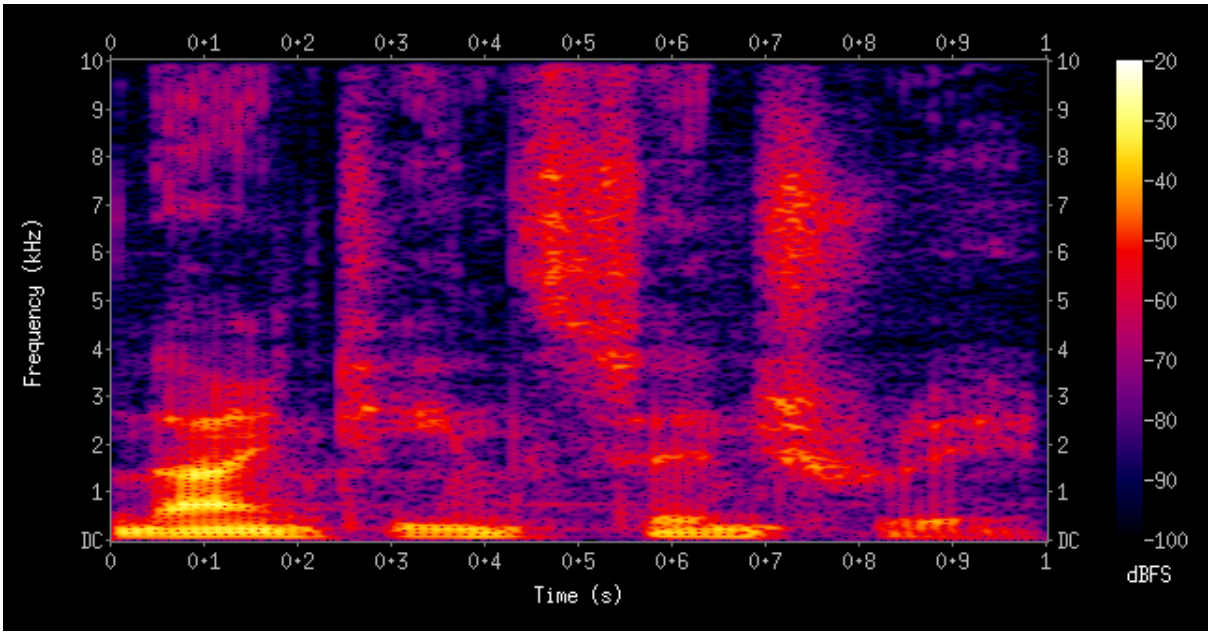
# CS767 - Music Genre Recognition

*Manuel Rodríguez Rodríguez*




## Project Description

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 the GTZAN dataset 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
- Hiphop
- Metal
- Reggae

As 100 samples per class is a very small sample size, we will split the 30-second clips into 10 3-second clips each, making it 1000 samples per class.

Once we have this, we will generate the image representation of a song, also known as a **mel spectrogram**, which covers all the main characteristics that represent sound. The idea

is that our CNN will be able to detect patterns in these spectrograms that allow it to differentiate between genres.

We will use this base dataset to find the optimal neural network structure. From there on, we will amplify the dataset with more songs from those same genres and create new genres using Spotify, and compare the results.

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 life partner.** 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 vs 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**.

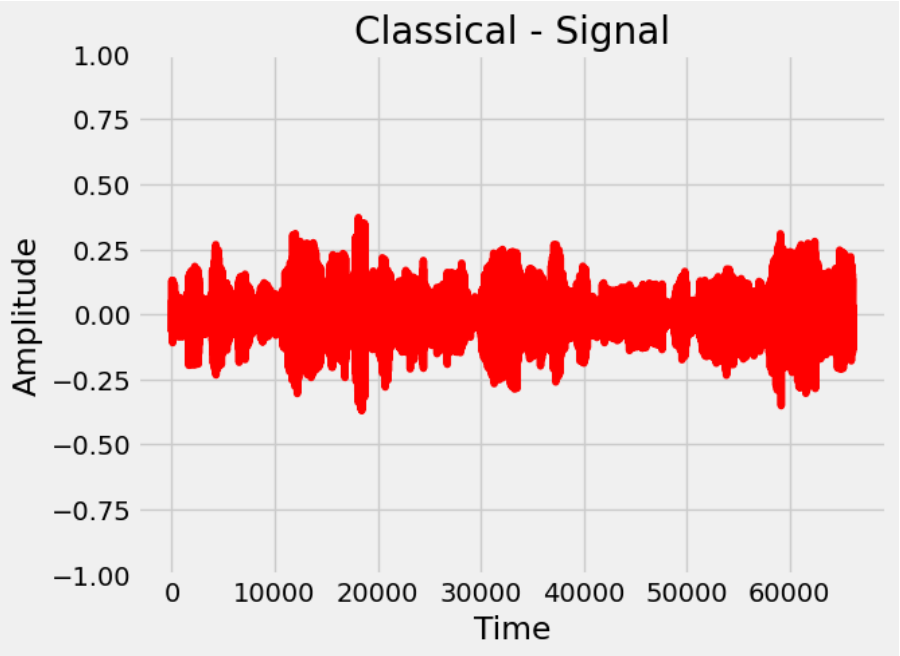
Most importantly however, we hope to use this project as a learning opportunity to improve our knowledge on Machine Learning, Artificial Intelligence and Neural Networks.

