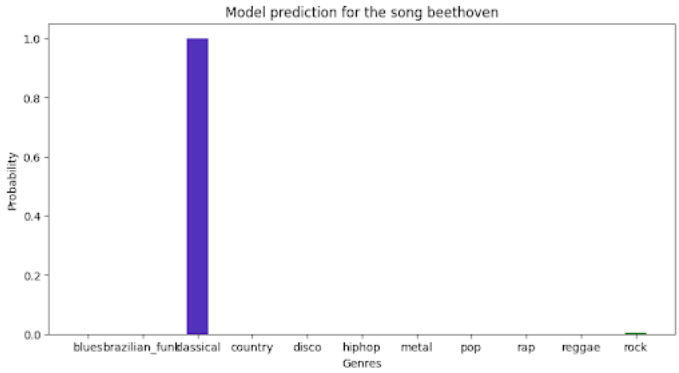

Fig. 23. Model Prediction for Beethoven's 5th Symphony.

No surprise here. Next, Viva la Vida:

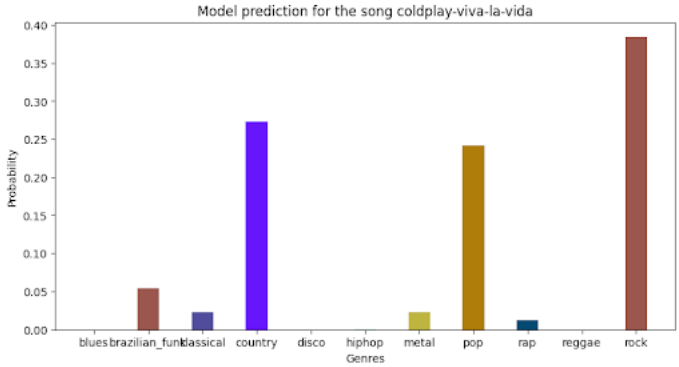

Fig. 24. Model Prediction for Coldplay's 'Viva la Vida'.

Finally, Rude Boy, original and remix, with the model being able to detect the Brazilian Funk in the song successfully.

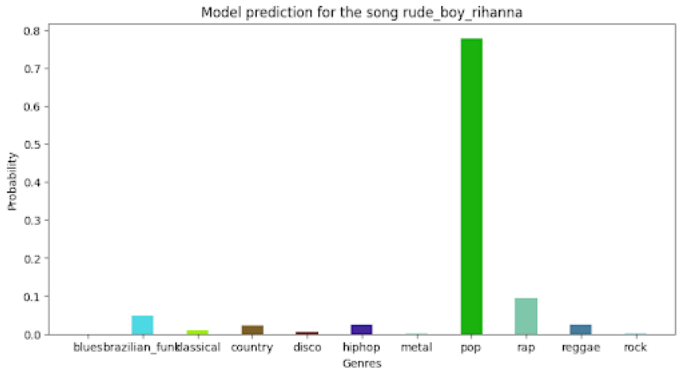

Fig. 25. Model Prediction for Rihanna's 'Rude Boy'.

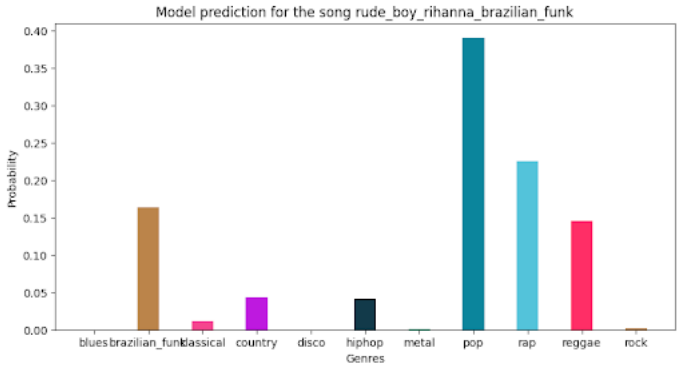

Fig. 26. Model Prediction for the Brazilian Funk remix of Rihanna's 'Rude Boy'.

