École Centrale de Lyon
TFE 2022

Groover
Paris

Rapport Final de Travail de Fin d'études

Implementation of audio tagging at scale towards better musical recommendations

GUINOT Julien

Master : Acoustique

Tuteurs:	
ECL:	GALAND Marie-Annick
	JACOB Marc
Entreprise:	SAINT-SUPERY Santiago
	PALMIERI Romain

Acknowledgments

I'd like to thank the team at Groover for their warm welcome and sharing their passion for the music industry and artists throughout this internship. Specifically, I'd like to thank the data team : **Santiago, Michael & Joris** for accompanying me in this research project with pedagogy and valuable lessons.

I'd also like to thank the professors at Ecole Centrale de Lyon and the University of Adelaide for their support over the course of this double master's degree which was rendered very difficult because of COVID-19 and the fact that I was the first student following it. Despite all this, we went through with it, and it has paid off. This report is the culmination of that work, and of a personal journey of discovery in artficial intelligence and machine learning, but above all of passion for music and audio.

Résumé du rapport:

Les systèmes de recommandation ont une importante notable et croissante dans l'industrie de la musique. Plateformes de straming, de creation et de production, ou des réseaux sociaux musicaux, les systèmes de recommandation font partie intégrante de notre quotodien musical. Groover en particulier est une entreprise tech vicant à créer des opportunités de carrière pour des artistes indépendants en leur permettant d'envoyer leur musique à des professionels de l'industrie, avec pour but de grandir leur réseau, recevoir des retours sur leur musique, ou être promus par ces professionels. Ceci place Groover dans une position particulière de la sphère des systèmes de recommandations : recommander les professionels, qui sont à la fois des utilisateurs et des produits, aux artistes. Avec pour motivation d'améliorer leur système de recommandation actuel, l'équipe R&D de Groover veut inclure l'analyse automatique de charactéristiques audio dans leur pipeline de recommandation avec pour but de 1. Réduire le bruit parasite de charactéristiques auto-attribuées par des artists en basant ces charactéristiques sur des modèles de machine learning audio déterministes, et 2. Fournir un nouveau service aux artistes qui leur recommande automatiquement des étiquettes pertinentes selon leur musique.

L'objectif du stage est de rechercher l'état de l'art de techniques de traitement du signal et les modèles de charactérisation automatique de la musique pour une implémentation à grande échelle dans le système de recommandation existant. La charactérisation automatique de musique est une tâche qui relève d'apprendre à des modèles de machine learning à labeliser de manière pertinente des audios musicaux avec des labels tels que des characteristiques acoustiques, le genre, le sous-genre, les instruments,etc. Pour plus de contexte, une présentation de la mission à Groover et du contexte approfondi du stage est conduite en première partie, avec une présentation approfondie de la tâche de music tagging et son importance. Nous interprétons que les modèles audio de machine learning ont la capacité d'agit plus largement sur d'autres domaines de l'acoustique.

Dans une deuxième partie, une étude exhaustive de l'état de l'art est conduite, explorant les techniques de prétraitement audio communes dans le contexte de machine learning audio, les modèles de tagging audio disponibles et leur performance sur des datasets usuels, et les datasets disponibles pour conduire nos propres expériences de music automatic tagging basées sur l'état de l'art. Nous isolons 12 modèles pertinents à notre tâche de music auto tagging : FCN, CRNN, MusiCNN, SampleCNN, SampleCNN+SE, Ensemble SampleCNN, CNSSA, Semi-supervised CNSSA, SpecTNT, CALM, CLMR, ShortChunk et ShortRes. Il est montré à travers une étude de l'état de l'art que les techniques de pré-traitement préférées appliquées à l'audio en entrée de ces modèles est la génération de mel-spectrogrammes avec une transformation logarithmique, une fréquence d'échantillonage de 16kHz ou 22kHz, et un overlap de fenêtre de 50%, avec une taille de fenêtre variant entre 256 et 1024. Il est montré que l'audio pur peut performer adéquatement sur des datasets plus volumineux et a du potentiel dans le domaine d'apprentissage de représentation audio. Trois datasets sont montrés comme étant pertinents pour notre tâche de tagging : MTG-Jamendo, FMA, et MTAT.

Après une sélection de modèles basée sur plusieurs considérations pertinentes vis-à-vis des contraintes du stage, 3-4 modèles sont sélectionnés pour être testés sur un dataset construit par nous-même. Trois types de labels sont isolés pour être testés pour chaque modèle: Genres, Sous-genres, ambiances/thèmes/instruments et tous les tags combinés. Un nouveau dataset est construit pour ces quatre types de labels. Les modèles sélectionnés sont entrainés sur ces datasets en utilisant les log-melgrams mentionnés précédemment comme donnée d'entrée. Les modèles entrainés montrent des résultats compétitifs avec l'état de l'art sur les métriques sélectionnées avec un potentiel d'amélioration basé sur notre tâche jamais explorée et le nouveau dataset construit pour ce rapport. Les modèles de tagging présentent également un intérêt en tant qu'extracteurs de représentations figés et leur rôle en tant que bloc de pré-traitement du signal généralisable à d'autres applications est exploré. Un diagnostic de potentielles améliorations basées sur la performance par-classe des modèles sur chaque type de tagging est proposée. enfin, le problème de classification par instance est abordé et il est montré utilisant des techniques d'aggrégation au niveau de chaque musique que l'aggrégation moyenne résulte en de meilleures performances en général sur l'évaluation de musiques entières.

Mots-clés libres: Traitement du signal, Audio, Acoustique, Technologie de la Musique, Music Information Retrieval, Intelligence Artificielle, Tagging automatique Musical, Machine Learning

Abstract:

Recommendation systems have a notable and growing importance in the music industry. Be it music streaming platforms, music creation tools or music-related social network recommendations, recommender systems are part of our everyday music life. Groover specifically is a tech company aiming to create opportunities for independent music artists by allowing them to send tracks to industry profesionnals to expand their network, receive feedback, or be promoted by said curator. This puts Groover in a specific spot in the recommender system space : recommending professionals, who are both users and products, to users. As a motivation to improve their current recommendation system, the R&D team at Groover wants to include audio features within the pipeline to 1. Denoise artist-chosen characteristics by basing artist tags on deterministic representation extraction models and 2. Providing a new feature of recommending relevant tags to artists based on their sample tracks.

It is in this context that this master's internship takes place. The goal is to research state of the art of music audio preprocessing techniques and automatic music tagging models for later implementation at scale into the recommendation system. Music tagging is the task of teaching learnable models to label music tracks with relevant labels, such as acoustic characteristics, genres, subgenres, instruments, etc. To this end, a presentation of the context at Groover and of recommendation systems is conducted. The task and importance of music automatic tagging is presented as well, with interpretation on the ability to extend audio machine learning work to other domains of acoustics. In a second part, a comprehensive study of the state of the art is conducted, researching common audio preprocessing techniques in the context of machine learning applied to audio, audio tagging models and their performance on canonical datasets, and the available datasets to construct our own tagging task based in the state of the art. Following this study, we isolate 12 models that are relevant to the task of automatic music tagging : FCN, CRNN, MusiCNN, SampleCNN, SampleCNN+SE, Ensemble SampleCNN, CNSSA, Semi-Supervised CNSSA, SpecTNT, CALM, CLMR, ShortChunk and ShortRes. It is shown through study of state of the art that the prefered preprocessing techniques applied to audio as input representations of machine learning models is the construction of log-scaled mel-spectrograms with an audio sample rate of 16kHz or 22kHz, and a window hop of half the size of the fft window, which varies for top results between 256 and 1024. It is shown that raw audio can also perform adequately on larger datasets and shows promise in the field of representation learning. three datasets are exhibited that can be of use to this specific tagging task : MTG-Jamendo, FMA, and MTAT

After model selection based on multiple considerations relevant to the time frame of the internship, 3-4 models are selected to be benchmarked on a custom dataset. Four tagging types are isolated to be benchmarked upon : Genres, Subgenres, Mood/theme/instruments and All tags combined. A custom dataset and split is built for all four of these tagging types. Selected models are implemented and trained on the relevant datasets using the previously mentioned log-melgrams as input. Trained models show competitive results with state of the art on appropriate metrics with room for improvement based on the novel task of music tagging proposed in this thesis. The interest of models as frozen representation learners and standalone learnable preprocessing blocks is argued. A diagnosis of potential improvement routes based on per-tag performance on each dataset is also proposed. Finally, the issue of multi instance classification is tackled and it is shown using song-level aggregation on the selected datasets that mean pooling results in better performance overall on song-level evaluation.

Keywords: Digital Signal Processing, Audio, Acoustics, Music Information Retrieval, Music Technology, Artifical Intelligence, Music Auto Tagging, Machine Learning

Contents

1. Introduction

Music intelligence is a relatively novel field in the domain of Artificial Intelligence (AI) and Machine Learning (ML). The general of goal Music Intelligence - or Music Information Retrieval (MIR) - is to extract relevant insights and context about music or a musical piece through learning models and neural networks. As music is a generally highly subjective field, the field of music information retrieval as well as general audio processing using ML is highly dependent upon domain knowledge and previous research in the fields of **signal processing, acoustics, musical acoustics**, and many more. The collaboration of these domains has, in recent years, led to many impressive tasks taken on by machines rather than humans. These include acoustic space classification, acoustic event detection, speech-to-text translation, computer music generation, audio synthesis, music classification, source separation, automatic transcription, and more. It is a field that combines artistic expression and highly technical research in the fields of both and acoustic engineer / researcher and a machine learning engineer /researcher.

