Artist classification through spectrogram learning - RecoNet (Recognize Net)

Julien Guinot
The University of Adelaide
julien.guinot@student.adelaide.edu.au

Abstract

Shazam is a music-identifying app which functions through deterministic fingerprinting of their immense database and cross-correlating real-time audio buffers to identify an ambient song.

Despite the power of the algorithm, and the wide use, the Shazam algorithm lacks robustness to real-life situations such as live performances or remixes. This prompts this project. This paper focuses on the proceedings and results obtained over the course of the project to replicate Shazam's functionality through deep learning and spectrograms.

Through Literature overview, we isolate relevant computer vision methods and models, as well as pre-processing and augmentation steps for audio data to be used in the project. Accuracy comparison for these models allows us to zero in on a given architecture, which presents both technical novelty, and a basis for relevant performance. Furthermore, this allows us to establish a baseline prediction for accuracy: 70%

Data sourcing and pre-processing is conducted to obtain discographies for various artists, segmented audio clips of isolated vocals are then fed to the aforementioned model. Through training, the model obtains 68% accuracy on never-before seen test data, despite technical difficulties

The proposed model learns coherent features, but not in-depth enough, which prompts thinking about next steps for the project, which could have performed much better with the necessary hindsight.

1. Introduction

This paper presents the entirety of proceedings in building a deep learning model for classifying vocalists from audio clips of their songs by extracting features from spectrogram representations of their voice.

Firstly, this paper will present inspiration and project goal considerations taking into account the problem to solve, current literature on the subject of audio classification. Related work and audio classification tasks will be presented, as well as initial project goals and those same goals reviewed taking into consideration resource limitations for the project.

A second part will address the dataset used for the project, which consists of audio clips of isolated artist vocals [1] , by identifying relevant data, sourcing it, and establishing a full pipeline of data cleaning, pre-processing and augmentation. All of the data will be presented in a raw state, cleaned and pre-processed state, as well as augmented state by relying on common audio and image augmentation techniques [2].

A third section will be dedicated to methodology, which includes a comprehensive review of current literature. Both convolutional [3], [4], [5] and sequential [6], [7] models will be covered, as well as a novel method combining both deep learning architectures [8], in an objective to identify which model will suit our task best. Data splitting into training, test and validation splits will also be covered, as well as performance metrics relevant to our task, which is a multi-class classification task. Furthermore, a brief review of current literature results will allow us to establish a baseline expectation for our model.

Two smaller sections will cover both code implementation and structure, and some custom particularities of the implemented code with regards to the long training times of the task, as well as the unusual nature of the data. Parameter choice will be determined, including pre-trained parameter weights for our model.

Finally, results will be discussed, both with regards to our original aspirations for the project, and with regards to current literature. Potential paths for increasing obtained accuracy will be discussed, and a plan for bettering of the

application will be established, with the objective in mind to continue the project beyond the course's scope.

2. Building the project

This section focuses on the main inspiration for the project, which is a music-recognition app called Shazam, and its inability to adapt to slight changes in music. This sparked curiosity regarding how to improve this flexibility, and the subsequent literature overview covers what was discovered in the domain of deep learning music recognition.

2.1. Inspiration - Shazam's lack of flexibility

Shazam is a music recognition app, released in 2002. The principle is quite simple: hold your phone up to the source of the music you want to identify, and, provided it is in Shazams' database, it will fetch the song name, artist, album, and Spotify link, and provide it to the user in under a second [9].

Figure 1: Logo for the Shazam app

With over 1 billion downloads (**Source : App store**), and database ties to the Spotify,Apple Music and Soundcloud databases, the app is very powerful and capable of recognizing most songs in a noisy setting, providing acceptable use case satisfaction for the average user.

Audio fingerprinting

The method used by Shazam to recognize songs is called Audio fingerprinting. The goal is to analyze the principal frequency components of the recorded audio for various bands on the frequency spectrum, and for various time intervals of the music, of the order of magnitude of the 10th of second [9]. This yields what the original paper calls a "Constellation graph", where principal frequency components for each timestamp of the song are recorded (Fig. 2):

By identifying each of these principal components, one can create a hash tag accounting for noise by introducing a fuzz factor. Below an example of a hash tag for a 0.1s interval of a given song:

```
hash = [23,40,57,131,252]
```

So, Shazam analyzes entire songs and stores all the hash tags for all sampling intervals into a database. This is shown in the Following Figure (Fig.3), taken from [10]:

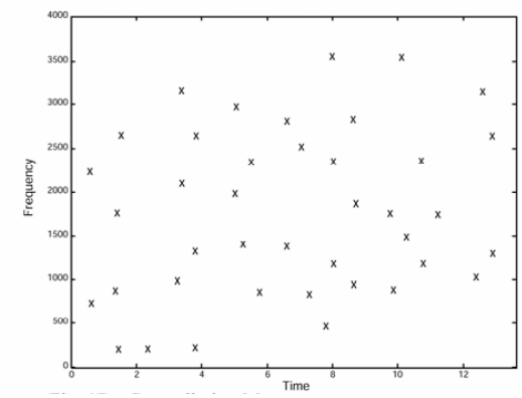

Figure 2: Constellation map taken from [9]

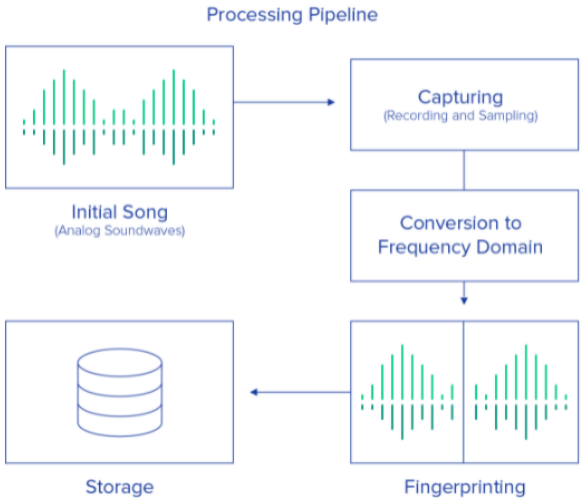

Figure 3: Shazam fingerpringing process

When a user records a song playing in the background, hash tags are generated for each interval, which allows to progressively narrow down to a handful of songs accounting for fuzz factor, which generally takes under a second [9]. Computation of distance between the recorded hash tags and the narrowed-down list of songs is then conducted, and the closest song is identified as corresponding to the song with the relevant hash tag in the database. This allows for minimal error in most cases, as well as being resistant to noise and distortion by cross-correlation of audio samples.

Inflexibility of the fingerprinting process

This process is robust in everyday scenarios. However, consider the use case where one is at a live performance, or a remix is being passed, or the song is being played at a DJ set to tempo-match another song, so is slightly sped up, filtered, or has an effect applied to it. This means that the

hash tags are not the same, and thus the correct song cannot be identified.

This means that the Shazam method is quite inflexible, as it cannot account for many real-life situations. This sparks curiosity, and a first intuition is that we should be able to train a Deep Learning model to act as a knowledgeable acquaintance which would be able to identify if not the exact song, then the album or the artist.

2.2. Related work - Explored tasks

To confirm that this is possible, relevant literature is explored, and previously worked-on tasks are identified (the methods and results obtained for these tasks will be discussed in a later section.

Artist recognition Artist recognition not restrained to vocals is explored in [7], [11], using the Artist20 dataset which is a compilation of songs from 20 artists, by extracting spectrograms from fixed-length audio clips and applying computer vision models to them. Zhang et. Al, on the other hand, use WaveNet, a sequential model. These differ from our approach in that they do not isolate the vocal to work on, and rather classify based on the artists' style. [12] Approaches this task as well, but using the million song dataset, which contains pre-extracted features for audio classification, and thus does not interest us as we wish to use raw audio signal data.