## What is a Mel Spectrogram?

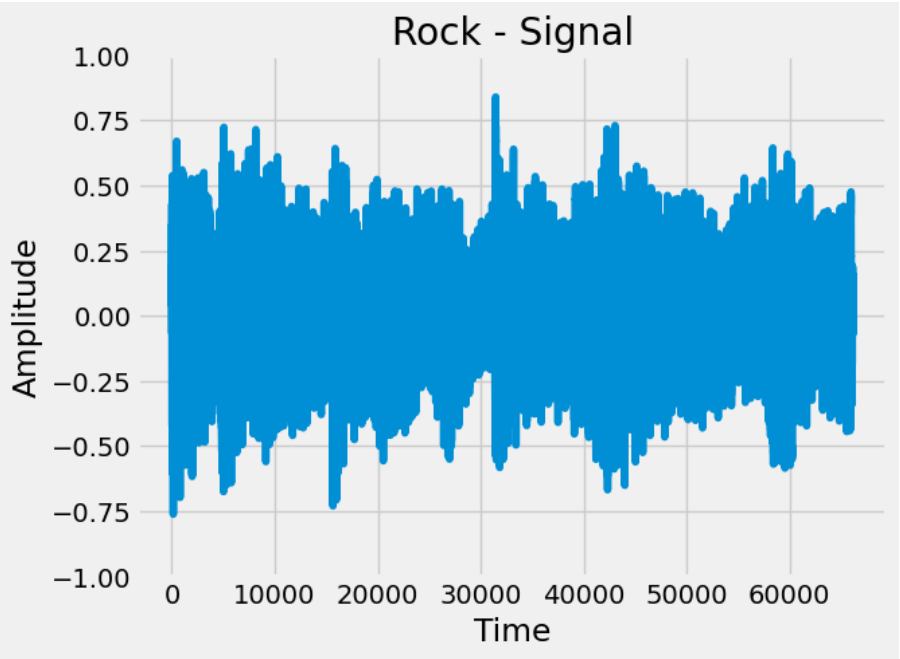
To train our CNN, we need to transform the audio into an image that the network will understand. To do this, we will need to generate a mel spectrogram, which manages to capture a lot of details about the audio.

To explain what a mel spectrogram is, we first have to understand the different steps involved in the process.

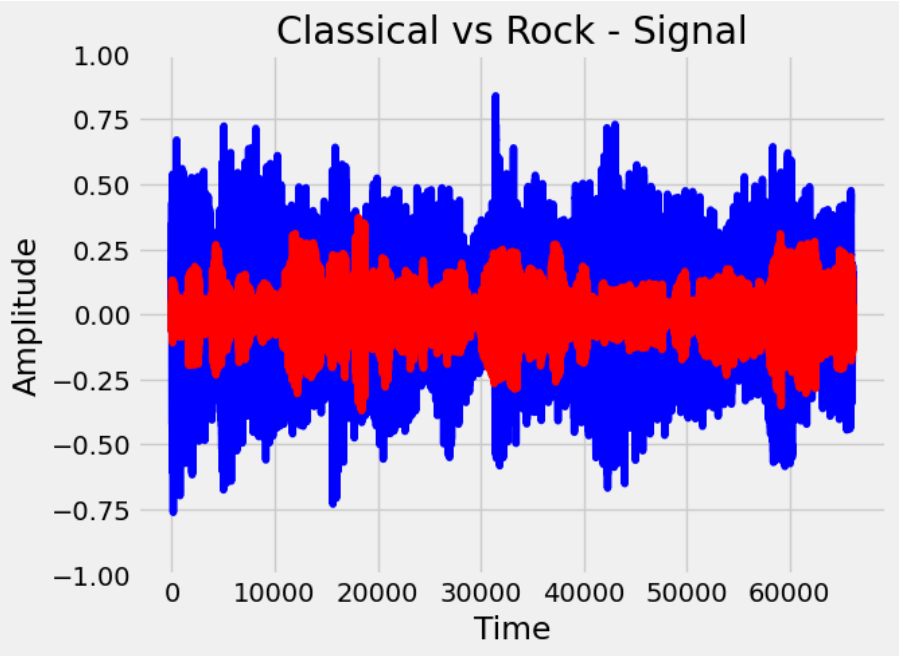
First, we take 22.050 samples (22kHz) per second of the variations in air pressure of the song. This generates the following waveform, which everyone is used to, where the x-axis is the time, and the y-axis is the amplitude. The higher the amplitude, the louder the song is at that point in time.




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

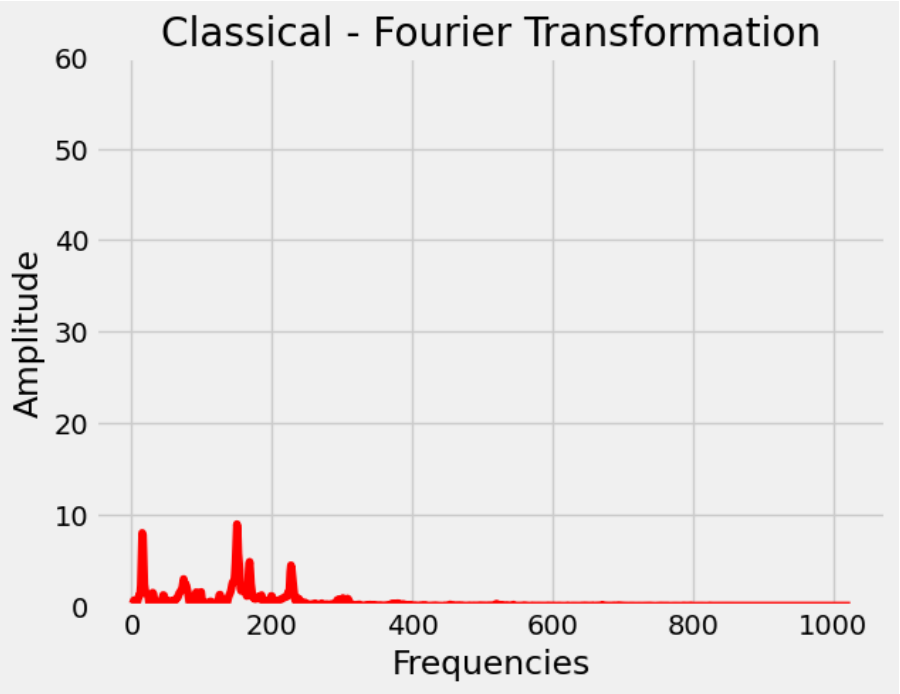



The problem with these waveforms is that they only capture the loudness of the audio, as an aggregate of all combined frequencies.