In recent years, music tagging, classification and characterization has become increasingly important for big players in the streaming space. Spotify, Deezer, Apple music, TikTok, YouTube have all implemented content-based recommendation for better musical recommendations to their listeners and better playlist generation. This progress has gone hand in hand with the progress of recommender systems on other platforms, which aim to recommend the best item in a catalogue to the user based on many factors (past user behaviour, personal content preference, global instantaneous trends...). It is essential to extract meaningful characteristics about musical audio for these big players to be able to recommend the best song to the end user, generate the best possible playlists, and predict future trends. Acoustics and Signal Processing play a key part in this characterization. By parametrizing audio representations and applying appropriate signal processing before the prediction pipeline, these musical predictive models can witness large gains in performance, and subsequently, their recommender counterparts do as well.

Groover is a company that falls nicely within this space with room to grow. Summarily, the aim of the company is to allow independent artists to send their tracks to music industry influencers. By guaranteeing a response from the influencers, the end user, the artist is given the insurance of constructive feedback, with a possibility of the artist sharing the track to their platform. To this end, Groover attemps to provide the best influencer recommendations to the artists using the platform. Again, these recommendations can be based of of past user trends, behaviour and current trends among influencers. However, it makes a lot of sense, both logically and business-wise, to propose high-quality content-based recommendations using the tracks that artists submit. It is in this context that the internship that serves as basis for this Thesis takes place. This report overviews the proceedings undertaken throughout the internship towards implementation music tagging-based recommendations in the already existing recommender system at Groover. Firstly, a comprehensive review of the context of the internship will be conducted, to provide all the necessary elements to fully understand the scope, importance and objectives of the internship. The report will then be separated into two main parts, representing the two major phases of the internship.

In a first part, the current state of the art of research in audio music tagging will be extensively and comprehensively covered. Audio preprocessing techniques, which stem from the field of signal processing and can be beneficially applied to the input to music tagging models, will be explored. The current existing music tagging models in the MIR landscape will be explained and dissected, as well as dicussion on the ability of music tagging and classification models to act as representation learners for audio signal and thus to be considered in and of themselves as (learnable) preprocessing techniques. Of course, such predictive modeling models need examples to learn from. This part will also cover and explore the available open source datasets for music tagging : Their data distribution, composition, available tags, and available metadata.

The second part of the thesis will focus on implementation and results of the audio tagging models at scale for all the models. To this end, the exploration and analysis of the available data will be covered, including data cleaning and quality analysis. Considerations for useful tags for the task at hand will also be covered, as well as a preliminary selection of models based on restrictions such as available space, computation cost, training time and inference cost. The process of preparing a final dataset for training and evaluation using all available music tagging datasets is documented in this part as well, with considerations of Wrangling, Merging and Resampling the data to provide better training. The design of the preprocessing and audio processing pipeline will be explained, as well as a word on audio augmentations in the context of the task at Groover. Training and results of each model for each task will be documented, as well as the task of bringing the final model into production and considering at-scale music processing, which is the end goal of the internship. A last section will focus on future considerations for better music tagging, concerning both audio processing techniques and novel models and representations to be used.

2. Company and Context

To understand the steps needed to conduct music tagging at scale and more specifically why this music tagging needs to be conducted, context must be provided, about both the company, Groover, and the need that the internship aims to fill. This sections focuses on the presentation of Groover, the team in which the intern was included, and how music tagging aims to improve the existing recommender system.

2.1. Company

2.1.1. Groover's mission and goals

Groover, founded in 2018, is a company which aims to provide the service to artist of sending their music to music industry professionals. By influencers, we mean : Spotify playlist curators, sound engineers, Youtube channels, music experts, soundcloud curators, and many more. The main mission of Groover is to **Empower all independent artist to get their music heard and accelerate their careers**, with the key rule to serve artists everyday.

Recently, Groover has been putting efforts into R&D and diversifying their services :

- Providing a curated list of groover-favorite artists with **Groover Obsessions**, with new releases being promoted and artists being followed by the Groover team
- Further helping artists accelerate their careers with **Groover Masterclass**, which aims to instruct artists about the music industry and give them insights into the workings of getting their careers off the ground

Groover has recently been gaining a lot of traction with important partnerships, prevalent ads, and artists kickstarting their career thanks to their services. Their growing market share in the US and globally, with influencers, artists and influencer ambassadors in over 20 countries, is key to their goal of being the go-to platform for artists to develop their careers.

Business Model

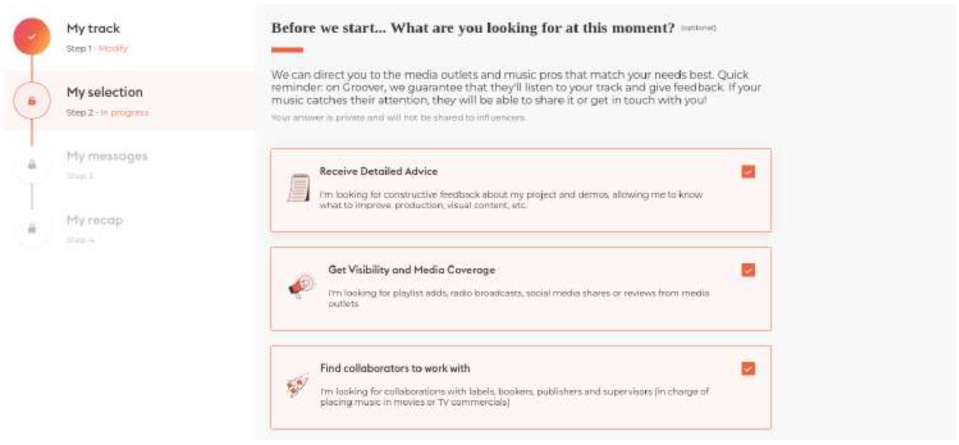
On the Groover Website, artists can start what are called "campaigns" and send one of their tracks to recommended curators within an alloted budget. Each curator has a certain cost to send the track to. The following figure (2.1) shows the services proposed by Groover for an artist after providing basic information about their track, in line with their goals : **Receive advice, get media coverage and exposure, and find collaborators**.

Following this step, the recommender algorithm, which is at the core of the provided service here, selects a list of relevant curators for the artist based on the artist's music, the curators's taste, and past behaviours of both the artist and the curator **if the artist is not new to the platform**. Figure 2.2 shows the recommendations for a test track.

Of course, the better the recomendations, the better the chance of the artist getting something back from his campaign, be it consistent and relevant feedback, getting posted to a spotify playlist, or scoring a collaboration. And the better artist satisfaction, the closer Groover is to its mission and the faster the growth. So, the recommender system is an important piece of the ecosystem at Groover, and this internship focuses on exploiting audio data and signal processing to improve it further.

2.1.2. Organization of the Groover team

To understand how the data team, which I joined, articulates around the rest of the Groover Team, the following figure (2.3) shows the basic organization of the team.



My track
Step 1 · Modify

My selection
Step 2 - In progress

My messages
Step 3

My recap
Step 4

Before we start... What are you looking for at this moment? (optional)

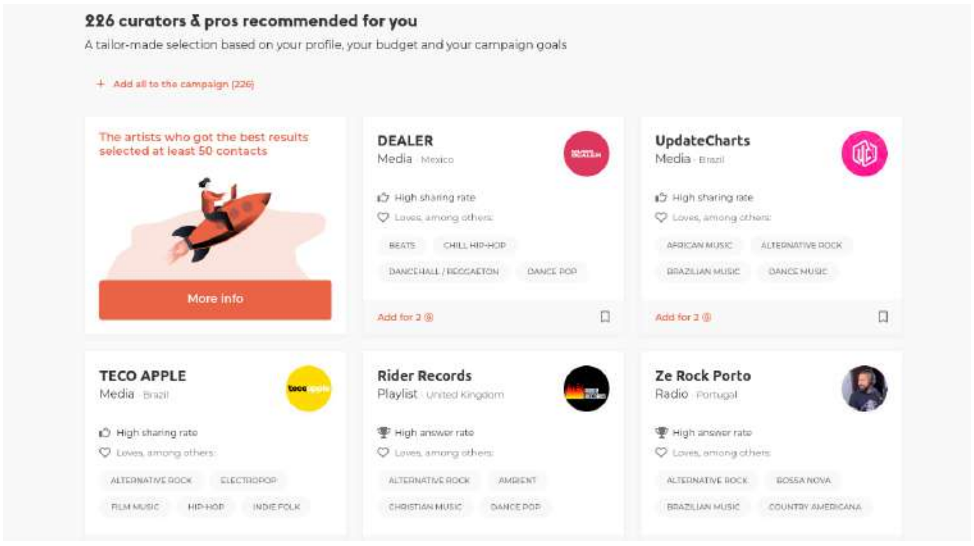
We can direct you to the media outlets and music pros that match your needs best. Quick reminder: on Groover, we guarantee that they'll listen to your track and give feedback. If your music catches their attention, they will be able to share it or get in touch with you!
Your answer is private and will not be shared to influencers.

Receive Detailed Advice ✓
I'm looking for constructive feedback about my project and demos, allowing me to know what to improve: production, visual content, etc.