Genre recognition Another task quite relevant to our aspirations is genre classification. This approaches Vocalist classification in that vocal features are often a determining characteristic of a given genre. Using the GTZAN dataset, which contains music clips for various genres, [4] and [13], as well as [14], tackle this task by using convolutional neural networks applied to spectrograms.

Vocalist classification This task is of great interest to us, as it pertains directly to the task at hand. [15] Tackles the problem directly for musical vocals by isolating the vocal from the music and extracting time-series features (Mel Frequency Cepstrum Coefficients) to be feed to a neural network. [3] Uses spectrograms as inputs to a convolutional neural network to identify speakers.

Bird Species classification Almost a classic task in MIR (Music information retrieval), The rainforest dataset is a classic dataset containing various bird species's songs. The goal is to implement a model able to identify the bird species present in a given recording. [16], as well as [8] and [2] tackle this problem by using spectrograms of the birdsong samples as inputs for a CNN.

2.3. Project goals

Now that we have identified which tasks have previously been explored in the literature, we are able to identify our own custom task which we want to tackle with this project. This section addresses this by clearly identifying the task at hand, which will serve in sourcing our data

2.3.1 Initial project goals and aspirations

Given the apparent complexity of identifying precise songs through deep learning, we choose to fall back on identifying the artist, as genre identification is rarely an issue in the use case of Shazam (The user already knows what genre they are listening to). Album classification might be possible, as seen in [11], but seems hard with the limited data available, not without building our own dataset. Ideally, our model would be part of an application which listens and converts in real time to spectrogram data (which seems to be the most common and fruitful method in MIR for classification).

Finally, we want our model to be able to identify as many artists as possible in a time frame comfortable for the user, with reasonable accuracy and with minimal misclassifications.

2.3.2 Compromising with regards to available resources

These goals are quite ambitious for the given time frame (4 months), with the given dataset (Custom made, cleaned, pre-processed), with insufficient material resources to train various models and proceed by trial and error. We constrain our model specifications and task to the following:

- **Artist recognition**
- **Using spectrogram Image classification**
- **Being fed 10s clips**
- **Reasonable accuracy, low misclassifications**

According to [11], 10s seems to be the most appropriate time frame to be feeding our neural network, as it achieves best accuracy with this duration on their custom model. The low misclassifications criterion stems from the use case, and the appropriate accuracy metric will be determined in a later section by reviewing results obtained in literature.

3. Data

Now that we have clearly identified the task at hand, relevant data needs will be chosen, and the corresponding dataset will be sourced and processed. This section covers the implementation of this selection.

3.1. Building the dataset

3.1.1 Identifying relevant data

The goal is to identify vocalists from raw audio data of their isolated vocals. We isolate vocals to recognize vocalists based on their vocal characteristics and not upon their musical style, which can be a byproduct of producers or temporary trends. The main data need for this project is thus **fixed-length Audio recordings of vocalist voices**.

So we need to source Audio recordings of artists songs. To source relevant data, a couple more criteria come into play :

- **Artist popularity: We want our dataset, which can obviously not be made up hundreds of artists, to contain data from popular artists**
- **The data must be diverse in terms of artist gender, time period, and musical genre, to not build a model uniquely tailored to rap for instance.**

Keeping these criterion in mind, the following figure (Fig. 4) shows what we focus on when selecting our data.

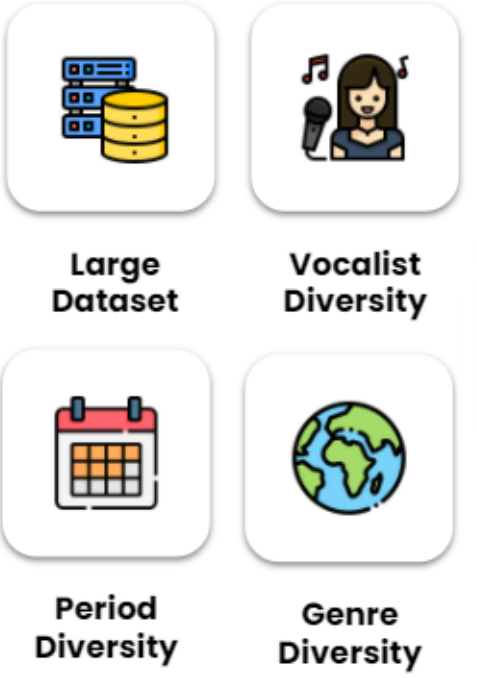

Figure 4: Data selection criterion

3.1.2 Sourcing data

As said before, we want vocalist recordings from popular artists, and from all decades and genres, or at least the most popular ones. Thus, we search for most popular artists by decade, and download the entire discography of each artist **From my personal Spotify local files**. This yields a total of **23 artists, 9 subgenres, 4 overgenres, 261 albums, 3145 songs, 11.7Gb of music, and 199 hours of music.** The following table shows the amount of songs per artist.

artist	Number of songs
Ariana Grande	112
BTS	91
Beyonce	61
Billie Eilish	77
Bruno Mars	39
Coldplay	95
David Bowie	416
Drake	200
Dua Lipa	28
Earth Wind and fire	229
Ed Sheeran	63
Eminem	268
Juice Wrld	60
Justin Bieber	190
Kayne West	126
Kendrick Lamar	59
Khalid	32
Michael jackson	139
Post Malone	63
Queen	315
Taylor Swift	123
The Beatles	225
The Weeknd	56

Table 1: Number of songs per artist for the unprocessed dataset

These figures will be expanded upon and re-presented in the following subsection.

3.1.3 Unprocessed dataset

The previous steps of ownloading and sourcing personal audio file data yields an unprocessed dataset which can be explored using Exploratory Data Analysis to verify it checks all of our data needs for the project.

Genre Distribution by year

One objective was to have a healthy distribution of music time periods to not only analyze contemporary artist vocals. This is due to the evolution in recording and mastering technology, which over the course of the years has led to heavy changes in vocalist performances. We want our model to maintain integrity even when identifying old vocalists, and not learn only modern mastering techniques. The following

figure (Fig. 5) shows the distribution of subgenres by year as well as global song releases:

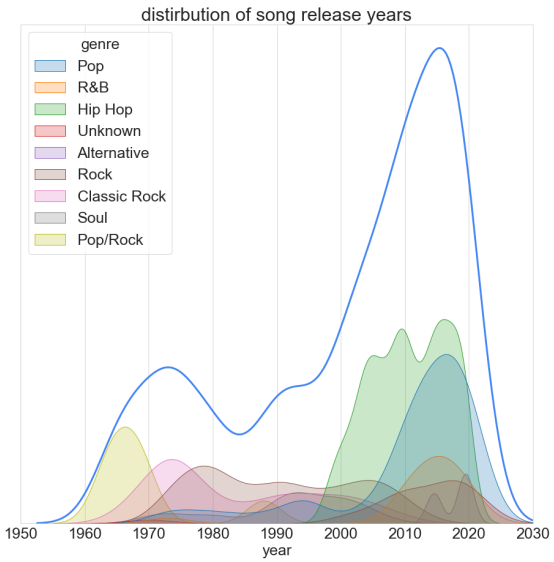

Figure 5: Subgenre distribution by year

Though inevitably more modern artists are selected in our dataset as shown by the distribution spike starting in the 2010s, which will be coherent for the use case, the performance of the model should not be impacted. The spike is due to pop music and hip-hop music and their late rise. So, the model will learn modern mixing techniques for vocals of that period, but for other, earlier genres, it will learn the appropriate features, when pop and hip-hop did not exist.