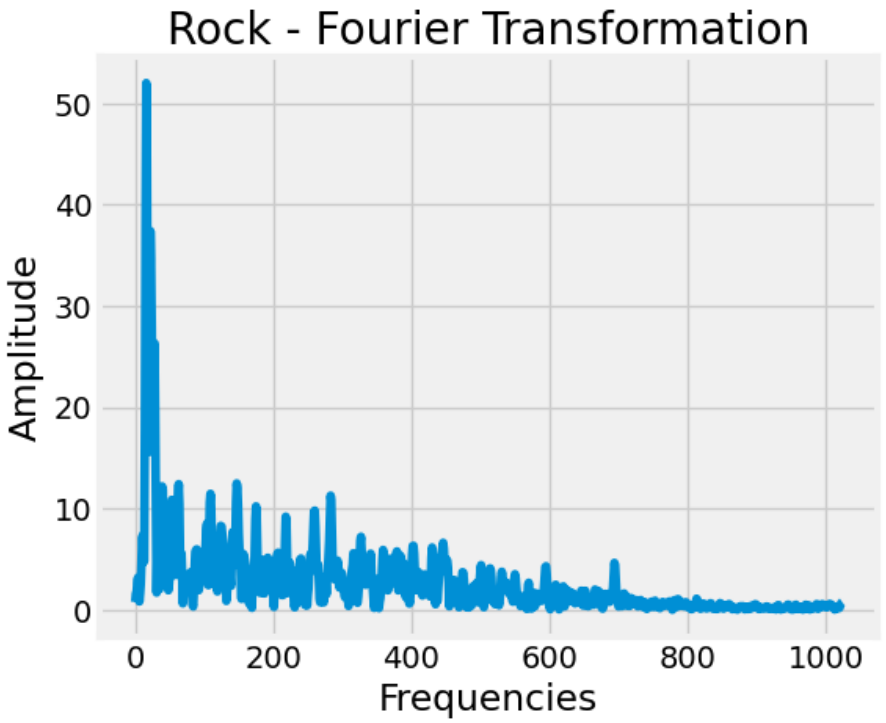



So yes, Rock songs tend to be louder than classical songs, but that's not nearly enough information to classify genres. To solve this, and gain more information on what frequencies the song is playing at, we apply a Fourier Transformation.

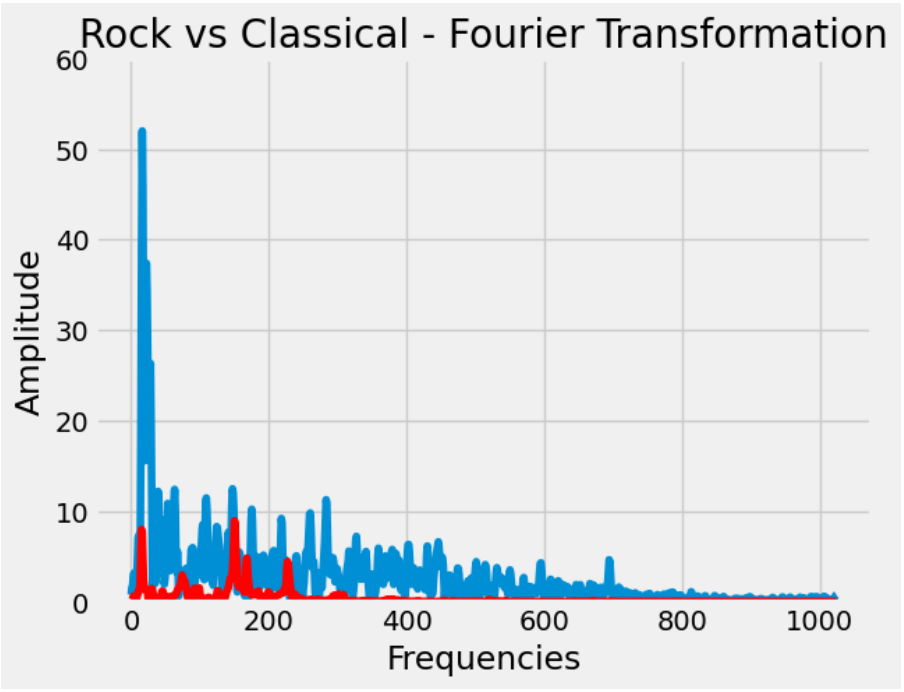
The explanation of this mathematical formula escapes the purpose of this project, but the Fourier Transformation basically gets the signal for each time input, and returns the decomposition of frequencies of the signal at each time point.




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:

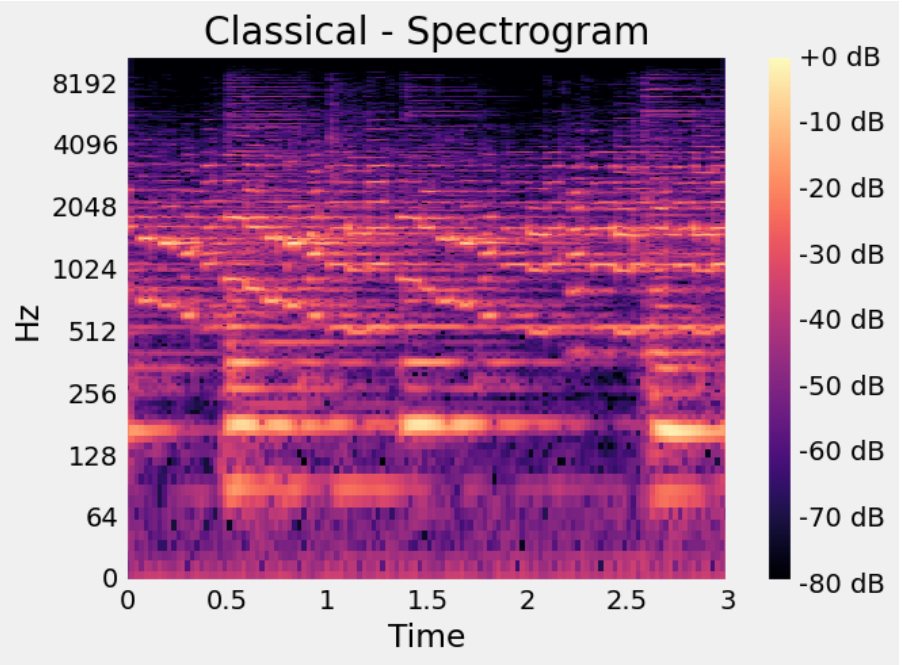



And the overlay:

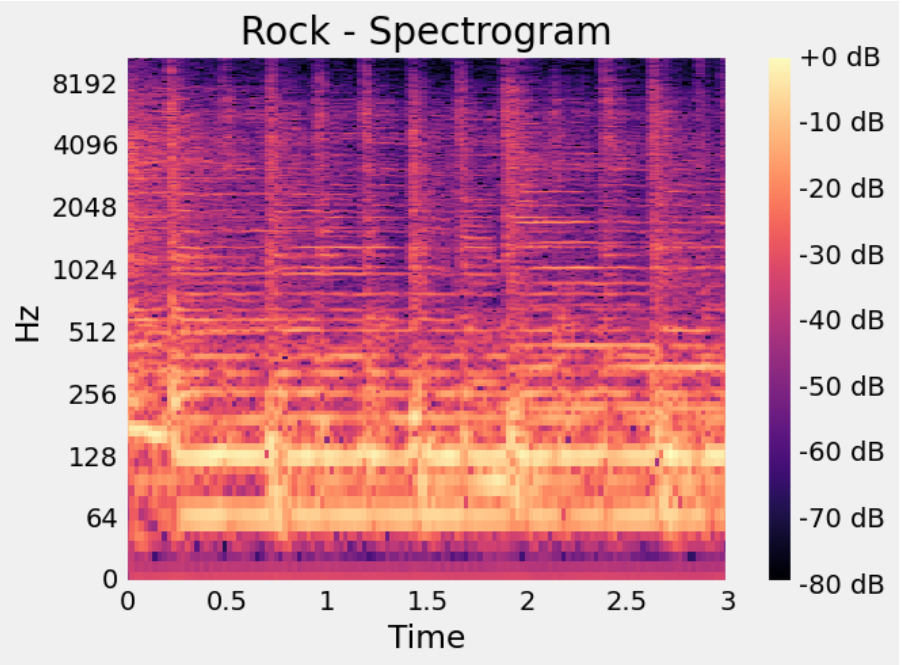



From this, we generate the spectrogram by applying the Fourier Transformation at every point in time, obtaining the decomposition of frequencies. We also convert things to log-scale so that it's easier to visualize, as numbers can get very big and we apply a color gradient.

This is the result, 'Classical' spectrogram:



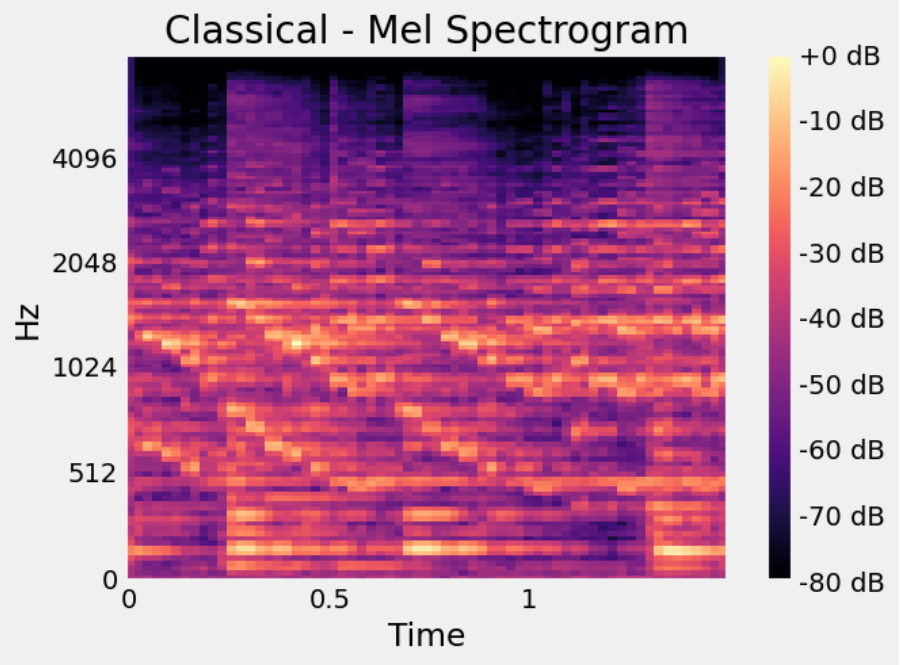

Versus a ‘Rock’ spectrogram:



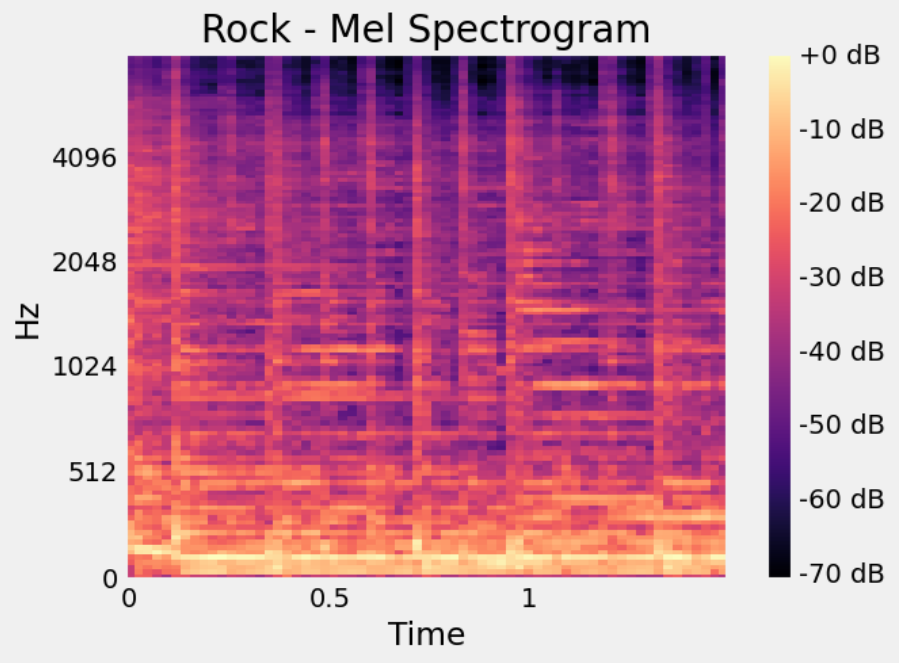

But this generates a spectrogram, where for each unit of time we have the amplitude in Decibels of each frequency. We’re still missing the **mel** in *Mel Spectrogram*.

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.




And the ‘Rock’ mel spectrogram:




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

# Dataset

As mentioned previously, we used the GTZAN dataset, 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
- Hiphop
- 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.

For each of the 9000 samples, we generated the corresponding mel spectrogram, splitting the dataset 80-10-10. So this meant 7200 samples used for training, 900 used for validation and 900 used for testing. This is included under the ‘original_dataset’ folder.

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 we downloaded the following Spotify playlists in MP3 format, converted them to the correct WAV, 22kHz, Mono, 16-bit files and added them to the dataset. These are included in the ‘custom_dataset’ folder.