Get Visibility and Media Coverage ✓
I'm looking for playlist adds, radio broadcasts, social media shares or reviews from media outlets

Find collaborators to work with ✓
I'm looking for collaborations with labels, bookers, publishers and supervisors (in charge of placing music in movies or TV commercials)

Figure 2.1.: Choosing campaign goals on Groover.co



226 curators & pros recommended for you
A tailor-made selection based on your profile, your budget and your campaign goals

+ Add all to the campaign (226)

The artists who got the best results selected at least 50 contacts
More info

DEALER
Media · Mexico
High sharing rate
Loves, among others:
BEATS · CHILL HIP-HOP · DANCEHALL / REGGAETON · DANCE POP
Add for 2

UpdateCharts
Media · Brazil
High sharing rate
Loves, among others:
AFRICAN MUSIC · ALTERNATIVE ROCK · BRAZILIAN MUSIC · DANCE MUSIC
Add for 2

TECO APPLE
Media · Brazil
High sharing rate
Loves, among others:
ALTERNATIVE ROCK · ELECTROPOP · FILM MUSIC · HIP-HOP · INDIE FOLK

Rider Records
Playlist · United Kingdom
High answer rate
Loves, among others:
ALTERNATIVE ROCK · AMBIENT · CHRISTIAN MUSIC · DANCE POP

Ze Rock Porto
Radio · Portugal
High answer rate
Loves, among others:
ALTERNATIVE ROCK · BOSSA NOVA · BRAZILIAN MUSIC · COUNTRY AMERICANA

Figure 2.2.: Recommended influencers for a test track on Groover.co

The team is divided into five main areas : **Business, Tech, Product design, Data & G&A**. We do not go into detail for members of each team except for ours, the data team. Of course, Groover is a growing company, so transversal interaction between roles is highly possible, and no structure is frozen (One of the reasons this company seemed attractive to me). However it is important to understand how all of these teams relate to the data team, so we specify each of the teams' global roles.

- **The marketing team** focuses on promoting Groover on social media and to grow the presence and platform of the company. *The data team ingests data and provides insight to this team, such as ad click-through rate, user trends and growth, and campaign efficacy, per their request*
- **The growth team** focuses on growing the user base of the Groover platform through various media. Again, per their request, the data team provides insight on all the ingested data by creating visualizations.
- **The curators team** manages the base of curators on the platform (curators is the vernacular term for influencers). Operating on a per-geographic region basis, they recruit new curators and manage the existing ones. They work in close collaboration with the data team to not only request metrics and insights on curators, but also to create ML models to flag unsatisfactory curators, who provide bad or abusive feedback, or do not deliver on provided feedback.

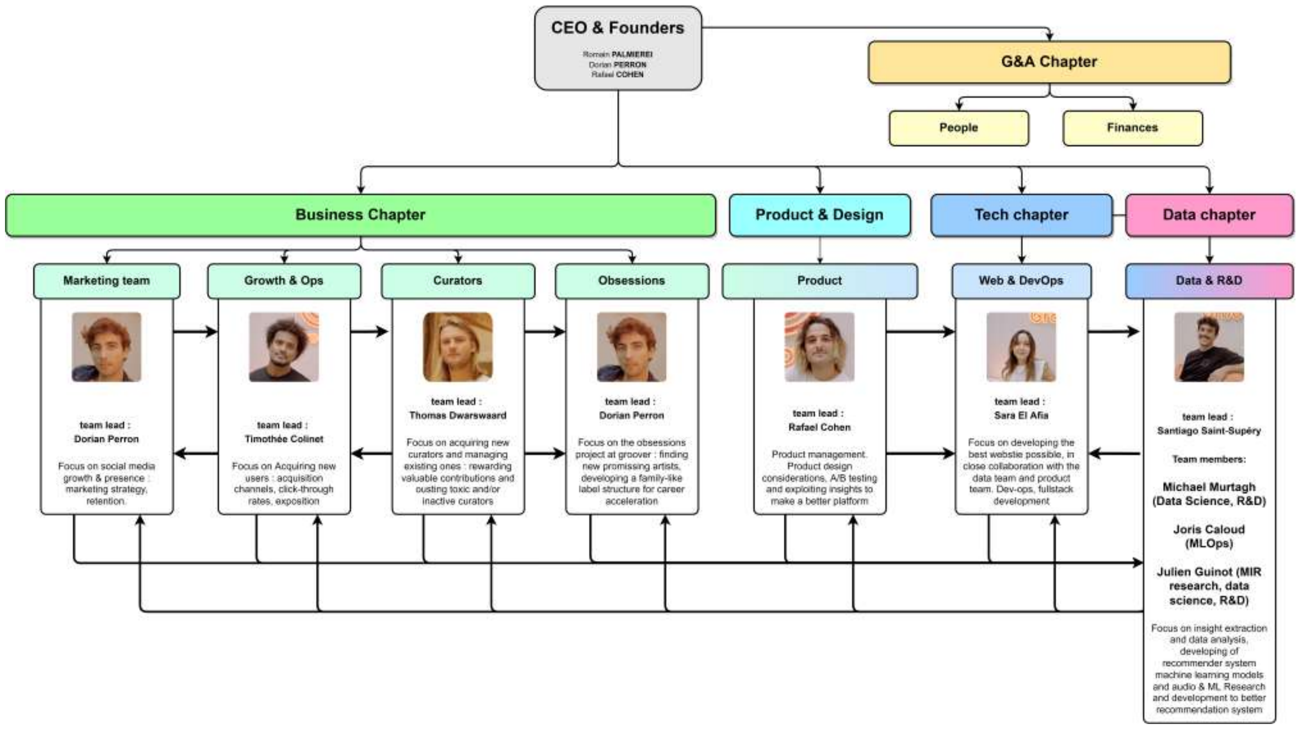

CEO & Founders
Romain PALMIEREI
Dorian PERRON
Rafael COHEN
G&A Chapter
People
Finances
Business Chapter
Product & Design
Tech chapter
Data chapter
Marketing team
team lead : Dorian Perron
Focus on social media growth & presence : marketing strategy, retention.
Growth & Ops
team lead : Timothée Colinet
Focus on Acquiring new users : acquisition channels, click-through rates, exposition
Curators
team lead : Thomas Dwarswaard
Focus on acquiring new curators and managing existing ones : rewarding valuable contributions and ousting toxic and/or inactive curators
Obsessions
team lead : Dorian Perron
Focus on the obsessions project at groover : finding new promising artists, developing a family-like label structure for career acceleration
Product
team lead : Rafael Cohen
Product management. Product design considerations, A/B testing and exploiting insights to make a better platform
Web & DevOps
team lead : Sara El Afia
Focus on developing the best webstie possible, in close collaboration with the data team and product team. Dev-ops, fullstack development
Data & R&D
team lead : Santiago Saint-Supéry
Team members:
Michael Murtagh (Data Science, R&D)
Joris Caloud (MLOps)
Julien Guinot (MIR research, data science, R&D)
Focus on insight extraction and data analysis, developing of recommender system machine learning models and audio & ML Research and development to better recommendation system

Figure 2.3.: Organization of the groover team

- **Obsessions team** focuses on recruiting new artists for Groover's label-like acceleration service, and dealing with events which showcase these artists. They interact with the data team to request recommendations for artists applying to these events (same as the online recommendation pipeline), and, as always, to request data insights and visualization

Moving on to the technical chapter, the teams are:

- **The product team** is halfway between marketing and tech. While it focuses on such metrics as click-through and activation funnel to analyse the components and steps in the website, it also delves into product design and management to create the best platform possible. Interactions with the data team include insights on A/B testing, and other metrics such as the activation funnel.
- **The web & devOps team** works in close collaboration with the data team on data ingestion and database structure. New data acquisition goes through the back-end team within the web&devops team. They focus on deployment of the website, front-end development in line with the product teams' asks, and continuous development/integration pipelines.

The data team

Finally, the data team, in which this internship took place, is in charge of creating data pipelines to generate visualization for all of the aforementioned teams (Data analysis function). It also develops state of the art machine learning models to better the recommendation pipeline and conduct other tasks (curator flagging, Natural Language Processing for better recommendations and automatic biography analysis...) (Data science function). Tight collaboration with the Web&Devops team happens on the data ingestion and infrastructure for the platform : This is the MLOps responsiblity of the team.

Finally, and this is where the internship takes place, the R&D facet of the data team focuses on research and development in Machine Learning and in the case of this internship, audio signal processing. In the scope of this facet, state of the art is explored to produce new relevant results towards implementing new models in the scope of the data science responsibility of the data team. The members of the data team are described here:

- **Santiago Saint-Supéry**, data team lead, participates in all facets of the data team
- **Michael Murtagh**, data scientist, aids in data analysis and data science matters, as well as R&D
- **Joris Caloud**, MLOps engineer, focuses mostly on the MLOps part of the team's responsiblities
- **Myself**, mostly focused on R&D (almost exclusively), though I participate sometimes in data analysis matters when help is needed, which has allowed me to broaden my horizons in data analysis frameworks and tools.

2.2. Context of the internship

To understand the role of audio processing in the internship, we must understand the technical context that goes behind the recommender system at Groover. to do that, we have to understand the current recommendation pipeline and how the intuition that signal-based music tagging can benefit it.

2.2.1. Data-driven artist-to-curator recommender system

What is a recommender system?

A recommender system is a type of machine learning algorithm, the goal of which is to make recommendations to a user based on a catalog of kown objects in a database. Use cases of such algorithms could be online shopping (**Amazon, Ebay, Asos or any online retail store**), Music recommendation (**Spotify, Apple music, Soundcloud...** , Video recommendation (**Youtube, Spotify**), Book recommendations... The use cases are numerous. Recommender systems as of late have become a high-value space in the machine learning landscape due to the existence of more and more complex and diverse models and concepts to apply to the idea of a recommender system. Recommender systems can be divided into two types : Collaborative recommender systems and content-based recommender systems.