Artist Diversity Another objective was to have a good amount of classes (artists) to base our classification on, with a good amount of data for each of those. The data for amount of songs per artist can be found in Table 1, and in Figure 6 below.

Though there seems to be some class imbalance, we will see later on that this is to serve the purpose of balancing genre distribution. Indeed, many of the overpopulated classes (*The beatles,david bowie, Queen*) are older artists and will serve to reinforce learning of older characteristics rather than newer ones. Essentially, newer genres win out by amount of artists, but older genres win out by amount of songs, as their discography has had more time to grow.

Genre distribution Our last criterion for our dataset is genre distribution and diversity. Indeed, it is important to have a balanced musical genre distribution, as to not bias the classifier towards a given musical genre (rap is spoken, for instance, while pop has more legato singing, and rock has more distortion in voices.) The genres of each artist are

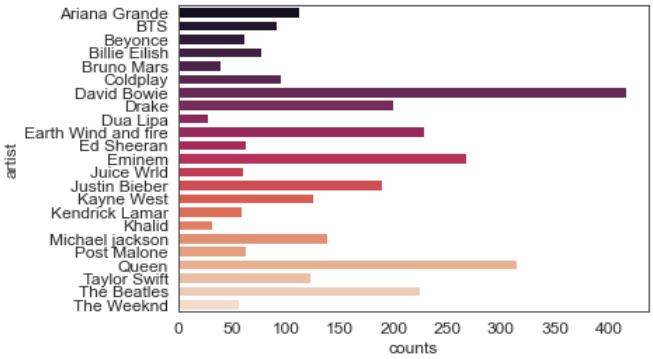

Figure 6: Amount of songs per artist

identified by extracting mp3 file metadata, and are classified into overgenres :

- **Rock** contains Alternative, classic rock, and rock-n-roll
- **Hip-hop** contains rap and hip-hop
- **DiscoSoul** contains RnB, Disco, Soul, and Funk
- **Pop** Contains Pop and pop-rock

The distributions for these overgenres and subgenres is shown below (Fig. 7), and shows a healthy 1/4 split for all overgenres, which is desirable for a non-biased classifier.

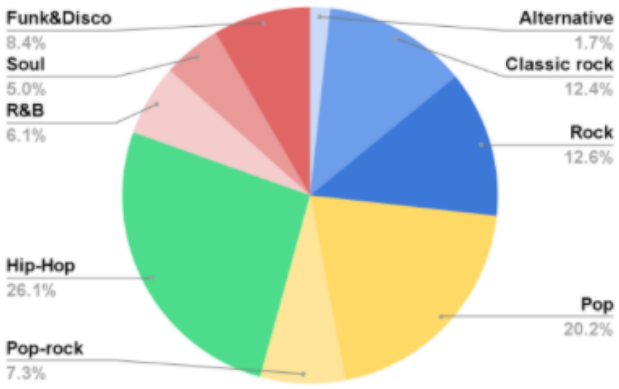

Figure 7: Subgenre and genre distribution for sourced dataset

Considering these three criteria, the data we sourced is both relevant to the task at hand and should not impede our ability to obtain results. With this in mind, we move forward towards cleaning and pre-processing the data

3.2. Data cleaning and pre-processing

This section focuses on cleaning and pre-processing our raw data in the best fashion to apply our model to. An explanation on the need to clean the data and to pre-process it will be undertaken, as well as a presentation of the full pre-processing chain and the code structure necessary to getting our full dataset

3.2.1 Cleaning

Cleaning the data is important as it gets rid of potential anomalous data points which do not fit our model's task and might bias our model in the wrong direction. The following audio clips are to be removed from the dataset:

- **Audio clips 30s or less in duration**. Though these clips are rare, they exist, and are often interludes where either the artist does not sing, or only speaks, and so in which there is no relevant information for our task
- **Instrumental-only audio clips** Having no vocals, these contain no relevant features to learn for the task at hand, and can be purged from the dataset
- **Songs featuring multiple vocalists**: Though the possibility of creating a multi-class multi-label classification task (as in [2]) exists, we choose not to complexify an already complex task, and consider multilabel classification outside the scope of this project. Thus these clips will only be parasitic to the task at hand.

This cleaning is hard to set up algorithmically (except for duration filtering, and so it is conducted manually before moving on to the pre-processing phase. After cleaning, the number of songs is reduced to 3097 songs, which represents a **9%** drop in dataset size, showing the necessity of the cleaning step.

3.2.2 Pre-processing

At this point, we have a dataset of variable-length mono-label audio clips which are to be transformed to spectrograms. Before anything, some pre-processing must be conducted to these clips for the following reasons:

- **Vocal isolation**: as we are working on vocals alone, a necessary step is to isolate the vocals for all songs
- **Splitting**: as all songs are variable length and computer vision algorithms take fixed-size images as inputs, we must split sampled into 10s-long clips and pad the remainder with silence to attain fixed-length input
- **Silence removal**: As the vocalist does not sing for the entirety of the song, after splitting some clips will be made up of silence. We evaluate their Decibel level using the pydub package to eliminate those reaching less that $-16dB$

At the end of this process, we have n_{song} audio clips for each $song$. The process is shown in the following Figures For Ariana Grande's hit 'positions'. Fig. 8 Shows the raw audio data, unprocessed:

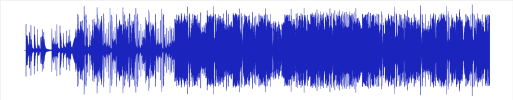

Figure 8: raw waveform for "Positions"

At this point, we have to separate the vocals from the instrumentals. Many models exist to do this, and we choose to serialize Deezer's **Spleeter** [1] model to extract the vocals. Spleeter uses tensorflow and keras as a backend, and processes a 3 minut song in approximately 5 seconds. The following figure (Fig. 9) shows the dark blue vocals waveform nested in the light blue raw waveform, which is extracted using Spleeter:

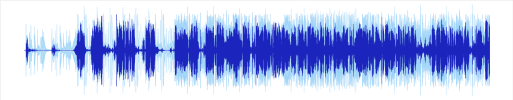

Figure 9: extracted vocals waveform for "Positions"

The next step is splitting the data around perceived silence. pydub is used to determine where silence is in the vocals waveform and that same package is used to split the remaining auio into 10s clips and padding the remainder to attain fixed-length input. Fig. 10 shows perceived silence (red) and 10s splits(green) for "Positions":

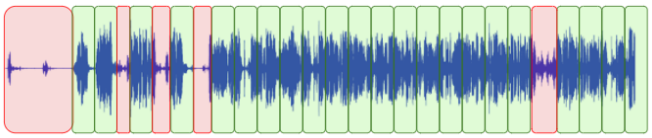

Figure 10: silence-vocals split for "Positions"

Finally, after eliminating the non-vocal parts of the song, we are left with the following split, which yields clips of length 10s, which are saved to the hard drive and stored into a dataframe for easy referencing. Fig. 11 shows the selected clips for Positions:

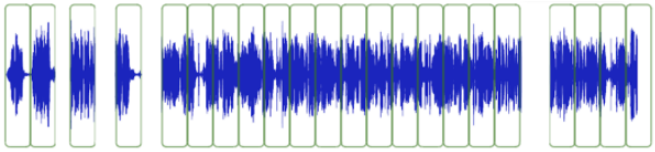

Figure 11: Final clips used as training data for "Positions"

The next step is logically to generate spectrograms for the selected clips. However, as we wish to apply audio data augmentation to the clips as a preemptive solution to overfitting in model training, We choose not to generate the spectrograms right away, but rather to keep the dataset as an audio dataset. Though this yields some loading time problems during training, [8] and [2] Show that data augmentation for such tasks is essential, and so spectrogram generation is kept as part of the pytorch custom dataset, which will be presented in a later section, though the generation step will be presented in Subsection 3.2.4.