The sources for each of the files used to train this model are here:

Rap playlist:
https://open.spotify.com/playlist/37i9dQZF1DX38t16fuNXJJ?si=8f37db54aa854c1d

Brazilian Funk playlist:
https://open.spotify.com/playlist/49eCUcvvK1NJbzn8LewQma?si=922703e560744040

GTZAN dataset: https://www.tensorflow.org/datasets/catalog/gtzan

# Model

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

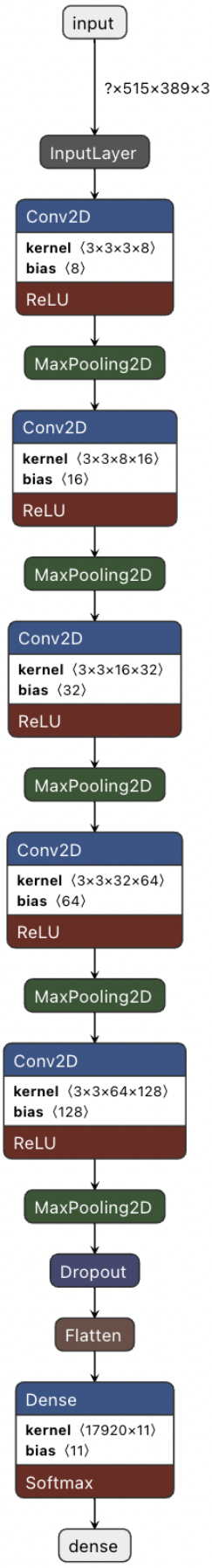
- Conv2D + MaxPooling with increasing filters
- Conv2D + MaxPooling with decreasing filters
- Conv2D + MaxPooling with the same number of filters

- Adding more Dense ReLu layers after the sequence of Conv2D + MaxPooling
- Batch Normalization & Dropout
- ResNet50

But the configuration that achieved the best results was the following:

1. Conv2D (8) + MaxPooling
2. Conv2D (16) + MaxPooling
3. Conv2D (32) + MaxPooling
4. Conv2D (64) + MaxPooling
5. Conv2D (128) + MaxPooling
6. Dropout of 0.3
7. Softmax

Which we can distinguish visually here:


input
?×515×389×3
InputLayer
Conv2D
kernel 〈3×3×3×8〉
bias 〈8〉
ReLU
MaxPooling2D
Conv2D
kernel 〈3×3×8×16〉
bias 〈16〉
ReLU
MaxPooling2D
Conv2D
kernel 〈3×3×16×32〉
bias 〈32〉
ReLU
MaxPooling2D
Conv2D
kernel 〈3×3×32×64〉
bias 〈64〉
ReLU
MaxPooling2D
Conv2D
kernel 〈3×3×64×128〉
bias 〈128〉
ReLU
MaxPooling2D
Dropout
Flatten
Dense
kernel 〈17920×11〉
bias 〈11〉
Softmax
dense

Or here:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 513, 387, 8)       224

 max_pooling2d (MaxPooling2D  (None, 256, 193, 8)      0
 )

 conv2d_1 (Conv2D)           (None, 254, 191, 16)      1168

 max_pooling2d_1 (MaxPooling  (None, 127, 95, 16)      0
 2D)

 conv2d_2 (Conv2D)           (None, 125, 93, 32)       4640

 max_pooling2d_2 (MaxPooling  (None, 62, 46, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 60, 44, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 30, 22, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 28, 20, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 14, 10, 128)      0
 2D)