4.3 Spotify 3-Sec Dataset

After Early Stopping at 40 epochs, the model performed like this on the Spotify 3-second dataset:

- **Training**: 99%
- **Testing**: 78%
- **Validation**: 77%

The accuracy per class was:

Table 5. Accuracy per genre for the Spotify 3-Second Dataset.

Genre	Accuracy
Classic	93%
Country	72%
EDM	83%

Flamenco	91%
Hip-Hop	76%
Indie	73%
Jazz	90%
K-Pop	73%
Latin	83%
Metal	72%
Pop	52%
Rap	83%
Reggae	93%
Rock	62%

Where we can see that ‘Pop’ was the big outlier in terms of performance, mainly due to the fact that ‘Pop’ conforms such a large and vast spectrum that other genres like ‘Hip-Hop’, ‘K-Pop’, ‘Rap’ also use elements of it.

The confusion matrix for each class:

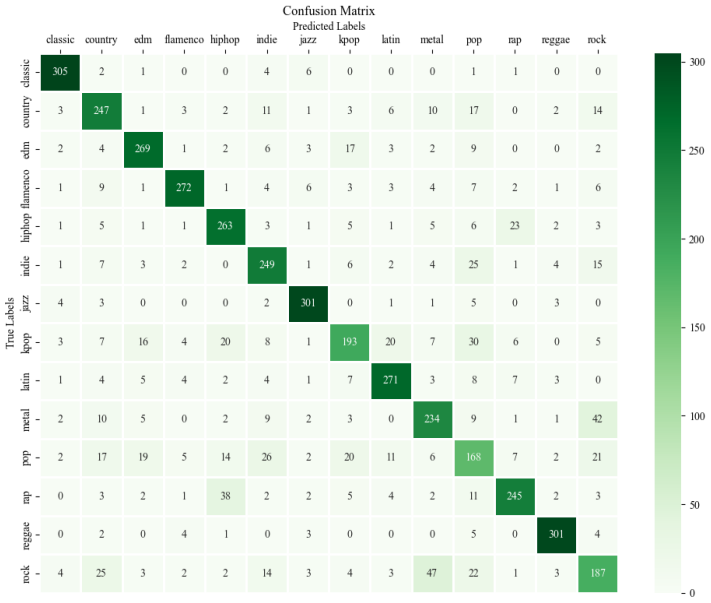

Fig. 27. Confusion Matrix for the Spotify 3-Second Dataset.

And finally, we will test on the following popular songs to see how it performs in the real world, after being trained a sparse number of genres:

- ‘Rude Boy’ by **Rihanna**.
- ‘5th Symphony’ by **Beethoven**.
- ‘Viva la Vida’ by **Coldplay**.
- ‘Silhouettes’ by **Avicii**.
- **Blackpink** K-Pop song I will not dare write down its title.
- ‘Nothing else matters’ by **Metallica**.

To process **popular songs**, what we are going to do is, convert from MP3 to the correct WAV format, split the song into 3-second clips, generate the Mel spectrogram of each, **test each individual slice of the song**, predict the class with our model and then add up the probabilities. Here is the complete code to process the songs:

```
def test_song(song, model):
    name, ext = os.path.splitext(song)
    to_delete, intervals = process_wav(song)
    files_to_delete = ["{0}_resampled.wav".format(name)]
    files_to_delete = files_to_delete + to_delete

    probabilities = np.zeros(len(list(train_generator.class_indices.keys())))

    images_to_delete = []

    for mini_part in files_to_delete:
        spectogram = generate_spectogram_for_song(mini_part)
        images_to_delete.append(spectogram)
        image = Image.open(spectogram)
        im_arr = np.frombuffer(image.tobytes(), dtype = np.uint8)
        im_arr = im_arr.reshape((image.size[0], image.size[1], 3))
        image = im_arr / 255.0  #Normalize the image
        image = np.expand_dims(image, axis = 0)
        random_test = image
        y_proba = model.predict(random_test, verbose = 0)
        np.set_printoptions(suppress = True)
        probabilities = probabilities + y_proba[0]

    files_to_delete = files_to_delete + images_to_delete
    for file_to_delete in files_to_delete:
        os.remove(file_to_delete)

    hexadecimal_alphabets = "0123456789ABCDEF"

    colors = ["#" + "".join([random.choice(hexadecimal_alphabets) for j in range(6)])
for i in range(9)]

    plt.clf()
```

```
    fig = plt.figure(figsize = (10, 5))

    # Create the bar plot with random colors for each
    plt.bar(list(train_generator.class_indices.keys()), probabilities / intervals, color
= colors, width = 0.4)

    plt.xlabel("Genres")
    plt.ylabel("Probability")
    plt.title("Model prediction for the song {}".format(song.replace(".wav", "")))
    plt.show()

    return probabilities / intervals
```