Collaborative recommender systems

Collaborative recommender systems rely on the interactions of other users with items in the catalog to deduce which item to best recommend. Consider a user-item interaction matrix where each interaction between a user and an item (an interaction can be a like, a purchase, a view, a rating, an early skip, a full listen in the case of songs...), is given an interaction score. Figure 2.4 shows the global appearance of a user-item interaction matrix. For our use case we consider the score is a 1-5 rating:

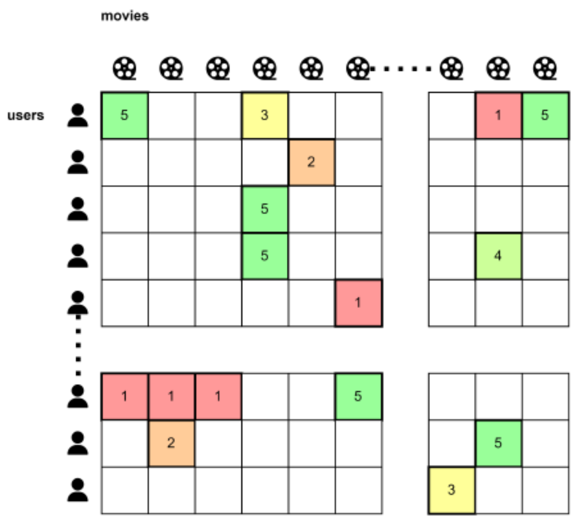


Figure 2.4.: Basic User-Item Interaction Grid for recommender systems

The intuition here is that by relying upon past user interactions, it can be possible to intuite what the user will choose next. This is possible by two methods : **Method-based inference** and **memory-based inference**. In memory-based inference, items and users are represented by large relatively sparse vectors, and recommendations are done based on nearest neighbours. **Users similar to each other are likely to Want the same items**. With memory-based inference however, an attempt to exhibit a latent model is done, model which can capture the relation between users and items. Users and items interactions are distilled into learned denser vector representations and a model is built to try to predict the best interaction in all the catalog.

However, this method suffers from the **cold start problem**: How to recommend anything to a user that has never interacted with anything before? This prompted the need for the other type of recommender systems :

Content-based recommender systems

Content based recommendation, on the other hands, focus rather on the features of the users and the items. For instance, the intuition that young men would prefer action movies than elderly women, is relevant. So is the inuition that French people would prefer French-spoken podcasts than spanish citizens. This intuition can yield a simple method that allows for recommending catalog items to users that have never purchased anything insofar that their features are available. Consider a model in which the respective features of the user and the item are used as input features for the model, which learns on pre-existing interactions without consideration for user trends and history. Figure 2.5 shows the basic working of a content-based recommender system, with feature vectors for the item and the user.

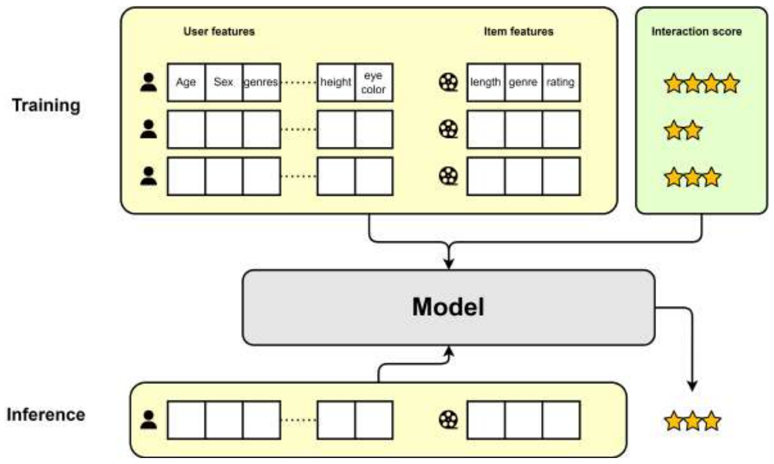


Figure 2.5.: Content-based recommendation example

So the model learns from the scores and codified features of the user and items and this learning is generalized at inference time by attempting to predict an interaction score for each item in the catalog (summarily, as some catalogs are very large and it is not possible to allow such computation times. However, the methods used to deal with very large catalog search in recommender systems are outside the scope of the internship and this report, so will not be covered though very interesting). Predictions are then ranked according to best score and the top items by predicted score are presented first to the user.

Hybrid models and a quick summary of the evolution of recommender systems.

These methods are somewhat mutually exclusive, as content-based recommendation does not take into account past behaviour of users. Collaborative filtering, on the other hand, does not take into account item features. So, most current recommender systems are what are called hybrid recommender systems : a mix of the two previous methods. Figure 2.6 summarizes the difference between the previous sytems and shows how a hybrid compromise is beneficial to the complementarity of the previous methods.

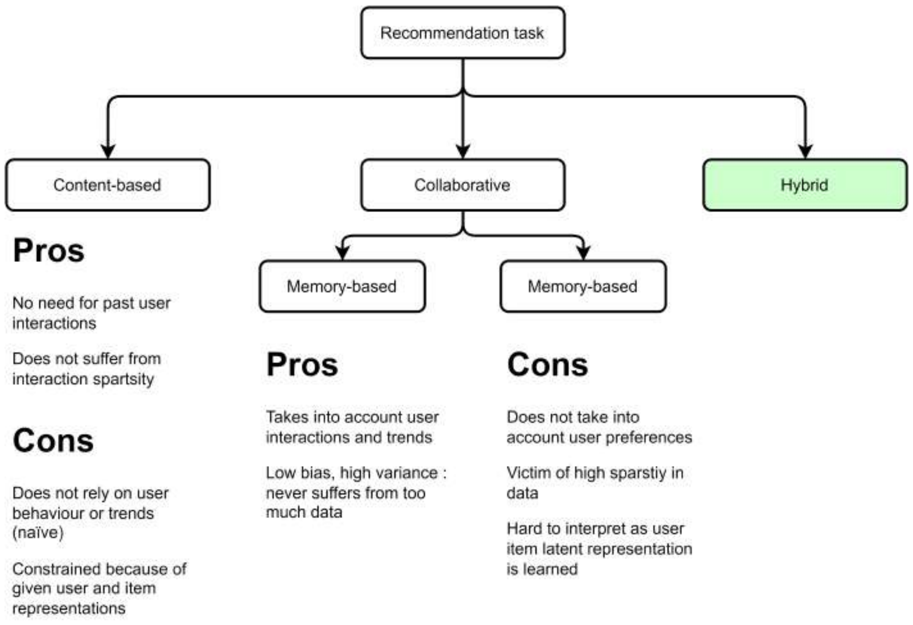

Recommendation task
Content-based
Collaborative
Hybrid
Memory-based
Memory-based
Pros
No need for past user interactions
Does not suffer from interaction spartsity
Cons
Does not rely on user behaviour or trends (naïve)
Constrained because of given user and item representations
Pros
Takes into account user interactions and trends
Low bias, high variance : never suffers from too much data
Cons
Does not take into account user preferences
Victim of high sparstiy in data
Hard to interpret as user item latent representation is learned

Figure 2.6.: Summary of the types of recommender systems

A deep dive into the recent development of recommender systems is outside of the scope of this internship. However, it is important to at least conduct an overview of the recent advances and a quick history of recommender systems to understand how the recommender system at Groover fits into this timeline.

A quick history of recommender systems

Collaborative item-item approaches and user-user approaches come first in the history of recommender systems. Essentially, this approach considers items in the item space and users in the user space and performs K nearest neighbours on the other vectors taken from the interaction matrix. Figure 2.7 below shows the working principle of item-item and user-user approaches:

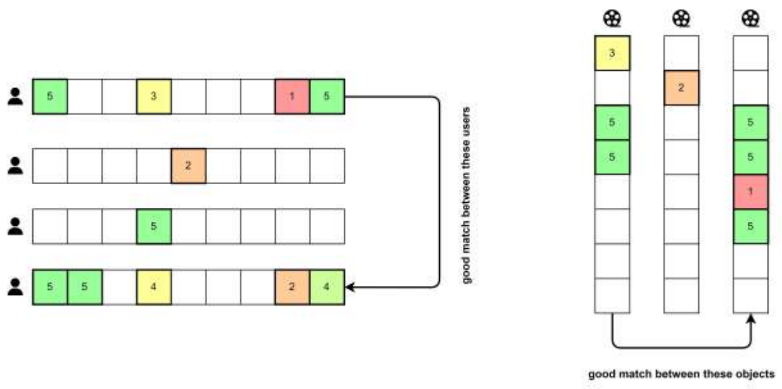

good match between these users
good match between these objects

Figure 2.7.: item-item and user-user approaches for recommendation

To bring precision to this difference, in the item-item approach, the user is recommended items similar to his preferred items. In the user-user approach, the user is recommended items preferred by users with similar tastes. A few issues arise here: for large catalogs and large user bases, the search algorithm does not scale well at all ($KNN\ is\ in\ O(nmk)$). Furthermore, successful items will always be recommended in this fashion. To palliate this, Interaction Matrix Factorization was conceptualized.