 dropout (Dropout)           (None, 14, 10, 128)       0

 flatten (Flatten)           (None, 17920)             0

 dense (Dense)               (None, 11)                197131

=================================================================
Total params: 295,515
Trainable params: 295,515
Non-trainable params: 0
```

On top of that, we tried Adam and SGD with Nesterov as optimizers, with Adam the clear winner. We also used different Learning Rate Schedulers and found best results with the Reduce On Plateau at a factor of 0.8.

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

# Results

For this chapter, we follow the following structure to present our results

- Original Dataset - training, testing, validation accuracy
- Original Dataset - Precision per class
- Original Dataset - Confusion Matrix per class (which genres are more confused with which?)
- Original Dataset - Popular Songs
- Custom Dataset - training, testing, validation accuracy
- Custom Dataset - Precision per class
- Custom Dataset - Confusion Matrix per class (which genres are more confused with which?)
- Custom Dataset - Popular Songs

## Original Dataset

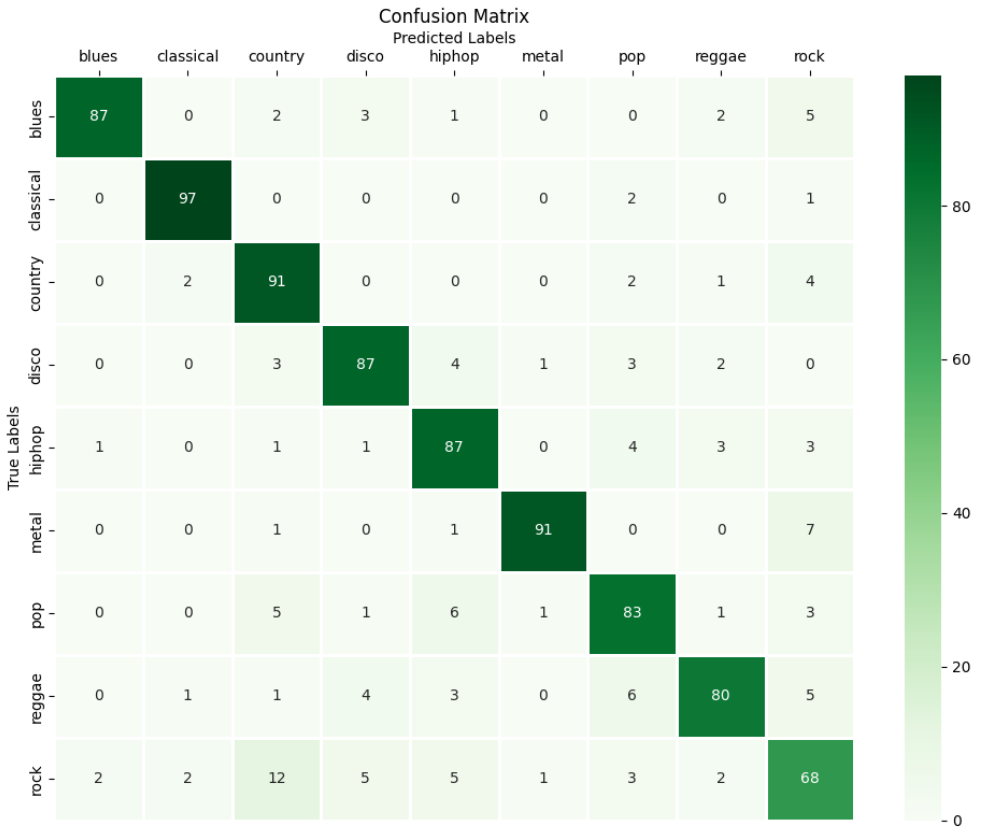
The original dataset of 9 genres did very well on training, reaching near 100% accuracy after 20 epochs. However, it lagged a little bit in testing/validation.

- **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 is 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.

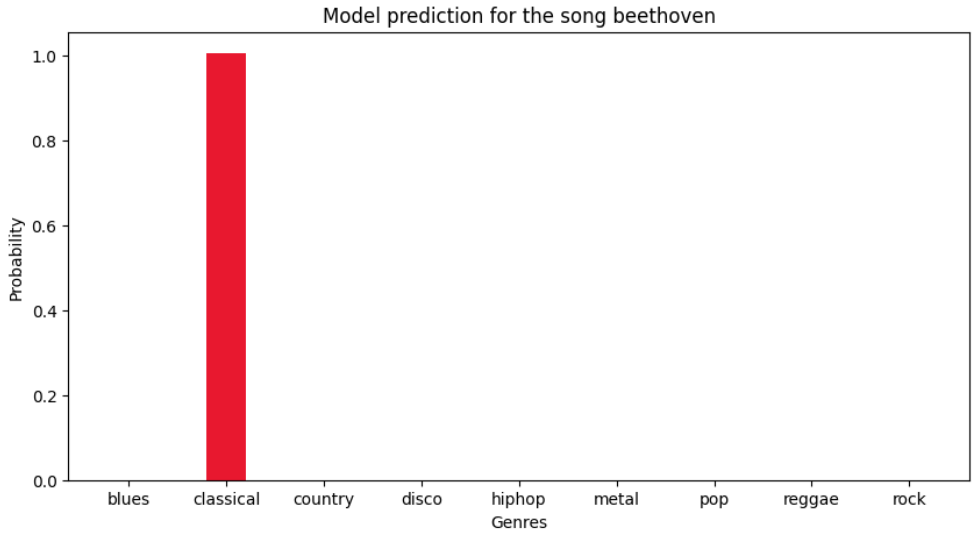
| | **Accuracy** |
|---|---|
| **Blues** | 97% |
| **Classical** | 95% |
| **Country** | 78% |