And the results:

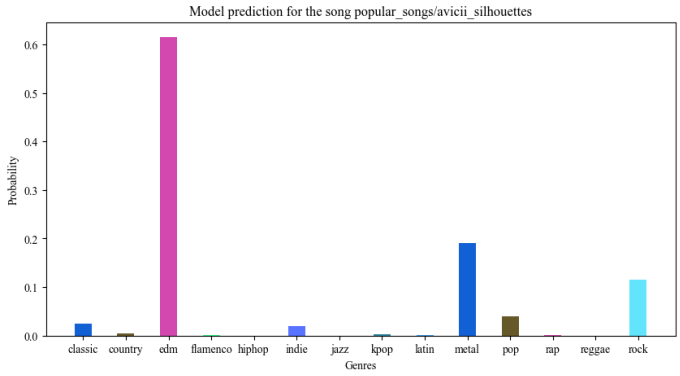

Fig. 28. 3-Second Model prediction for ‘EDM’ song ‘Silhouettes’ by Avicii.

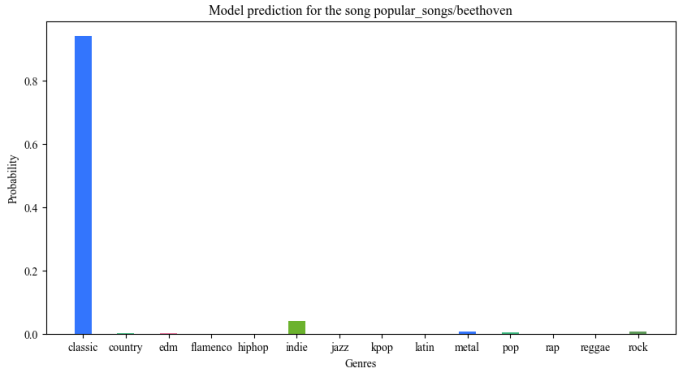

Fig. 29. 3-Second Model prediction for ‘Classical’ song ‘5th Symphony’ by Beethoven.

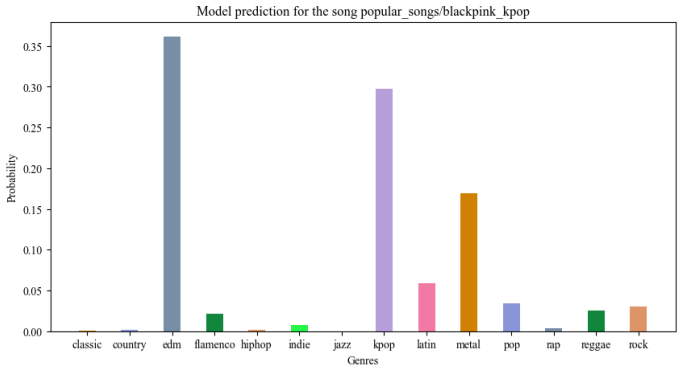

Fig. 30. 3-Second Model prediction for ‘K-Pop’ song by Blackpink.

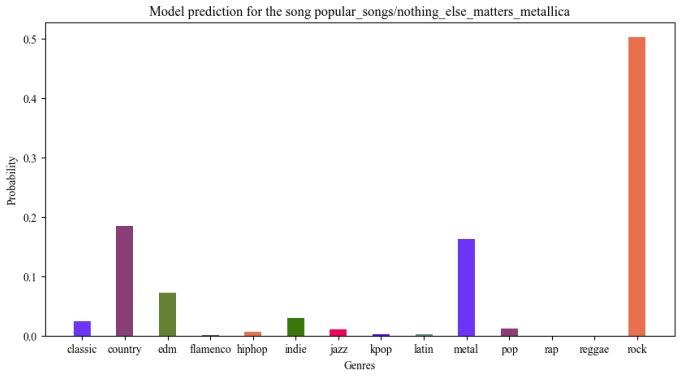

Fig. 31. 3-Second Model prediction for ‘Rock’ song ‘Nothing else matters’ by Metallica.