The intuition is as follows : the user-item interaction matrix is a representation of interactions between users and items that is the result of latent descriptors that are either known or unknown. For content based filtering, we can provide these features and let the model learn the interaction between these features. With matrix factorization, we let the model learn the representations themselves. We assume the matrix M can be decomposed into the product of a User matrix U and a item matrix I with a residual error reconstruction matrix E, as shown in Figure 2.8 [33] [16]:

$$M = UI + E \tag{2.1}$$

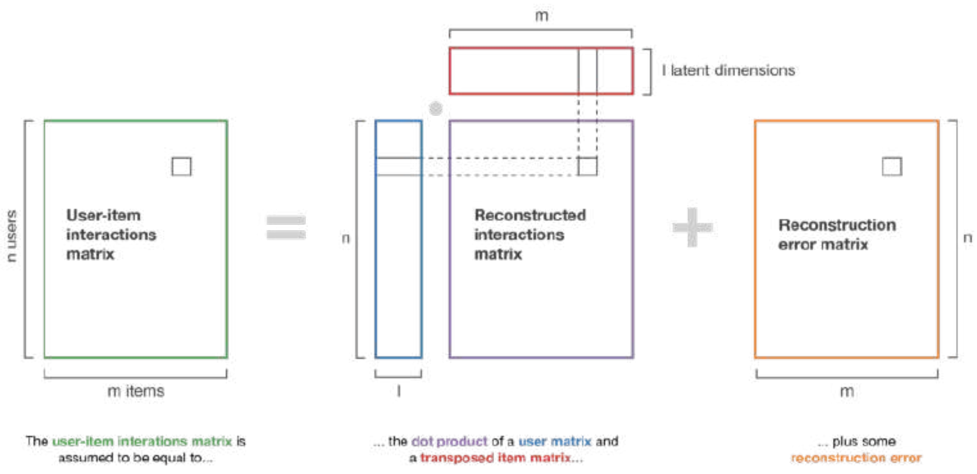

m
l latent dimensions
n users
User-item interactions matrix
m items
n
Reconstructed interactions matrix
l
Reconstruction error matrix
n
m
The user-item interations matrix is assumed to be equal to...
... the dot product of a user matrix and a transposed item matrix...
... plus some reconstruction error

Figure 2.8.: Matrix factorization working principle (Image from[33])

Finding the best latent representation for users and idem corresponds to a minimization problem defined as follows [33]:

$$(U, I) = argmin_{UI} \sum_{i,j \in E} [U_i I_j^T - M_{ij}]^2$$

Which can be learned by gradient descent backpropagation classically as with with any machine learning algorithm. This allows to compute the recommendation score for a user and an item by calculating the associated learned representation vecors, which deals with the issue of scaling. However, a new issue arises, which is that of Matrix Sparsity. These interaction matrices are often practically empty for large catalogs (sparsity at Groover is 99.5%), which makes learning latent representations a high variance problem, requiring large latent representations.

A few solutions were proposed to paliate this over which we will not go into detail on, as the state of the art rapidly moved on to newer and more problem-appropriate models:

Sequential and session-based recommenders, modern recommender architectures.

Following these initial models, newer and deeper models which aim to encompass more inherent characteristics of what a recommendation problem is made their appearance. Deep recommender systems started combining collaborative and content-based recommender systems by applying learnable models to vocabulary embeddings of both users and items, to combine both user behaviour representations and item feature representation [45] [14]. The general structure of such a network is shown in figure 2.9:

After combining user-behaviour and content into one model the notion of sequentiality came about : user behaviour while making choices based on recommendations is naturally sequential. The user interacts with a certain amount of items and the order of interaction is important. For instance, if the user listens to episode 1 of a podcast, recommending episode 2 next seems like a fair intuition. Without going into details. From 2014, which is when the original study for collaborative & content based recommendations featured [45], recommender systems aimed to tackle sequentiality:

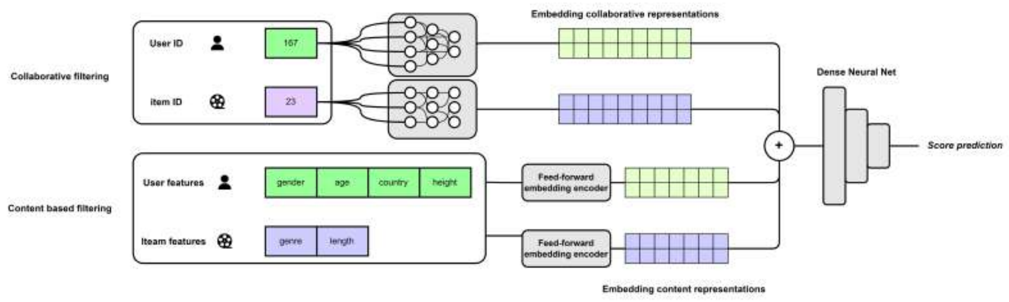


Figure 2.9.: A collaborative-content Neural Recommender System much like that at Groover

- Recurrent Neural networks tackled the problem of sequentiality first due to their appropriate architecture for handling sequences [35] [18] [13].
- Convolutional Neural networks followed suit [39] as well as graph neural networks due to their ability to model relationships between entities [53].

Sessionality is another aspect of recommendation that should not be neglected : between sessions a user will not necessarily have the same needs. Consider a user logging in to a electronics retail website to purchase a computer one day and a phone the next. If a computer was purchased the previous day, though It might make sense to recommend a computer based on the sequence, it does not based on the session.
Recurrent Neural Networks, again, tackled this problem first [15], and the session-based recommender system has since become state of the art [46]. Many domains have branched out from the exploration of these sequential and sessional models, including transformers, which have established themselves as *de facto* state of the art on recommendation problems [38], [5]. Recently, approaches in self-supervised learning [23] [54] and reinforcement learning [28] , [1] have shown the value of autonomous learning and trial-and-error algorithms for the recommendation problem. **Contextual bandits** for instance is an algorithm in which individual agents called bandits attempt to propose new recommendations using environment-based decision making using an evolving reward mechanism. This model has shown promise in recommender systems by introducing a stochastic dimension to recommendation and 'mixing up' the recommendations, learning when the recommended item is or is not interacted with, and converging upon human-like recommender behaviour that is not purely representation-driven.

This, of course, is only an overview of the possibilites linked to recommender systems and many more paths have been explored as of late with deeper and deeper architectures. The recommender system is important in the scope of this internship as it represents the core of R&D at Groover, and the objectives of this internship all converge towards building upon the recommender system and making it more performand using audio processing and deep learning applied to musical signals.

The Groover recommendation pipeline

This section focuses on taking a look at the particularities of the Groover recommendation pipeline and the architecture at the beginning of the internship. We define the intricacies of the Groover recommendation task in relation to other more classic recommendation tasks, as well as exhibit the current architecture for the model and the business-educated decisions acted upon to design the data preprocessing pipeline and model itself.

Recall that the objective of recommendation at Groover is to recommend relevant curators to independent artists based on their objectives for the track they have submitted for the campaign. This relevancy can be based on user - which are also referred to as bands - and curator features, but also on the sequence of interactions of bands with curators and curators with bands. The Groover recommendation task is a tricky one. The goal is not purely to recommend curators based as items - curators are humans or organizations of humans, so a purely catalog-like recommendation task, such as **Amazon, Ebay, HM or any retail store** would tackle, does not suffice to encapsulate the problem. Neither is it a purely network-driven task such as Facebook or Linkedin, where users are recommended to users, and (most of the time) the interaction between these users are equal. It is similar in that the band will send a request to a curator to listen to, review or share their track, and the curator will either accept or deny - as he would a connection request on Linkedin - however as the curator delivers feedback and a service to the band, it cannot be only seen as a user-to-user interaction.

The curator here, which is the recommended (to the *recommendee*, which is the band) acts as both a purchasable item and an interactable user. This difference with most common recommendation tasks makes the Groover recommendation pipeline an interesting task to tackle. As the pipeline is in constant evolution, the following explored the recommender architecture at the beginning of the internship.

The database at Groover stores information for both bands and curators, some of which can be both. This information, for bands, can be **The country of origin, a vector of relevant genres and subgenres, the goal of the campaign, relevant information extracted from the biography**. The same can be applied to curators with some extra features : curators are required at sign up to specify a list of **liked, neutral and hated** subgenres. Furthermore, artists and curators are specified by BandID and InfluencerIDs, which pertain to the collaborative aspect of the recommendation system. All interactions between bands and curators are annotated with a score from 0 to 1:

- 0 means the track was not accepted by the influencer : it is a negative interaction, and the model should learn to attribute a score close to 0 to interactions that will probably not end in an acceptation from the influencer
- 0.25 means the curator has provided feedback on the track but has not decided to share it
- 0.5 means the curator has agreed to share the track but has not yet done it
- 1 means the curator has shared the track

So interactions are scored in ascending order : better interactions are scored closer to one (or just have a higher score) and worse interactions score lower. To summarize this, the following tables show the database tables available for use by the recommender system :

Interaction table		
bandID	*InfluencerID*	*Score*
12	2	1
43	3	0
661	2	0.25

Table 2.1.: Interaction table example

Band table					
bandID	*Subgenres*	*Country*	...	*Bio information*	*Date joined*
1	[rock, pop]	FR		[1999, ...]	10/1/2021
2	[drumstep, hardcore]	US		[Michigan ...]	12/5/2020

Table 2.2.: Band Table example

Band Table						
BandID	*Country*	...	*Language*	*liked subgenres*	*hated subgenres*	*neutral subgenres*
1	FR		FR	[trance, techno]	[ballad, folk]	[acid]
2	USA		EN	...	...	...