| **Disco** | 86% |
| **Hip Hop** | 81% |
| **Metal** | 97% |
| **Pop** | 81% |
| **Reggae** | 88% |
| **Rock** | 71% |

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



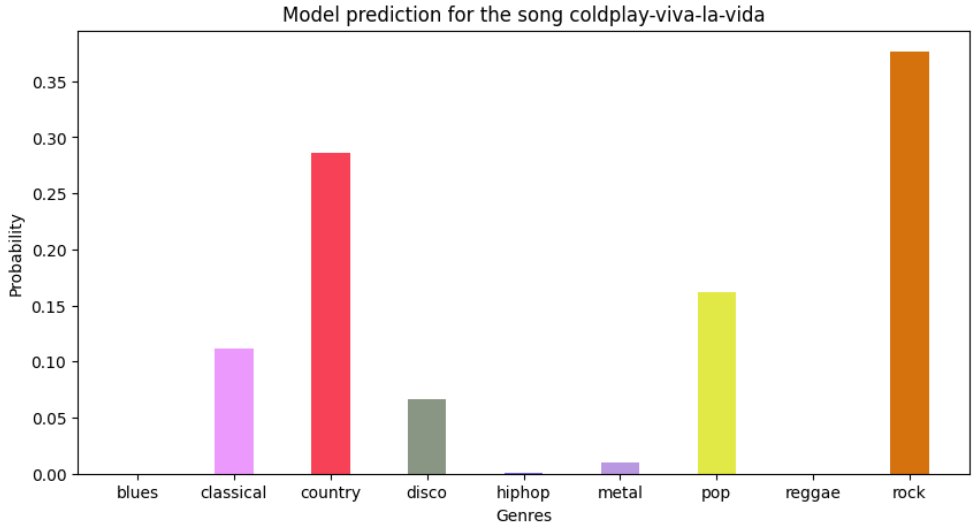

Some interesting results we can observe in this image, like ‘Rock’ being mixed up with ‘Country’, but ‘Country’ is not being mixed with ‘Rock’. ‘Classical’ is far and away the easiest genre to classify, as it is hard to get it mixed up with any other. ‘Metal’ is also confused with ‘Rock’, but ‘Rock’ 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:



As expected, no problems classifying it as ‘Classical’.

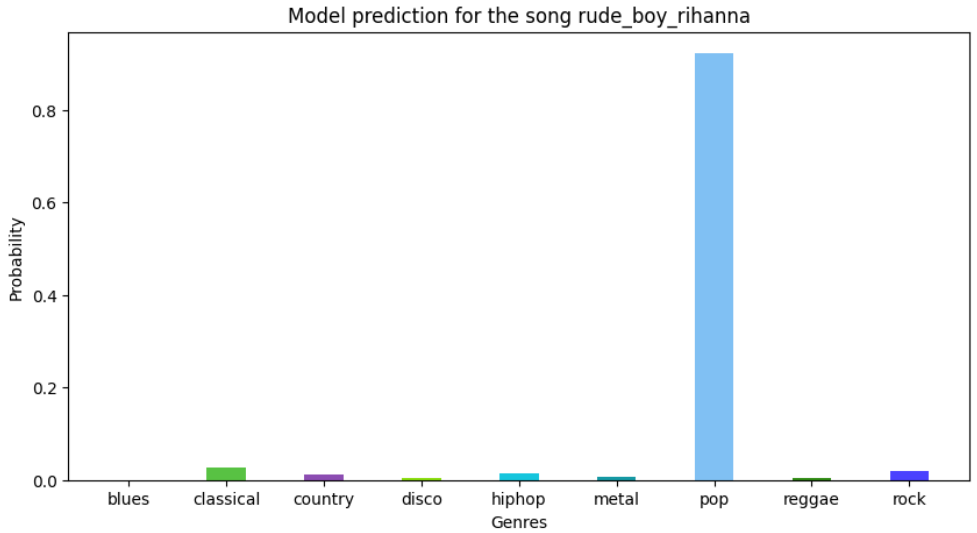
Another test, Coldplay’s ‘Viva la Vida’:



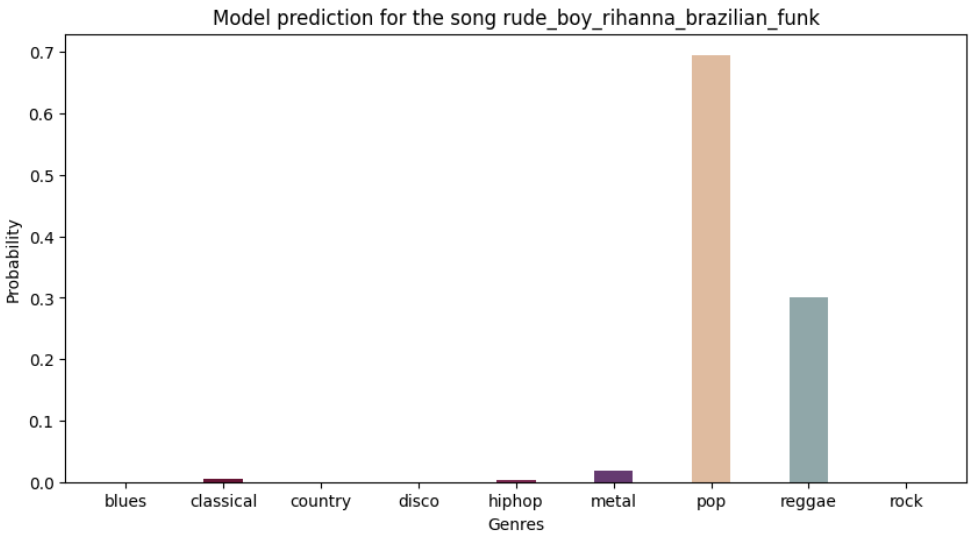

Viva la Vida is an interesting song, because although it is regarded as ‘Rock’ it has a lot of elements from other genres, and our model does a great job at identifying most of them. From ‘Pop’ to ‘Classical’, Viva la Vida is more a mix of genres and we can determine our model to be successful in navigating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




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




It still classifies it as ‘Pop’, but it has been able to detect that it is not 100% Pop any more, 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.

## Custom 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:

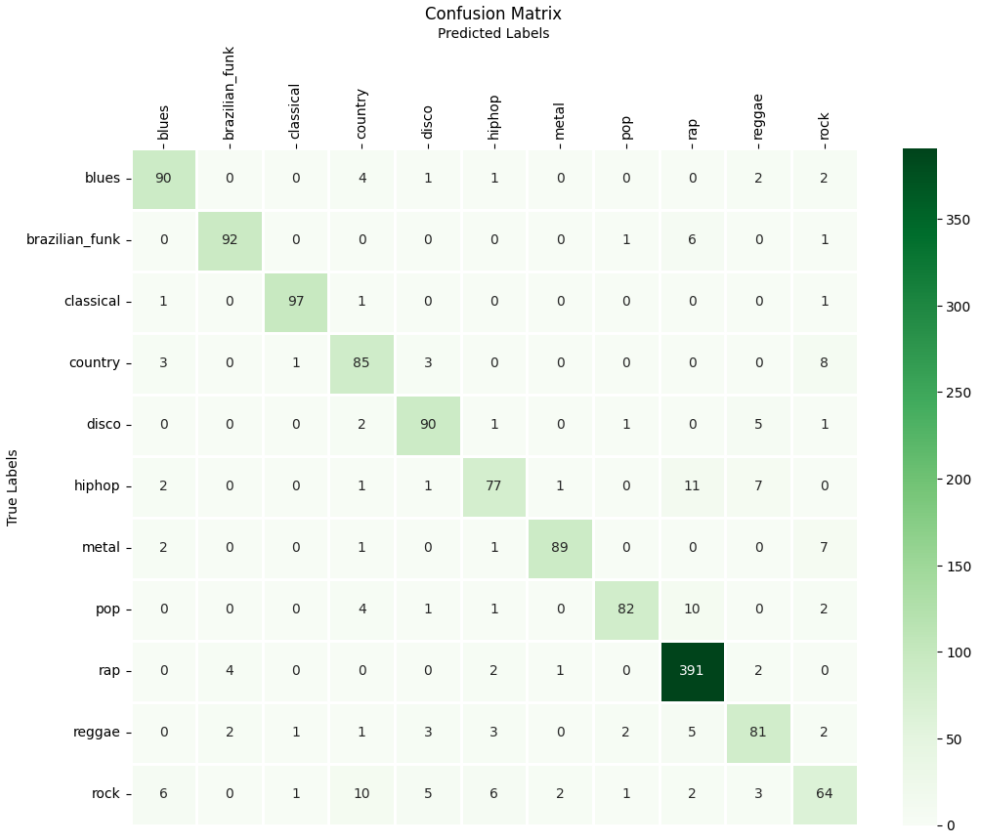
- **Training**: 99%
- **Testing**: 88%
- **Validation**: 87%

If we take a 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 behind, so more training data would do them good.