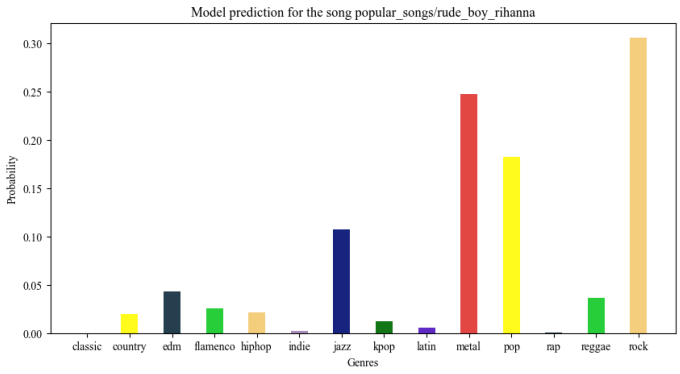

Fig. 32. 3-Second Model prediction for ‘Pop’ song ‘Rude Boy’ by Rihanna.

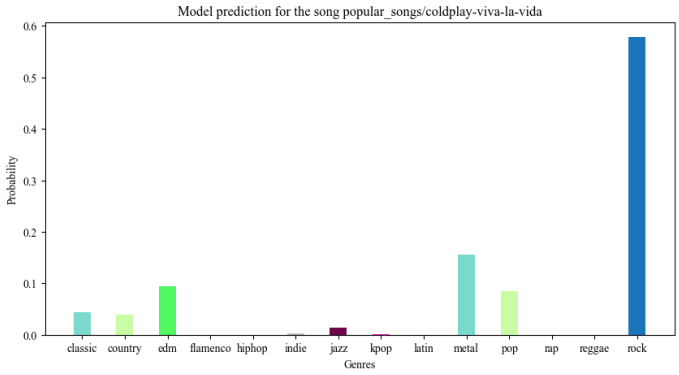

Fig. 33. 3-Second Model prediction for 'Rock' song 'Viva la vida' by Coldplay.

4.4 Spotify 9-Sec Dataset

For this alternative dataset, we will retrain the final model we obtained in the Model Building phase for the new 9-second Mel spectrograms and compare the results with the previous 3-second Spotify dataset.

Early Stopping triggered 34 epochs into training, with the following accuracy metrics:

- **Training**: 99%
- **Testing**: 75%
- **Validation**: 75%

Slightly worse results than the 3-second dataset, but let's take a deep dive into the results. First, the accuracies per genre:

Table 6. Accuracy per genre for the Spotify 9-Second Dataset.

Genre	Accuracy
Classic	89%
Country	68%
EDM	82%
Flamenco	84%
Hip-Hop	76%
Indie	71%
Jazz	90%
K-Pop	72%
Latin	81%
Metal	71%
Pop	36%

Rap	74%
Reggae	89%
Rock	58%

Again, very poor results for ‘Pop’ and ‘Rock’. It makes sense however, as they are probably the biggest two genres in this list, so a lot of overlapping is naturally occurring.

The confusion matrix:

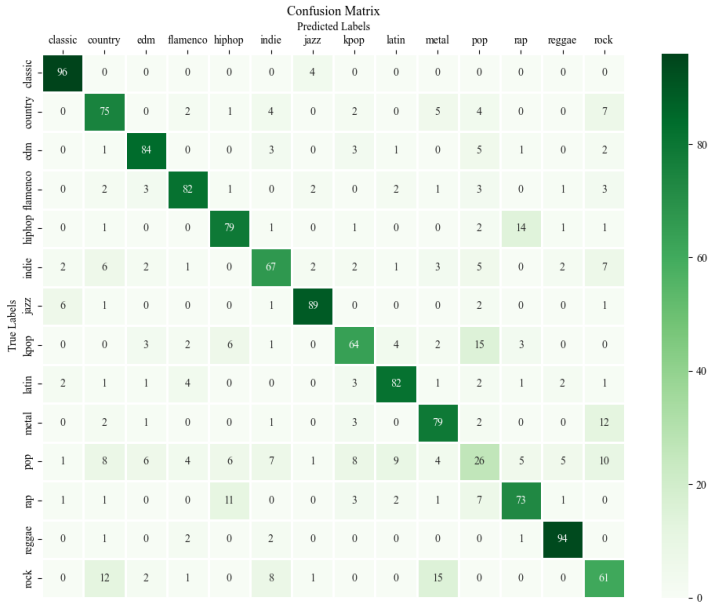

Fig. 34. Confusion Matrix for the Spotify 9-Second Dataset.

And finally, the popular songs:

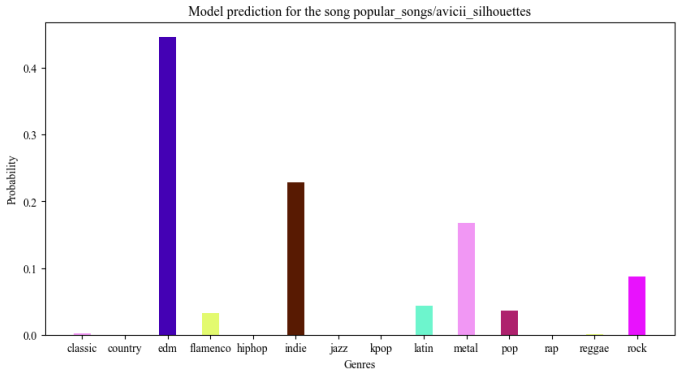

Fig. 35. 9-Second Model prediction for ‘EDM’ song ‘Silhouettes’ by Avicii.

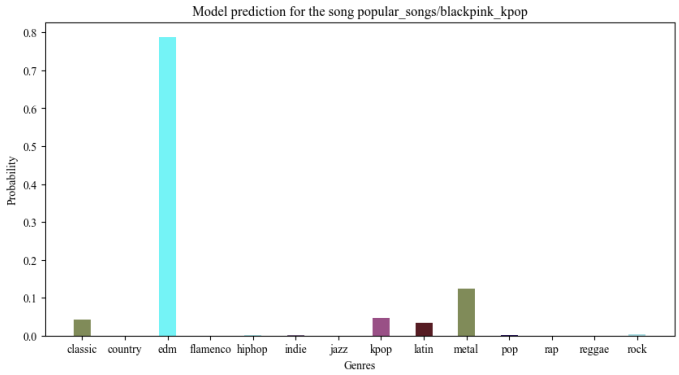

Fig. 36. 9-Second Model prediction for ‘K-Pop’ song by Blackpink.

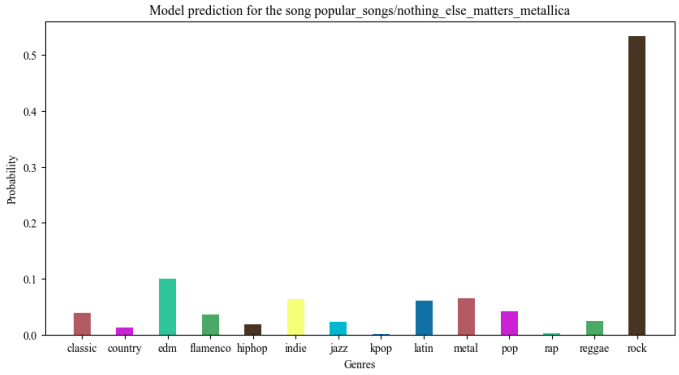

Fig. 37. 9-Second Model prediction for ‘Rock’ song ‘Nothing else matters’ by Metallica.