The following section focuses on data augmentation techniques for spectrogram learning, which will show why we cannot pre-generate spectrograms.

3.2.3 Data Augmentation

Data augmentation serves as a regularization method to avoid overfitting in deep learning complex models. The principle is to apply a transformation or series of transformations to a training sample in order to modify the data and obtain various samples from the same training sample. This allows models to be trained on slight variations of training samples and thus avoids overfitting to the exact training set.

For our task, the exact pre-processing chain will be explained in subsection 3.2.4. However, as we are converting audio clips to spectrograms and inputting the spectrograms into a computer vision model, it is interesting to apply both audio augmentation (before spectrogram generation) and image data augmentation (after spectrogram generation). This is why it was not optimal to generate spectrograms before the data loader.

This section focuses on data augmentation techniques for both types of data.

Audio data augmentation

Our Audio data augmentation chain consists of 5 audio effects that are applied with varying probability, the chain itself being applied with a given probability. This allows us to generate vastly varying training data for our model. To compare data augmentation techniques, the following figure (Fig. 12) shows the spectrogram of a 10s clip from "Positions", which we will apply all audio augmentation techniques to successively, without image audmentation techniques:

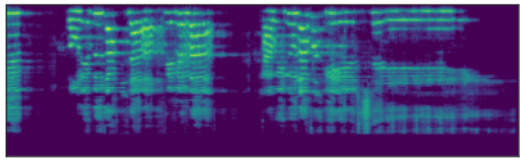

Figure 12: Original 10s clip from positions

- **Gaussian Noise, p = 0.5** : Gaussian white noise is applied to the audio signal, with a given signal-to-noise ratio as a parameter. Fig. 13 shows the resulting signal:

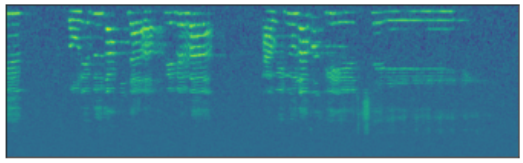

Figure 13: Gaussian Noise applied to the 10s clip

- **Pink Noise, p = 0.5** : Pink noise (the power level of which decreases logarithmically along the frequency spectrum), is applied also with a given signal to noise ratio (Fig. 14)

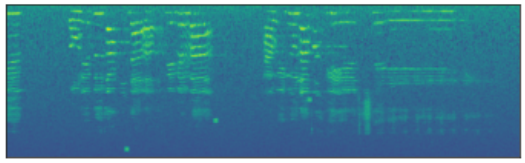

Figure 14: Pink noise applied to the 10s clip

- **Timeshift, p = 0.3** a random circular timeshift is applied to the signal, depending on the sampling frequency (Fig. 15)

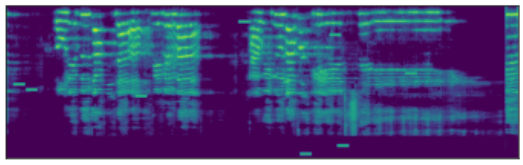

Figure 15: TimeShift Applied to the 10s clip

- **Volume Control, p =0.1**: A sine wave volume modulation is applied to the signal depending on the sampling frequency, at a random amplitude (Fig. 16)

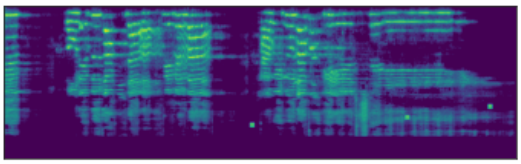

Figure 16: Volume control Applied to the 10s clip

- **Click track, p = 0.25** : Based on perceived tempo for the track, a click track containing clicks is added onto the track, which aids the model in recognizing genre tempos (Fig. 17)

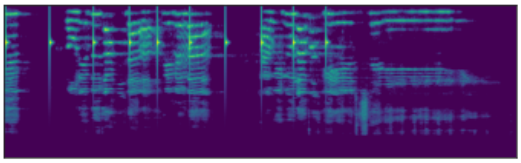

Figure 17: Click track Applied to the 10s clip

Image data augmentation Once the spectrogram is generated from the augmented audio, another potential step is to add image data augmentation on top to further decrease the risk of overfitting. Same as previously, the techniques implemented in the chain are described below, and their application on the same clip as previously without audio augmentation is shown:

- **Random brightness, p = 0.4:** The brightness of the image is randomly decreased to within a given threshold, which correlates with a global decrease in volume for the clip (Fig. 18):

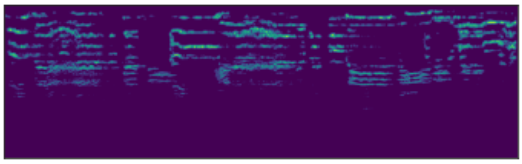

Figure 18: Random brightness applied to the generated spectrogram

- **Cutout:** Cutout consists of setting all pixels in a rectangular region of the image to zero, effectively cutting out a portion of the spectrogram (Fig. 19).
- **Coarse Dropout:** Similar to cutout, coarse dropout randomly cuts multiple areas of the image out, effectively creating holes in the image (Fig. 20)

  Though many other techniques exist, these are the ones that were used in this project, inspired by [8] and [2].

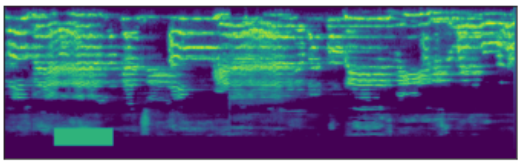

Figure 19: Cutout applied to the generated spectrogram

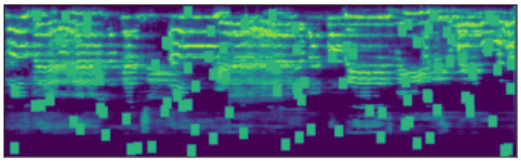

Figure 20: Coarse Dropout applied to the generated spectrogram

Adding anything that blurs or adds more noise or jitter to the image would be quite overkill with the noise audio augmentations implemented before, and flipping or geometrically altering the image would be counterproductive, as the model would never have to identify backwards singing.

3.2.4 Pre-processing chain

So, having established a pre-processing chain all the way from original unsplit audio to Spectrograms of 10s clips of augmented data, we are now able to Proceduralize all those steps. The code for the whole pre-processing chain can be found in the src/Preprocessing-pipeline folder in the following github repo:

`https://github.com/Pliploop/Spectrogram_Artist_Recognition`

One code structure aspect that might be of interest is the parallel generation of dataframes containing relevant data during the pre-processing step, to facilitate visualizations and graph-making:

- **When splitting the songs between vocal and instrumental, a dataframe on non-split songs with unique uuids is generated for tracking throughout the process**
- **When segmenting the songs into 10s clips, a new dataframe containing unique uuids for all segments.**

The global pre-processing chain is shown in the following figure. In yellow, the pre-processing chain and in grey the augmentation chain are shown. (Fig. 20)