Table 2.3.: Curator table examples

The task is similar to what was described before : the interaction table can be reformulated as an interaction matrix, and the band and influencer tables provide the features needed for content-based recommendation. The task can be formulated as a gression task with the objective to predict the score for a given interaction between a user and a curator. At inference time, the user who is asking for recommended curators is screened through the whole catalog of curators and the results are arg-ranked by score, providing in theory most relevant curators at the top.

Based on [45] [18], the recommender system at the start of the internship had the following general architecture (Figure 2.10) *Nota : for confidentiality purposes, the exact architecture of the recommender system shall not be divulged in this report. However, it is not relevant to know the exact blocks and parameters of each of these blocks in our case - nor is it relevant to the scope of the internship*:

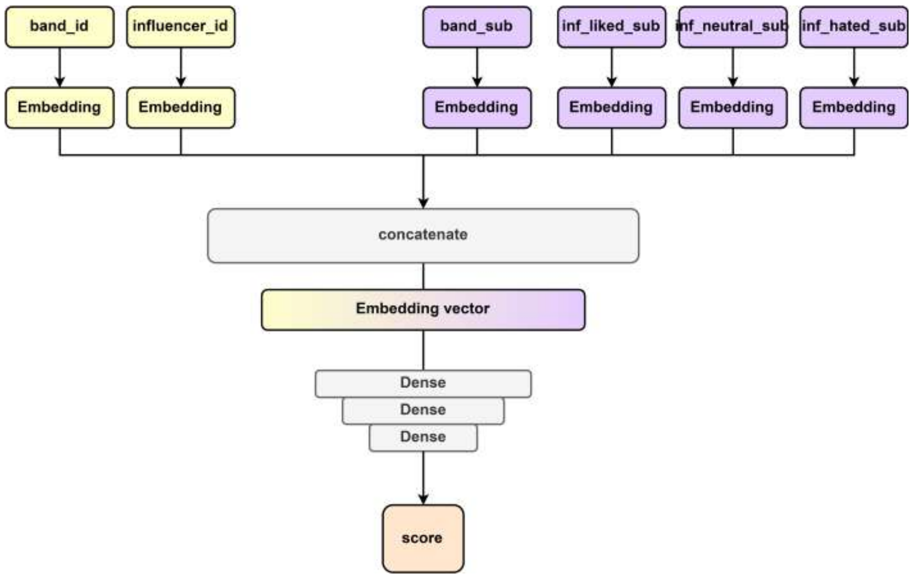


Figure 2.10.: Base recommender system architecture at Groover

Note that here only subgenre content features are shown, but others are taken into account. This a relatively basic collaborative-content neural model shown previously. However, as we have discussed in the previous section, curators are not catalog objects : they have behaviours just as bands do that might bias the recommender system. Business-motivated solutions are proposed to these issues to make the recommender system more prevalent to the considered use case.

- **Influencer that always accept tracks sent by bands will be strongly positively biased by the algorithm**. This is due to the collaborative filtering aspect of the recommender system. Intuitively, highly selective curators are more desirable and should be more highly ranked in the recommendations. The adopted solution is to penalize the score of influencers by a pickyness rate variable. The higher the pickyness rate of the influencer, the more the score of the interaction is boosted during training. The lower the pickyness, the more the score is penalized.

- **Influencers can bias the recommendation by saying they are partial to all subgenres**. The standard interaction is on average more positive between bands and curators that have similar liked genres. This is a relationship that the model will *a priori learn and take into account at inference time.* So, by saying that they love all subgenres, an influencer can be ranked higher for all artists, which is undesirable. **Again, the score is peanlized at training time for influencers who fill up the whole subgenre list in their liked subgenres section**

Finally, though the model learns relevant correlations between both content features (namely subgenre correspondence) and past interactions between users and influencers, we want it to be somewhat grounded in an intuitive use-case truth : **Bands have a higher chance of being accepted by curators with similar tastes in music.** And though the model is expected to learn this, it is not explicitely provided (recall that collaborative content filtering relies on learned and not provided representations). The solution to this is adjust the score at training time in accordance to the cosine similarity between vector representations of the subgenres for the band and liked and hated subgenres for the curators. The simplest vector representation to use that provides a known representation and is not learned is **one-hot encoding**. Given a vocabulary V of cardinality n;, the one-hot representation of a list of items of that vocabulary is the vector:

$$OH = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \vdots \\ \delta_n \end{bmatrix} \tag{2.2}$$

Where δ_i is the kronecker delta equal to 1 if the ith item of the vocabulary is in the list of items. To compute a ground truth of perceived similarity between the subgenres of the band and the liked and hated subgenres of the curator, we use cosine similarity, which is defined for two vectors A and B as :

$$Cosim(A, B) = \frac{A \cdot B}{|A||B|} \tag{2.3}$$

Which is a measure of the similarity of n-dimensional vectors. We then mitigate the score at training according to this obtained similarity. Including all score mitigation to reflect curator behaviour and business use case applicability, the final training score expression can be given as follows:

$$s_{new} = s_{original} - p_{subgenres} - p_{pickyness} + cosim(OH_{band}, OH_{liked}) - cosim(OH_{band}, OH_{hated}) \tag{2.4}$$

Where s denotes score and p denotes penalization. This is the final regression problem the recommender system aims to tackle. The following figure (Figure 2.11) shows the adapted architecture from previously with the inclusion of the mitigations

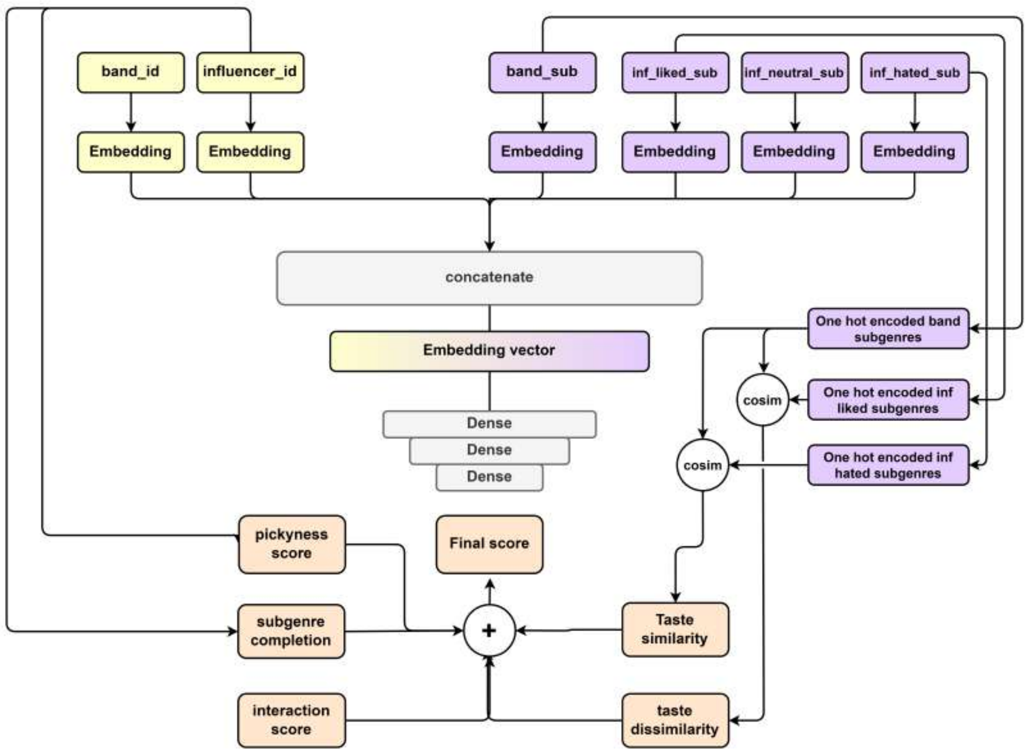


Figure 2.11.: Current recommender system

This is the architecture as it stands at the start of the internship. As mentioned previously, the recommender system is the subject of most R&D efforts from the data team at Groover. And as was stated previously, many directions have been explored by state of the art to better recommender systems. Here are the current directions that are being explored at Groover - without going into much detail as the specifics of these are outside the scope of the internship:

- **Contextual bandits**, which were described previously, have already been implemented and deployed on part of the user base. Using the recommender-specific and business-specific metrics which will be elaborated on in the next section, the effectiveness of the algorithm on the recommendation system will be evaluated and generalized to the whole user base of needed
- Efforts in natural language processing to extract relevant features from artist biographies
- Efforts in sequential and session-sequential recommender systems to model user sessions (campaigns)

And of course, the field of R&D which this internship is focused on is the incorporation of audio feature extraction models at scale in the pipeline to derive audio features from artist-uploaded. We will elaborate on the exact goal and potential value of these audio features in a following section but the intuition is simple : Groover is an audio-centered platform that conducts user-to-user recommendation. There must be some usable feature in the audio files provided by the artists to make better recommendation. This sets the context for the internship and explains the importance of signal processing, audio machine learning and latent representation learning in this context.

2.2.2. Music tagging, signal-based recommendation

An important domain of R&D at Groover is Audio, and more specifically automatic music tagging. This section focuses on explaining the task of music tagging as well as the potential applications at Groover. We also discuss the metrics, both business-oriented and ML-oriented, used to evaluate a recommendation system. As the goal of the internship is at term to develop a model and put it into production, we need measurable indicators that the boost in recommendation quality brought by our model is substantial.