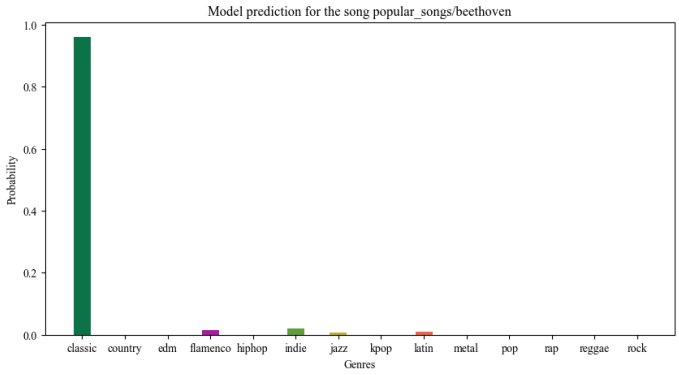

Fig. 38. 9-Second Model prediction for 'Classical' song '5th Symphony' by Beethoven.

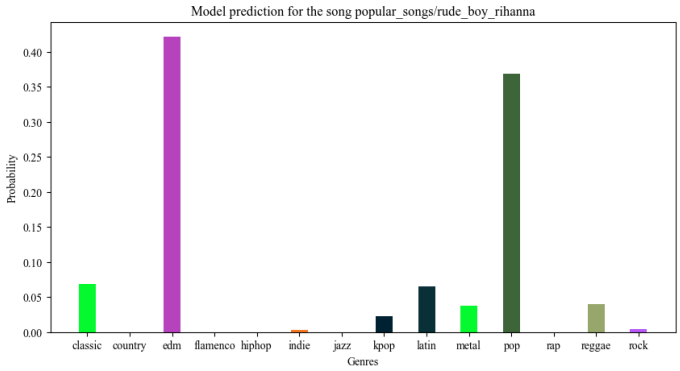

Fig. 39. 9-Second Model prediction for 'Pop' song 'Rude Boy' by Rihanna.

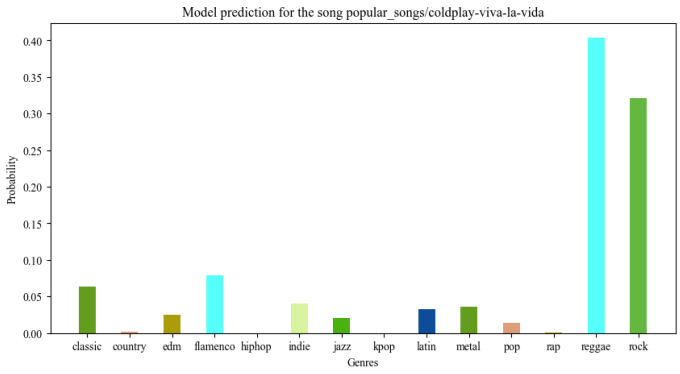

Fig. 40. 9-Second Model prediction for 'Rock' song 'Viva la Vida' by Coldplay.

5 Conclusions

Overall, we are very happy with the outcome of this project. We are able to detect the genres in popular songs outside of the dataset and even distinguish remixes from the original song. We have learned tremendously in terms of building a dataset from scratch, working with Convolutional Neural Networks, overfitting and all the general problems that arise when doing Machine Learning.

Regarding future improvements, we feel like there are a lot of things that can be done to expand on our research, like:

- **Extract more features from the audio**. Right now, the model is only trained on the Mel spectrogram, but there are many more features we can extract from the audio that would allow it to predict even better.
- **Better network structure**. Although a lot of different approaches have been attempted at improving accuracy and we believe this base model is good, it is also true that with even more testing we could extract a few more % points in terms of accuracy and we encourage any reader of this paper to try by themselves some of the approaches we have not covered here.

In conclusion, we deem this a successful project that has enriched our knowledge of CNNs and audio manipulation. For anyone interested in checking the actual source code or the Jupyter Notebook, **please refer to the Github for this project** [22].