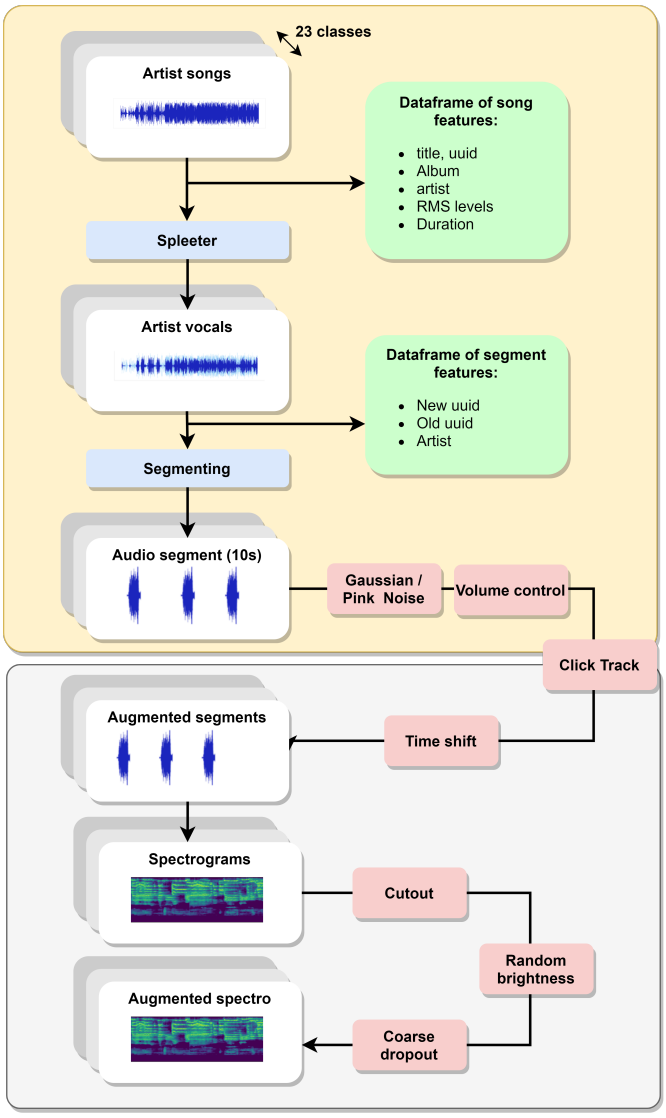

Figure 21: Full pre-processing chain

The final dataset is made up of **60987** augmented audio clips of isolated vocals ready to be fed into the selected model, which will be selected in the next section.

4. Methodology

This section focuses on the methodology approaches chosen to undertake this project. an overview of current literature results obtained with various models will be conducted to choose the model to use. Then, an overview of training considerations such as data splitting, k-fold validation, and performance metrics will be undertaken, as well as a quick estimation of what can be expected for the performance of our model

4.1. Model Choice

As seen in section 2, many tasks have already been undertaken that are similar but not quite the same as the one at hand. This subsection overviews results obtained with which methods for those tasks, and leads to an educated choice concerning the model we will be using.

4.1.1 Related work and results

Though the task of audio classification is an explored problem, the exact task at hand not been explored quite yet in literature. However, many papers hold valuable insights on the dos and don'ts of audio classification, and present various methods towards achieving desirable accuracy. This section focuses on conducting an overview of the existing classification methods as well as selecting a model and choosing training parameters.

4.1.2 Literature overview

Many audio classification tasks have been attempted from various angles and from various models, as discussed in **Section 2.2**. We address the various techniques used for relevant tasks here as well as their success, the metrics of which will be summarized at the end of this section.

Genre Classification

[14] Uses Multilayer Perceptrons with varying amounts of hidden layers to achieve **85%** accuracy on the GTZAN dataset, with less convincing results on other genre classification datasets (**75% and 45%**).

[12] implements Convolutional MLPs on the Million song dataset which contains pre-extracted audio features, and achieves **29.5%** accuracy after training.

[13] Implements many models, including VGG19, on the GTZAN dataset towards genre classification, and achieves at best **60%** on genre recognition, while [4] achieves **91.3%, 91.0% and 90%** accuracy rrespectively using pretrained **DenseNet,ResNet and Inception** CNNs

Bird species recognition

[16] Uses a custom CNN architecture closely resembling that of VGG16 for bird species classification, attaining at best **69%** precision when classifying unique species with no other background noise.

One of the top submissions on kaggle for the Rainforest connect competition [8], achieves **88.3% validation accuracy** by using a novel architecture [8]: a ResNet pretrained

network which passes its intermediary activation maps as features into a WaveNet [6] structure as input features. This is of great interest to us, as we will be using this architecture for our model.

Artist classification and speaker recognition

[11] Uses a custom network architecture (CNN with gated recurrent units) with 3s segments as input on the *Artist20* dataset to achieve **96%** F1 score on song recognition tasks.

[12] implements a basic CNN followed by MLP to achieve **35.6%** accuracy on artist recognition tasks, while [15] gets a mean F1 score of **92.6%** by using MFCC (Mel feature Cepstrogram Coefficients, which are different from Mel Spectrograms) on isolated vocals of vietnamese pop singers. [7] uses WaveNet as a classifier to achieve **95.7%** accuracy on *Artist20 dataset*, which further convinces us of the potential of the WaveNet algorithm. Finally, [3] uses spectrograms and A combination of gated residual networks designed for multitasking, the complexity of which is outside the scope of this project, to achieve **88.5%** accuracy on speaker (not singer) recognition.

4.1.3 Sequential versus Convolutional

Two model architectures stand out when reading previous work for these tasks: **Sequential models** and **Convolutional models**. In this section, we overview two convolutional models and a sequential model with the objective to choose between the two, and settle on a combination architecture inspired by [8]

DenseNet and ResNe(s)t

Considering our dataset is comprised of spectrograms, we will remain focused on methods used on this type of data to draw inspiration for our model. Seeing the results of **DenseNet** and **ResNet** in [4], we will be using transfer learning on a One of the models provided by the torch backend as a baseline for our model.

The DenseNet model, proposed in 2017 in [17], Is a CNN architecture than has the particularity of linking all convolution layers between each other, which **Alleviates vanishing gradient issues, greatly reduces the number of parameters to learn, and allows for very deep neural networks, meaning more subtle features can be learnt.** The general structure for DenseNet is shown below (Fig. 21):

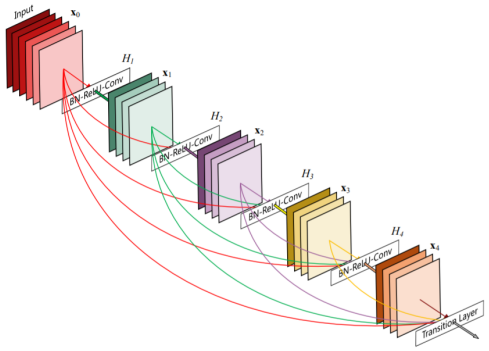

Figure 22: DenseNet general structure

ResNet was implemented for the first time in [18] And introduced Residual blocks, which are a previous version of the all-connected structure in denseNet. By feed-forwarding inputs from some convolutional blocks to the following, vanishing and exploding gradients are alleviated. This allows creation of very deep convolutional neural networks for the first time, and thus capturing some subtle features that could not be captured before. The general structure for ResNet is shown below (Fig. 22):

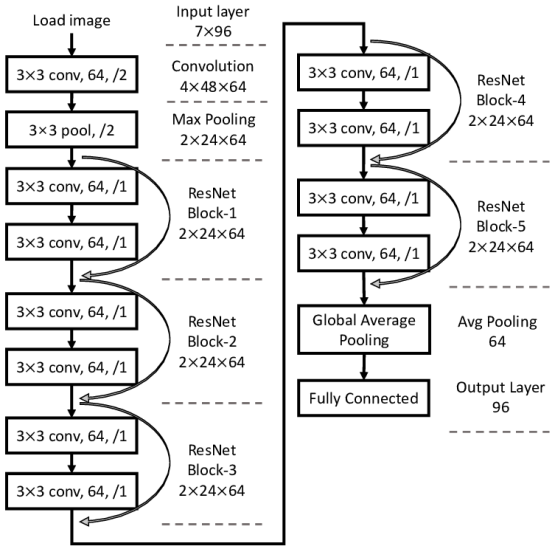

Figure 23: ResNet general structure

A variaion on ResNet which could be interesting for our usecase is ResNest (Residual Split-attention blocks Network). On the one hand because it was implemented in [2], and we have available pre-trained weights to use (Trained on the rainforest bird song dataset). On the other hand, Resnest implements a split-attention mechanism by stacking split-attention blocks. The cardinal group representations are then concatenated, and, similarly to resnet, the original input is added to the concatenated conardinal group

vector [19]. This Resnest basic block is shown Fig. 24:

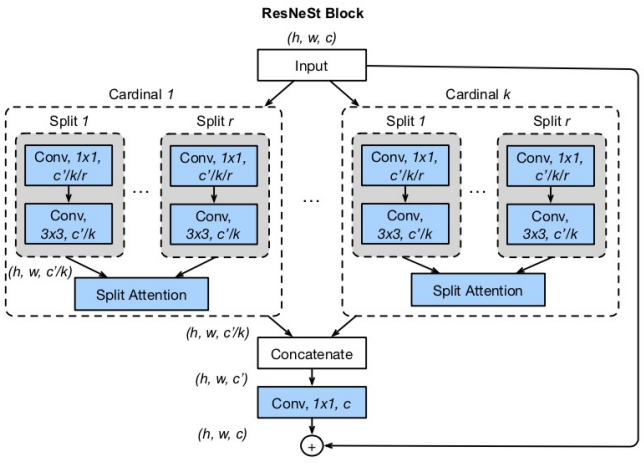

Figure 24: Resnest block structure

This is the convolutional structure of most interest to us for reasons detailed above.

WaveNet

Though this architecture might seem promising, more accurate results have been observed in [7] with WaveNet classification, and [8] uses WaveNet as a supporting algorithm for ResNet, achieving good results on the Rainforest Connect dataset. The WaveNet architecture used for classification in [7] is shown in the following figure (Fig. 25), and has the advantage of being able to work on raw signal data.

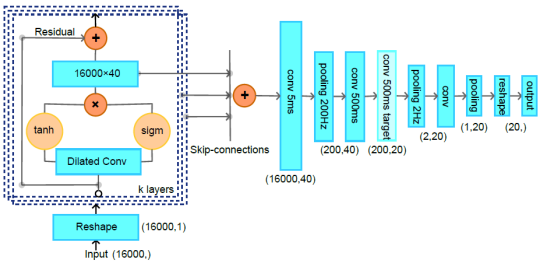

Figure 25: WaveNet architecture for classification

Combining Models

Our technical aim and novelty is going to be to combine WaveNet and DenseNet in one of two ways:

- **By using the activation maps from DenseNet as an input for the waveNet architecture, as in [2]**
- **By training WaveNet and DenseNet in parallel and using the concatenation of their outputs as input for our classification block.**

Both of these architectures are shown Fig. 26.

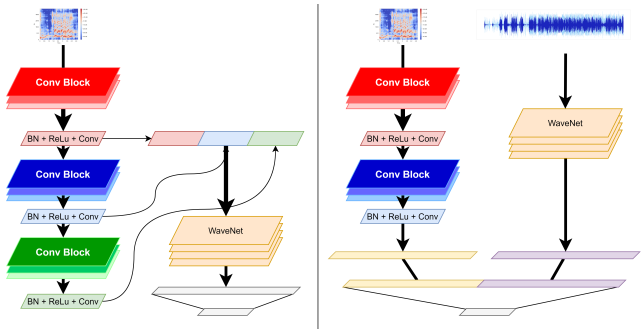

Figure 26: Proposed architecture for the project

This dual architecture is affectionately dubbed **RecoNet** (RecognizeNet)

Technical difficulties :

However, due to technical difficulties which will be discussed in depth in section 6.3.3, The model was greatly simplified. A resnest was still used, but we do not use WaveNet in this study, as the main hardware for training the model died in the middle of training at 5h/epoch, and with one week left to go and the final model parameters not yet chosen, we had to speed up training, and thus simplify the model, to finish in time.

4.2. Data splitting

The data was split using a stratified split, according to a **60-20-20** training-validation-test split. this means the following table holds true for the population of each class given a base dataset of size **60987**:

Training set	36592
Validation set	12197
Test set	12198

Table 2: Train-validation-test split for training

The original plan was to use K-fold validation for our model training. The idea behind k-fold validation is to train k models on k folds of the training split, by changing the train-test-validation split after training each model, according to fig. 27:

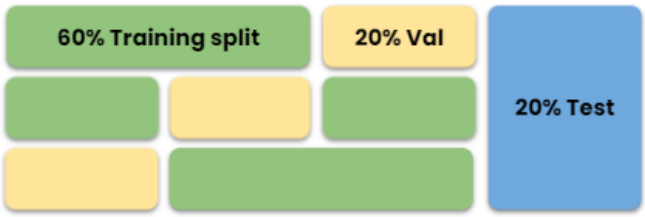

Figure 27: K-fold validation split for 3 folds

Having then trained K models, the optimal model can be selected by evaluating each one on the test set which remains the same. **Due to the same technical difficulties as discussed previously, this was impossible to implement in the allotted time. However, it is still worth mentioning here, as it is a solid next step towards making this model better**

4.3. Performance Metrics

4.3.1 Multiclass classification metrics

As stated previously, this problem is a multi-class classification problem, and because of the data cleaning done in section one, each training sample only belongs to one class (featuring tracks were removed). We choose the following metrics over the dataset to evaluate the performance of our model. Recall the definition of the following metrics for a binary classification problem, where TP, FP, TN and FN are respectively the amount of true positive predictions, false positive, True negatives and False negatives. Precision is notated p, recall r, accuracy a and the F1 score F_1

$$acc = \frac{TN + TP}{TN + TP + FN + FP},\ p = \frac{TP}{FP + TP}$$

$$r = \frac{TP}{TP + FN},\ F_1 = \frac{2TP}{2TP + FP + FN}$$

4.3.2 Accuracy

Global accuracy is of course an essential metric for this type of problem. Essentially, if we notate CP the amount of correct predictions and n the amount of samples on the test set, we have:

$$acc = \frac{CP}{n}$$

Which is a good first indicator of the performance of the model, given that the classes for our classification problem are relatively balanced.

4.3.3 Macro-averaged precision, recall and F1

Other metrics to consider are macro and micro-averaged metrics for the classes. Before getting into the actual formulae for these metrics, it is important to consider what they measure.

Micro-Average metrics measure metrics by taking into account the weight of each class in the dataset. Which means that a model which classifies more often into the most frequent class will be well-scored.

Macro-average metrics measure metrics by considering all classes to be equal in importance. This is what interests this project, as our goal is not to overclassify an artist, but rather to strike a balance between correctly and incorrectly identifying all artists at about the same rate. So, we will use macro-average metrics.

The macro-average metrics are simply the harmonic mean of each metric over all classes scored in a one vs All fashion (binary classification with the considered class being the positive and all others the negative. So, for m classes:

$$r_{macro} = \frac{\sum_0^m r_i}{m},\ F_{1macro} = \frac{\sum_0^m F1_i}{m}, p_{macro} = \frac{\sum_0^m p_i}{m}$$

4.3.4 All vs one metrics

We'll be wanting a more precise look into which classes are being correctly classified rather than not. So, with the same One VS all considerations as previously, all metrics (a, p, r, F_1), can be computed for each individual class to get a better overview of how the model classifies the song snippets. Another metric is ROC curves, but as those are time-consuming to plot for one class and we have 23 classes, we choose to not use them for this project.

4.3.5 Confusion matrix

Finally, A confusion matrix will be used as a way to identify which classes a given misclassified class is being separated into. This gives good insight on what additional steps could be taken to separate the data more (pitch correcting, frequency masking, etc) if The Weeknd is often misclassified as Drake, for instance.