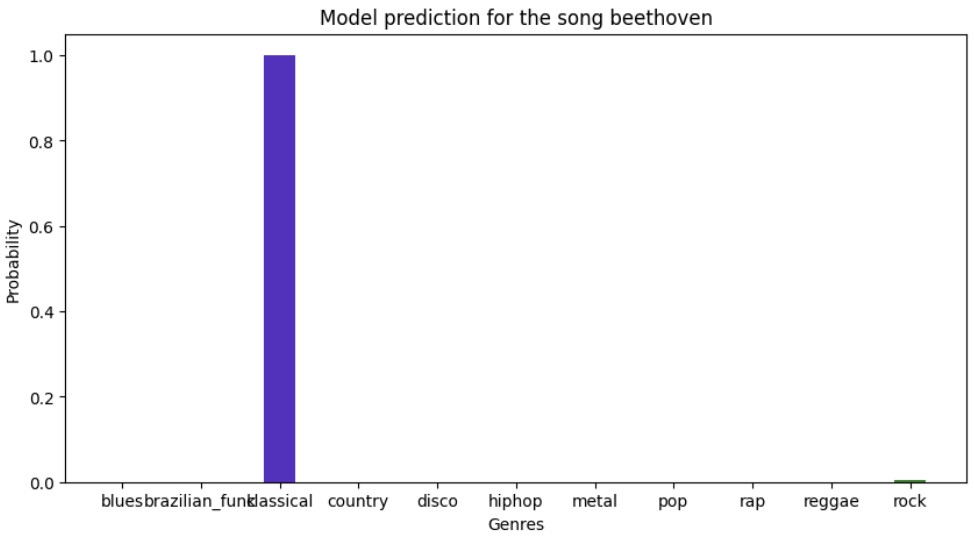
| | **Accuracy** |
|---|---|
| **Blues** | 87% |
| **Brazilian Funk** | 94% |
| **Classical** | 97% |
| **Country** | 78% |
| **Disco** | 87% |
| **Hip Hop** | 84% |

| | |
|---|---|
| **Metal** | 96% |
| **Pop** | 94% |
| **Rap** | 92% |
| **Reggae** | 81% |
| **Rock** | 73% |

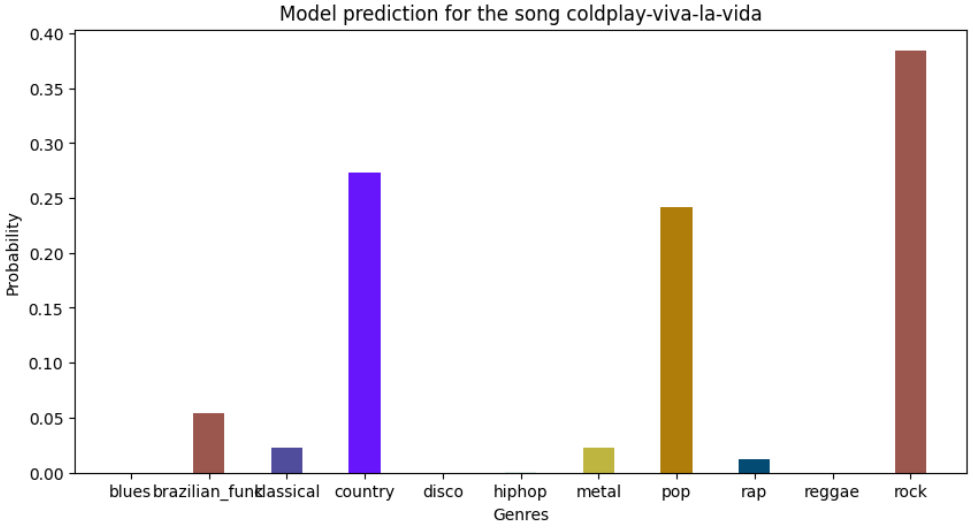
The confusion matrix:



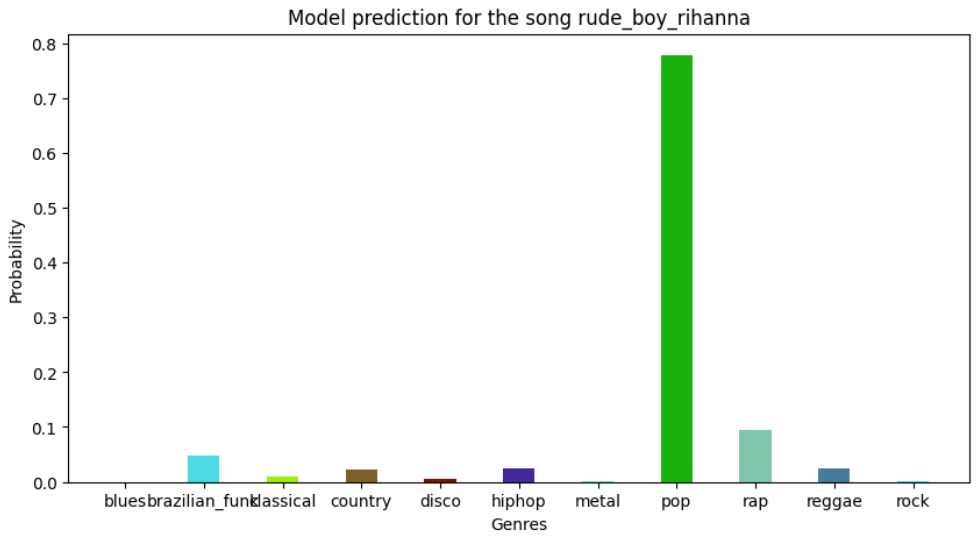

And now we perform the same test on popular songs as with the original dataset. First, Beethoven:

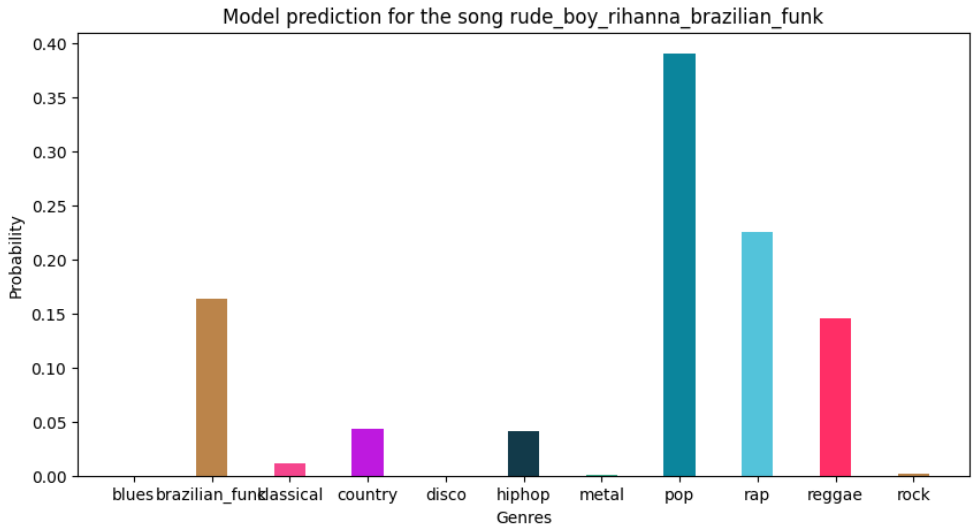



No surprise here. Next, Viva la Vida:




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




## 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.

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

- **Increase the size of the dataset**. Nowadays, we don't really need GTZAN anymore. It's really easy to access Spotify playlists, download them and feed them to the model to train and improve on. Right now, the original dataset has 9000 samples and the custom dataset has around 14000. It would be really simple to expand this number ten or even a hundred fold with Spotify playlists. Especially interesting would be to expand the dataset on those genres that don't yield great accuracy.

- **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 really, really good, it is also true that with even more testing we could extract a few more % points in terms of accuracy.

In conclusion, we deem this a successful project that has enriched our knowledge of CNNs and audio manipulation.