References

1. **Music Genre Recognition with CNNs**, https://towardsdatascience.com/music-genre-recognition-using-convolutional-neural-networks-cnn-part-1-212c6b93da76, last accessed 2023/05/02.
2. **Generating Mel Spectrogram from a .WAV file,** https://www.kaggle.com/code/msripooja/steps-to-convert-audio-clip-to-spectrogram/notebook, last accessed 2023/05/02.
3. **GTZAN Dataset**, https://www.tensorflow.org/datasets/catalog/gtzan, last accessed 2023/05/02.
4. **SpotDL**, https://github.com/spotDL/spotify-downloader, last accessed 2023/05/03.
5. **Spotify Playlist 'Best Rap Songs of 2000',** https://open.spotify.com/playlist/37i9dQZF1DX38t16fuNXJJ?si=8f37db54aa854c1d, last accessed 2023/05/02.
6. Sofia Léo Moreira, **Spotify Playlist 'Brazilian Funk'**, https://open.spotify.com/playlist/49eCUcvvK1NJbzn8LewQma?si=922703e560744040, last accessed 2023/05/02.
7. **Spotify Playlist 'Best Classical Music of All Time',** https://open.spotify.com/playlist/0bJvpsn0TDZwIDUjz4d75S?si=081c9ab8c8c543e0, last accessed 2023/05/02.
8. **Spotify Playlist 'Best country songs (greatest hits)',** https://open.spotify.com/playlist/1mJhSx6aYQmINsZ8dG4gzU?si=3f396c2d95a94dd8, last accessed 2023/05/02.

9. **Spotify Playlist 'EDM Classics (TOP 100)',** https://open.spotify.com/playlist/1lS6v9h4MXOw6f6y8MkS8w?si=3df5a281e1e147fd, last accessed 2023/05/02.
10. **Spotify Playlist 'Clasicos Flamenco',** https://open.spotify.com/playlist/4INDh1s1809UtfNOmj03ii?si=69e65a33d6fd42fd, last accessed 2023/05/02.
11. **Spotify Playlist 'Best HipHop 2000s',** https://open.spotify.com/playlist/4vzXcD9NhOfCx3tOkFgKT0?si=56d59db8889741df, last accessed 2023/05/02.
12. **Spotify Playlist 'Best indie songs of all time',** https://open.spotify.com/playlist/0Sm64Lu6z1OK8yM3Oeo4Wx?si=11fd4f406f344c20, last accessed 2023/05/02.
13. **Spotify Playlist 'Jazz Classics',** https://open.spotify.com/playlist/37i9dQZF1DXbITWG1ZJKYt?si=fda4ed31c0c541b5, last accessed 2023/05/02.
14. **Spotify Playlist 'Greatest kpop hits of all time'**, https://open.spotify.com/playlist/1HmNOUNQKVnAr2Y0GbHS7b?si=db92ba6356dd4687, last accessed 2023/05/02.
15. **Spotify Playlist 'Best Latin Songs of 2022',** https://open.spotify.com/playlist/37i9dQZF1DX8L1VmOcEBjS?si=c9ab61ae651641c5, last accessed 2023/05/02.
16. **Spotify Playlist 'Best Metal Songs of All Time',** https://open.spotify.com/playlist/1yMlpNGEpIVUIilZlrbdS0?si=3c832309535b4dcd, last accessed 2023/05/02.
17. **Spotify Playlist 'Best Pop Songs of All Time',** https://open.spotify.com/playlist/6vI3xbpdPYYJmicjBieLcr?si=106446bb54d8488b, last accessed 2023/05/02.
18. **Spotify Playlist 'Best Rap Songs of All Time',** https://open.spotify.com/playlist/4kG4PPMg4vAEXdeRbnsDmo?si=6f217cc6da584cdd, last accessed 2023/05/02.
19. **Spotify Playlist 'Reggae classics 70s 80s & 90s',** https://open.spotify.com/playlist/71R43lBYQZ6JQXH6LmRo1I?si=95e7a6da92ed4627, last accessed 2023/05/02.
20. **Spotify Playlist 'Best Rock Songs of All Time',** https://open.spotify.com/playlist/35eZS8blAMfzR7ZNO9WjKN?si=361626c4a0114349, last accessed 2023/05/02.
21. **Optuna**, https://optuna.org/, last accessed 2023/05/03.
22. **Music Genre Recognition using CNNs on GTZAN + Spotify,** https://github.com/mrodriguez2/Music-Genre-Recognition-on-GTZAN-Spotify, last accessed 2023/05/04.