4.4. Performance expectations

The metrics which are going to be of most interest to us in this study are overall accuracy and recall, as well as per-class recall. Indeed, what interest us is the percentage of correct predictions when presented with a positive data point for each artist: basically, which percentage of the time does the model miss out on any given artist. This is from our use case.

From the results obtained by current literature, we aim for a **baseline accuracy goal of 70%**. As many studies reach 80% accuracy, we believe this can be improved upon. However, we choose to set a lower standard due to low resources and exploratory nature of the project.

5. Implementation

This section focuses on code implementation of the previously-described methodology and model, with a special focus on two main elements which were custom-made for the project: a custom dataset and a custom trainer.

5.1. Code

The code for all the implementations described in this paper can be found at:

`https://github.com/Pliploop/Spectrogram_Artist_Recognition`

5.2. parameter Choice

5.2.1 Pretrained parameters

One of the reasons we chose to implement ResNest despite its high model complexity is the availability of pre-trained weights for the model. Cornell university conducted a study on birdsong classification, the weights of which were then used in [2] To narrow in on their own birdsong dataset.

Though birdsong and human vocals are not quite the same, the pretrained weights we use were trained on spectrograms, and rather than Use pytorch's pretrained models, which were trained on image classification datasets such as CIFAR and ImageNet, these weights start us much closer to our goal.

So, we implement a model parameter loading step in our custom trainer when loading the model, which allows us to start at 40% accuracy on the validation set instead of 3% with CIFAR-pretrained models.

5.2.2 Hyperparameters

Finally, Hyperparameters are selected for training the algorithm. A good starting point for training length is 100 epochs, though we will see that given the time span of each epoch, this was not possible.

The original intention was to conduct a **grid search** to identify optimal hyperparameters. Again, time was too short to do this, and so we settle with setting up a scheduler for learning rate, setting an initial high value for learning rate, and progressively lowering it as validation loss plateaus.

Weight decay is added to the optimizer to regularize the mode during training, and a common value is selected (10^{-6})

6. Results

This section focuses on results obtained for the model through and after training. Training curves will be presented, as well as performance metrics discussed in section 4. Finally, we will discuss these results and what could have been done better over the course of the project, especially concerning material shortcomings and overfitting questions

6.1. Training

Due to the lack of time, the model was trained for as many epochs as possible. **Cross-entropy loss** was used with the previously-presented scheduler to reduce learning rate as validation loss plateaus. The **Adam optimizer** was used due to its robustness in the face of noisy data and sparse gradients. The formula for the loss function is shown here, given M classes with indicator y_{ic} of whether or not prediction i belongs to class c, and estimated probability of class belonging by the model p_{ic}:

$$L(x_i) = -\sum_{c=0}^{M} y_{ic} \log(p_{ic}) \tag{1}$$

Training accuracy and validation accuracy are also evaluated over the course of the training process. The training graphs of our final model are shown below (Fig.):

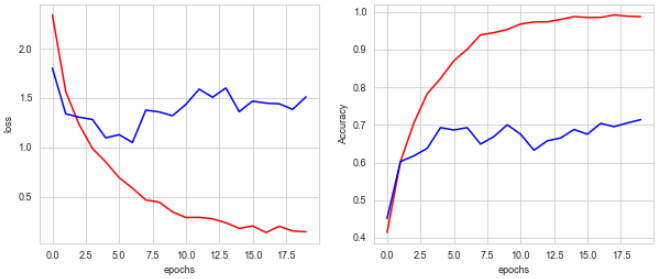

Figure 28: Training curves for the final model

Unfortunately, these curves display strong evidence of overfitting (big difference between training and validation split). Though we did not have enough time to train another model, potential solutions will be addressed later on. We do meet our goal of 70% accuracy for validation data however.

Learned classification features

One interesting element to consider when training classification models is the potential of applying K-means or KNN as a support to the classification block. To do this, we wish to visualize the data points in the test set and to evluate the distance and separation of classes.

We use the t-SNE algorithm to dimension reduce the feature output of the penultimate fully-connected later, which allows us to generate the following visualization of the data points:

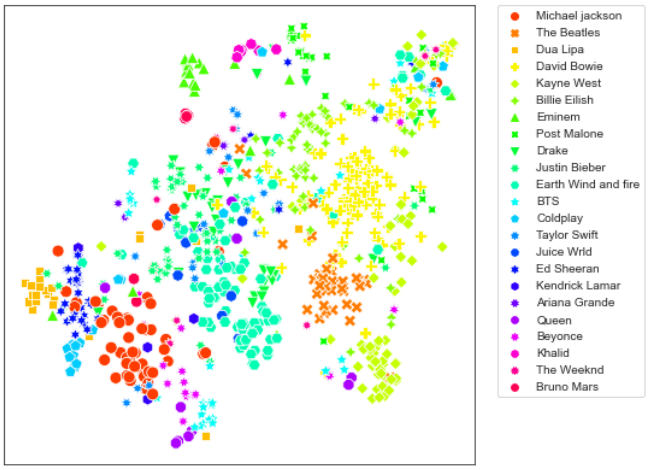

Figure 29: TSNE applied to the last layer of our trained model

The results are very convincing! clearly the model has learned something, as data points for each artist are often clustered together. Billie Eilish, Dua Lipa, and BTS are very distinctly separated, which is coherent with their particular vocal textures. The model might have learned korean language features for BTS, which is an amusing side-effect

6.2. Metric Performance

This subsection focuses on evaluating the model on the test set generated previously, according to the performance metrics discussed in section 4.

6.2.1 Multiclass Metrics

The following table summarizes the multiclass metrics discussed previously for the model on the test set.

type	accuracy	precision	recall	f1
Macro	0.682	0.679255	0.698233	0.671996
Weighted	0.682	0.704398	0.682500	0.679423

Table 3: Metrics table for macro-averaged and weighted averaged metrics

We fall just slightly short of our 70% accuracy goal for this model, which is a shame but not surprising given the technical problems encountered over the course of this project. However, recall seems to hold up well, with on average 70% of positives being correctly identified. The following confusion matrix (Fig.) allows us to see where our model makes mistakes, and which artists are classified the most as others:

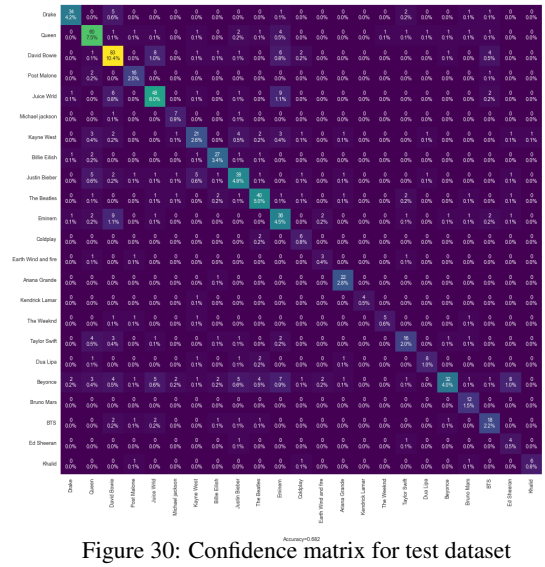

Figure 30: Confidence matrix for test dataset

In line with our acceptable accuracy, most classifications are along the diagnonal, which is a nice sanity check. Furthermore, the misclassifications are coherent: Bruno mars s often misclassified as Michael jackson for instance, which makes sense given their very similar vocal grains. Kendrick lamar, the two most influential rap artists in the dataset, are sometimes misclassified as each other. Globally, the model has learned coherent features, but not in depth enough.