What is music tagging?

Music tagging is a sub-task of the more general music classification task, part of a canonical and previously explored set of tasks in the domain of Music Information Retrieval, and more Specifically Discriminatory Music Information Retrieval.

Music is a subjective art and qualifying it objectively is debatable even by human metrics. However, it is possible to attribute to a musical track a set of **tags** in an attempt to describe it as concisely as possible. These tags are used to describe high-level information about a given piece of music and are used by music streaming platforms (**Spotify, Deezer, Apple Music, Youtube Music**) to construct playlist or make recommendations. Tags can be seen as a set of descriptors of music, which can be of many types : **Genres, Subgenres, Acoustic Features such as danceability, energy, Moods, themes, instruments, vocal registers...**. The following figure (Figure 2.12) shows a set of example tags for the song *South of the river*, by **tom misch**:

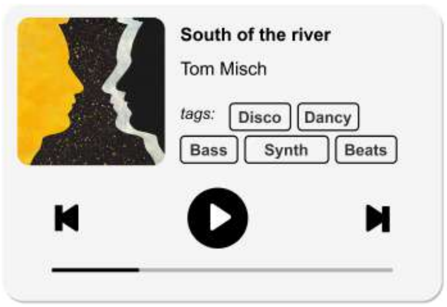


Figure 2.12.: An example of tags on a music track

These tags were manually annotated and thus subject to high variance and subjectivity until [40] aggregated hand-crafted acoustic and signal processing descriptors of frame-level samples of music pieces to predict the musical genre of the audio sample. This prompted the exploration of Deep Neural Networks for use as music classifiers, starting with [6] in 2016. Deen Neural Networks provide the advantage of learning relevant harmonic and temporal representations of audio signals through their downstream task objective and thus sidestep the need for hand-crafted audio features [6].

So the goal of music tagging is to predict a set of labels for a given audio track. This makes it a multi class classification task per the multiple labels that can be assigned to the tracks. However, there can be multiple labels - tags - attributed to a single track, which makes this task a **multilabel multiclass classification task**. The objective of the task becomes the following : given an audio signal, model a predictive score for each label the model is trained on based on the audio features of the track and rank these scores to predict the relevant tags for the track. The following figure (Figure 2.13) shows the inference pipeline for the general task of automatic music tagging:

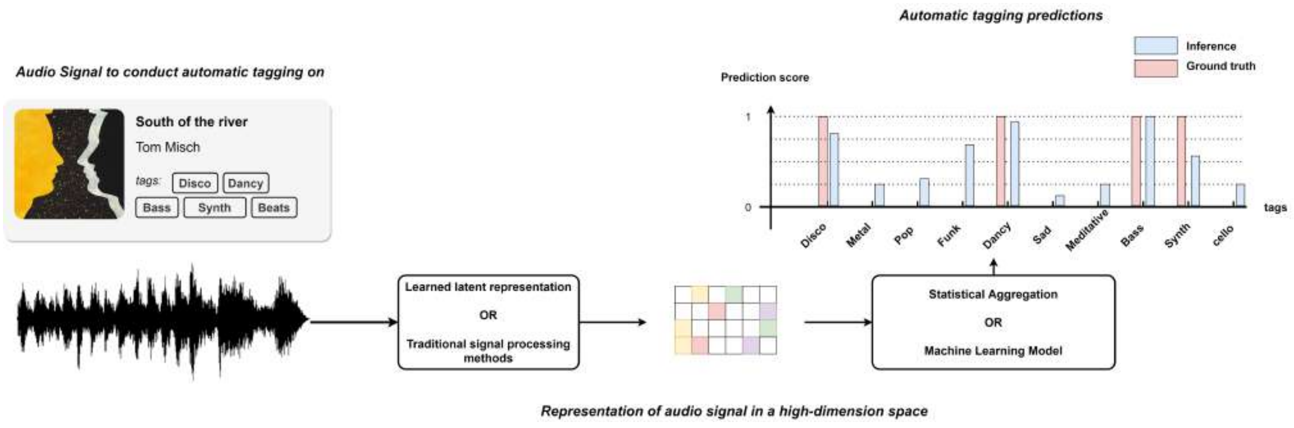

Audio Signal to conduct automatic tagging on
South of the river
Tom Misch
tags: Disco Dancy Bass Synth Beats
Learned latent representation OR Traditional signal processing methods
Representation of audio signal in a high-dimension space
Statistical Aggregation OR Machine Learning Model
Automatic tagging predictions
Inference
Ground truth
Prediction score
1
0
tags
Disco Metal Pop Funk Dancy Sad Meditative Bass Synth cello

Figure 2.13.: End to end music tagging

In the above figure, ideally inference predicted scores would be as close as possible to the ground truth scores. However, music tagging is tricky, per subjectivity of the task itself, which induces noisy datasets, and the complexity of attaining high accuracy on such a task.

Metrics for automatic music tagging

We want to evaluate the accuracy of the music tagging algorithm we implement before incorporating it into the recommendation pipeline to ensure the tag representations we use are coherent (in other words, we want our tagging algo to be the best as possible to provide meaningful tags to the recommender system). To do this, we use three metrics. First, recall the format of the model output versus the ground truth. The model output is a vector of n floating point values from 0 to 1 that represent the relevancy of the n tags. The ground truth value is a vector of same length with binary values of 0 and 1 (See figure 2.14)

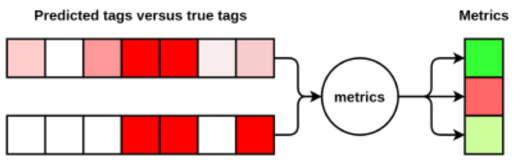


Figure 2.14.: Metric evaluation for music tagging

Area Under Receiver-Operator-Characteristic Curve The first metric is the area under the receiver operator characteristic curve. Consider first TP true positive predictions - where the prediction is close to 1 and the actual value was 1, FP false positives where the predicted value was close to 1 and the actual value close to 0, and their equivalents FN false negatives and TN true negatives. We define TPR true positive rate and $FPRthefalsepositiverateas$:

$$TPR = \frac{TP}{TP + FN} \quad \text{and} \quad FPR = \frac{FP}{FP + TN} \tag{2.5}$$

The receiver operator characteristic curve is defined as the couple (FPR,TPR) at one decision threshold for the model selection. For instance, if we choose the first decision threshold as 0.5 such that all values predicted above 0.5 are determined to be 1, this will yield an evaluation of (FPR, TPR) for this threshold. The ROC curve is the plotting of all these (FPR,TPR) tuples for many decision thresholds (figure 2.15):

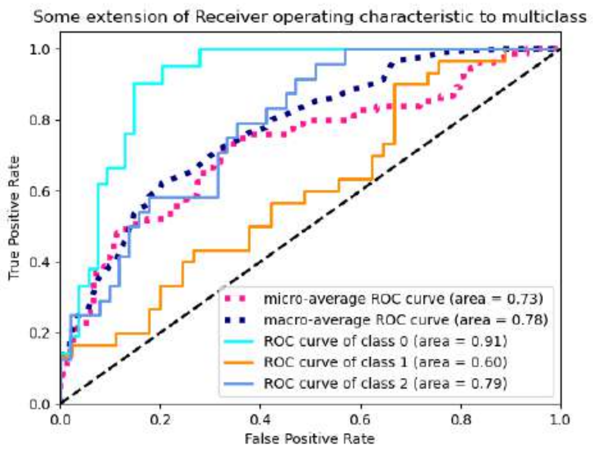


Figure 2.15.: ROC curve for a general ML application

The Area Under the ROC Curve (ROC-AUC) is one of the metrics used to determine multilabel multiclass classification performance as it is threshold invariant and shows a performance scale from 0 to 1 - A perfect classifier having 1 TPR for all FPRs.

Area under Precision-Recall Curve We define the Precision-Recall (PR) Curve as the curve (Pr,Re) Where Pr and Re are Precision and recall, defined as:

$$Pr = \frac{TP}{TP + TN} \quad \text{and} \quad Re = \frac{TP}{TP + FN} \tag{2.6}$$

And the same as before, the better the model, the higher the area under the (Pr,Re) tuples curve for various decision thresholds will be. PR-AUC can also be called average precision (AP) and denotes how many of the retrieved tags are relevant.

Cosine Similarity Finally, although PR-AUC and ROC-AUC are the main metrics for the music tagging task [8] [6] [7], [4] [37], we also use average cosine similarity over the predicted tags (as defined in equation 2.7) to have a raw evaluation of the similarity of the predictions - essentially, the better the model, the better the average cosine similarity.

signal and tag-based recommendations at Groover

How can music automatic tagging help at Groover? Two main contributions could be brought to the recommendation pipeline by implementing automatic music tagging:

Subgenre suggestion at signup: Currently, when Artists sign up to Groover to launch a campaign, they define themselves by a set of subgenres. Likewise, curators define their liked and disliked sugenres themselves. While it is important to maintain personnalized input from both parties, human inputting of genres is both subjective and noisy, which introduces high variance in the recommender system training.

- **Subjectivity** entails that an artist that is, by a given curator's standards, closer to indie rock than alternative rock, but by the artists' own standard, closer to alt than indie, will define himself as being an alt artist over an indie artist, while he could reasonably be called both. This will bias the recommender system towards curators with an affinity for alt more than indie for this curator, meaning that 1) they might miss out on great curators for them, and 2) they might be recommended some curators that do NOT like their music as much as they would expect, thus leading to higher rejection rates.
- **Noisiness** is caused by the fact that users and curators alike might define themselves by many subgenres to boost their "matchability", or might put only one subgenre because they do not really know how to define their music. Suggesting subgenres to artists at signup based on the analysis of their music by auto-tagging might reduce the variance of the distribution of number of subgenres and thus lead to a less noisy dataset for training the recommender system.