6.2.2 One Vs All binary classification

The following table summarizes the results obtained by considering each class's metrics for a binary classification problem:

Ignoring the high accuracy for each class, which is normal as each class is very minoritary when considering the problem as a binary one, recall holds up extremely well in this case, which is very promising for eventually continuing the project in the future, and reassures us that the route taken was not completely incoherent.

6.3. Discussion

6.3.1 Results

Globally, the results are not exactly up to par, but are promising. The model learns vocal charateristics, as Table 4 shows that vocalists with very characteristic vocal features (Bruno Mars, Kendrick Lamar, Dua Lipa) perform very well accuracy-wise and recall-wise.

class	accuracy	precision	recall	f1
Drake	0.98	0.87	0.77	0.82
Queen	0.95	0.71	0.78	0.74
Post Malone	0.92	0.70	0.77	0.73
David Bowie	0.99	0.70	0.84	0.76
Juice Wrld	0.95	0.71	0.71	0.71
Michael jackson	0.99	0.58	0.78	0.67
Kayne West	0.96	0.62	0.51	0.56
Billie Eilish	0.98	0.75	0.82	0.78
Justin Bieber	0.95	0.66	0.64	0.65
The Beatles	0.97	0.74	0.75	0.75
Eminem	0.93	0.52	0.63	0.57
Coldplay	0.99	0.46	0.75	0.57
Earth Wind and fire	0.99	0.43	0.50	0.46
Ariana Grande	0.99	0.81	0.96	0.88
Kendrick Lamar	1.00	1.00	0.80	0.89
The Weeknd	0.99	0.83	0.56	0.67
Taylor Swift	0.97	0.64	0.53	0.58
Dua Lipa	0.99	0.73	0.57	0.64
Beyonce	0.93	0.86	0.39	0.53
Bruno Mars	0.99	0.60	1.00	0.75
BTS	0.97	0.60	0.67	0.63
Ed Sheeran	0.98	0.25	0.67	0.36
Khalid	0.99	0.86	0.67	0.75

Table 4: One-vs-all Metrics for each class

By reducing overfitting, and attempting to implement the previously-discussed dual model architecture, I believe the model could reach at least 80% accuracy. However, technical difficulties came upon the project, and that forced me to dumb down the project on certain aspects to finish on time

6.3.2 Addressing Overfitting

One huge problem as shown by the training graphs in Fig 28, is the gross overfitting the model seems to perform on the training data. Overfitting is addressed in multiple ways generally:

- **Reducing model complexity**
- **Adding training data**
- **Augmenting the training data**
- **Adding regularization to the optimizer**

two of these steps were already undertaken, as dropout layers and weight decay were implemented during training. The data was also augmented, as shown in section 2.

Addid training data might have been possible by downloading more artist discographies. However, at 5hrs per epoch, the training time to even dream about reaching 100 epochs was abysmal, so without more computing capacity, this is not feasible.

The last option is to reduce model complexity. It is true that the ResNest Model used in [2] is quite complex and quite deep, with 150 layers of cardinal blocks, which totals to over 250MB in model size. In [2], this was appropriate as the problem was multilabel, which is not the case here. However, it presented the added advantage of being pretrained on birdsong, which might have saved us days in training time.

So, I believe the best adjustment of trajectory for this project would be to implement the dual-architecture model described in section 4 and Fig. 26, with a much less complex convolution model. Perhaps **DenseNet** or **Inception**, given their good results in literature for similar tasks

6.3.3 Adjusting project goals along the way

As mentioned many times in this paper, the project was subject to many changes along the way:

- Ambitions were decreased from a real-time song classifier to a 10s clip artist classifier
- The dual architecture model was dropped due to the lack of computing power

But most importantly, as the main hardware rig for the project (**GeForce RTX2070 graphics card**) died over the course of the project, and my computer with it, progress was brutally reset midway through training the final model (100 epochs). Furthermore, the PC used to continue training was not cuda-capable, and training went from 45 mins/ epoch on the full dataset to 5hrs / epoch on only 8192 data points.

So, above and beyond lesson learned to always push my progress progressively, I believe this project could have done much better with more computation capabilities and more time, which is Why I will be continuing it on the side beyond the scope of this course.

7. Conclusion

Over the course of this study, we identified a real-world problem (the inflexibility of shazam's algorithm) to solve, and sought to solve it with deep learning methods. Through sourcing and analysis of data needs, we were able to build a relevant and high-quality custom dataset, which we then trained our model on.

However, the project was not exactly a success. Though the model did learn some coherent features and was visibly proficient regarding some artists, the lack of time and of resources did not allow us to thoroughly explore the capabilities of the model we initially chose to implement. This did provide some valuable lessons though, and the project will be continues beyond the scope of this course, taking the time to explore all the possibilities and eventually narrow down to song recognition, which I believe is possible.

List of Figures

List of Tables

References

[1] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam, "Spleeter: a fast and efficient music source separation tool with pre-trained models," *Journal of Open Source Software*, vol. 5, no. 50, p. 2154, 2020. Deezer Research.

[2] "Kaggle : Minimal implementation of densenet." https://www.kaggle.com/meaninglesslives/rfcx-minimal. Accessed: 2021-09-15.

[3] Y. Zeng, H. Mao, D. Peng, and Z. Yi, "Spectrogram based multi-task audio classification," *Multimedia Tools and Applications*, vol. 78, no. 3, pp. 3705–3722, 2019.

[4] K. Palanisamy, D. Singhania, and A. Yao, "Rethinking cnn models for audio classification," *arXiv preprint arXiv:2007.11154*, 2020.

[5] M. Dörfler, R. Bammer, and T. Grill, "Inside the spectrogram: Convolutional neural networks in audio processing," in *2017 international conference on sampling theory and applications (SampTA)*, pp. 152–155, IEEE, 2017.

[6] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[7] X. Zhang, Y. Gao, Y. Yu, and W. Li, "Music artist classification with wavenet classifier for raw waveform audio data," *arXiv preprint arXiv:2004.04371*, 2020.

[8] "Kaggle : Resnet + wavenet joint model." https://www.kaggle.com/aikhmelnytskyy/resnet-wavenet-my-best-single-model-ensemble. Accessed: 2021-09-15.

[9] A. Wang *et al.*, "An industrial strength audio search algorithm.," in *Ismir*, vol. 2003, pp. 7–13, Citeseer, 2003.

[10] "How shazam works : Inside developpers." https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition. Accessed: 2021-11-05.

[11] Z. Nasrullah and Y. Zhao, "Music artist classification with convolutional recurrent neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2019.

[12] S. Dieleman, P. Brakel, and B. Schrauwen, "Audio-based music classification with a pretrained convolutional network," in *12th International Society for Music Information Retrieval Conference (ISMIR-2011)*, pp. 669–674, University of Miami, 2011.

[13] J. Pons and X. Serra, "Randomly weighted cnns for (music) audio classification," in *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 336–340, IEEE, 2019.

[14] A. Van Den Oord, S. Dieleman, and B. Schrauwen, "Transfer learning by supervised pre-training for audio-based music classification," in *Conference of the International Society for Music Information Retrieval (ISMIR 2014)*, 2014.

[15] T. P. Van, N. T. N. Quang, and T. M. Thanh, "Deep learning approach for singer voice classification of vietnamese popular music," in *Proceedings of the Tenth International Symposium on Information and Communication Technology*, pp. 255–260, 2019.

[16] E. Sprengel, M. Jaggi, Y. Kilcher, and T. Hofmann, "Audio based bird species identification using deep learning techniques," tech. rep., 2016.

[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[19] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha, *et al.*, "Resnest: Split-attention networks," *arXiv preprint arXiv:2004.08955*, 2020.