So, the first application of music automatic tagging is to present artists with machine-based recommendations regarding subgenres at signup based on their music to reduce the noisiness and subjectivity of the data fed to the recommender system, which intuitively would lead to better recommendations.

The other main contribution that can be made is contributing directly to the content-based features in the recommender system. By storing the musical tags of tracks for each artist and accepted and rejected tracks for each influencer, one can build an average/majority vote/softmax tag-score identity of the user (which can be either a curator or a band). At training time and inference time, these consitute valuable content features that are neither subjective nor noisy. They are not subjective because these "tag vectors" are learned representations from a model that is deterministic once trained, and they are not noisy because we can implement a fixed number of top-tags to consider for the tag representation, and if we do not even then the vocablary size of the tags is fixed.

With regards to what can be seen in figure 2.11, these tag representations of artist and curator tastes can also be used as another "ground truth booster", as with the cosine similarity ground truth in the aforementioned figure. Recall that these tags can be moods, themes, languages, instruments, which includes much more diversity of taste representation than a higher-level subgenre to determine taste. Cosine similarity in the case of tags would also take into account if a curator is in the habit of recommending more uplifting tracks, more energetic tracks, tracks with good bass lines. The new recommender system, integrating tag representations, would then look something like this (Figure 2.16):

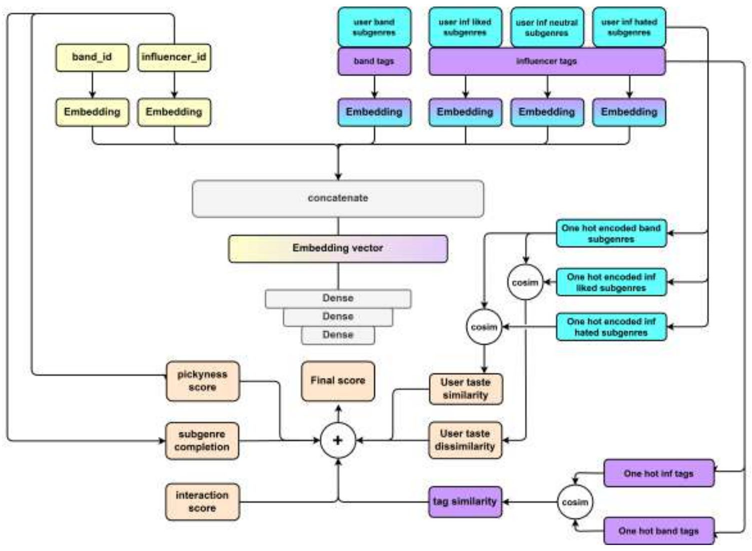


Figure 2.16.: Automatic music tagging contribution to the recommender system

Of course, these tagging representations can also be used as features in any other model that takes content-based representations as an input. So should the architecture change in the future, implementing these tagging models at scale will serve for the next version of the recommendation pipeline as well.

Measurable results of better recommendation

it is important to monitor the influence of modifications upon the recommendation pipeline through quantitative metrics : these metrics can be separated into two main categories : **business related categories** and **recommender system metrics**. In all this section, *preds* denotes model predictions (scores) and *gt* denotes ground truth scores:

Recommender system metrics

- **(R)MSE or Root Mean Squared Error** is a common metric for evaluating regression tasks. Since our task is to predict a continuous score between 0 and 1, this metric is adapted to the evaluating the quality of the score predictions.

$$RMSE(preds, gt) = \sqrt{\sum_i \frac{(pred_i - gt_i)^2}{n}} \tag{2.7}$$

  RMSE is robust against 0-valued ground truth and distribuion shift in predictive data, which makes it an adequate loss function for training our model.

- **MA(P)E** Mean Absolute (Percentage) Error makes more sense from an interpretability standpoint, as it is used to evaluate the percentile or absolute difference between ground truth score and score prediction. However, it is weak to 0-valued ground truth and thus is not used as our loss metric but rather as an interpretable metric.

$$MAPE(preds, gt) = \frac{1}{n}\sum_i |pred_i - gt_i| \tag{2.8}$$

- **Cosine similarity**, described previously, is used to describe the subgenre similarities between the user and the predicted influencer. Intuition gives that the higher the similarity between top predictions, the better the recommender system. We define Average Cosine Similarity at k (ACS_k) as the average cosine similarity between the artists' subgenres and the curators' liked subgenres for the top k ranked curators for a given artist (this is averaged over all recommendations in the test set)

$$ACS_k(preds, gt) = \frac{1}{nk}\sum_n \sum_{i<k} Cosim(subgenres_{artist}, subgenres^i_{curator}) \tag{2.9}$$

- **Personalisation index** is a measure of the variability of recommendations for different users. For instance, for a database of two users and six curators A,B,C,D,E,F, two recommendations (A,B,C) would yield a personnality index of 0. (A,B,C) and (A,E,F) would yield 0.66. The personalisation index is a measure of the capacity of a recommender system to adapt to different tastes.

- **Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain**: Both are a measure of the capacity of the model to target the correct curators for a given band in the top k recommendations, and NDCG specifically is a measure of their order of relevancy [16]. If NDCG is high at 5 and low at 10 then the 5 first recommendations are correctly ordered by order of relevance and recommendations 5-10 are poorly ordered. MAP does not take into account ordering, however. It is better to strike a balance between these two metrics to consider order of recommendation as well as hit rate.

Should the developed recommender system including audio be put into prediction before the end of the internship, these metrics will serve as a benchmark comparing the newer recommender system to the previous one.

Business metrics for recommender system monitoring The following business metrics are used to determine whether or not a recommender system is beneficial from the user point of view at Groover:

- **Number of recommendations tracks were sent too** In the first page looking at how many of the top k recommendations were selected by bands when building their campaigns is crucial to knowing whether the recommender system is doing a good job at finding relevant curators
- **Missed shot count**: The proportion of recommendations that were sent a track and refused it (predicted high score and truth zero score). The lower the Missed Shot Count (MSC) or "Sadness rate", the better the recommender system.
- **Time spent on platform and time spent looking at influencers**. The lower the time spent looking at influencers and the lower the time spent on the platform, the less the recommendations of the recommender system have caught the eye of the user
- **Feedback rating & number of active contacts**: On the platform, users are able to rate influencers for their feedbacks (e.g a 5 star feedback is a good inlfuencer match, a 2 star is a bad influencer match). This and the amount of influencers the bands are still in contact with after a campaign are good metrics to evaluate whether or not the recommender system is providing good recommendations *a posteriori*

2.3. Objectives for the internship, timelines

This section focuses on defining clear objectives for the internship as well as expected timelines from the point of view of Groover. The macroscopic and single most important objective of the internship is **Implementing a music tagging model into the current recommendation pipeline**, based on what the previous section covered. However, this task can be decomposed into multiple phases: **A research phase, a development phase, and a production - quantiy control phase**. Furthermore, the internship will also have subtasks to support the data team in other endavours, which will be briefly presented here.

2.3.1. Research on the state of the art of audio processing for and by music tagging applications

The first phase will be a state of the art research phase to establish all the necessary information about music auto tagging. What it is, what are the main models to conduct it, what is the history of the task, the contributions over the years, as well as the necessary preprocessing.

Audio preprocessing is an important bottleneck for most audio machine learning applications as the audio representation derived from the signal processing methods selected for the task will dictate the representation used as input for the model. Spectral resolution, sampling rate, sampling algorithms, Spectral Weighting, mel scales and many other potential representations will be thoroughly and comprehensively explored through this step in the research process, which is an important one and will fully exacerbate the need for a clear understanding of audio preprocessing techniques to deal with audio machine learning problems.

Audio Augmentation techniques will also be explored. The state of the art of audio models for tagging will be discussed, and as a last step the available data to train our models on will be explored : open source datasets, the data they contain, as well as their strengths and limitations, will be discussed.

Overall, the Research phase for this project is expected to last about 2 months, with all the previous steps being tackled in that order. These steps are highly interconnected however, so it will be necessary to tackle them in order of best comprehension, while conducting back-and-forth checks between the various aspects of audio tagging (e.g model choice is highly dependent on both audio preprocessing steps and available data)

2.3.2. R & D on implementation and benchmarking of audio preprocessing techniques and music tagging models

Once the state of the art for these models has been established, the available models and methods will be selected based on time constraints, material constraints, data constraints, and task appropriacy. Furthermore, the full preprocessing, data wrangling and selection pipeline as well as the implementation of the various models described in the previous section and their evaluation will be undertaken. The implementation of this code base as a modular pipeline is expected to take about 3 months, leading to the near end of the internship. This means :

- fully analyzing the preprocessing needs for the task based on the models we have selected based on the available data and preprocessing techniques
- building the preprocessing pipeline including all signal preprocessing steps
- implementing the relevant and selected models
- building and executing the training pipelines on all the models and the selected tags
- Evaluating the various models on the evaluation split to benchmark the best ones
- finetuning the best models for our task
- going back on the preprocessing, training and evaluation pipeline when needed.

2.3.3. At-scale implementation of the designed music tagging pipeline

With an estimated 1 month left in the internship, remains to productionize the code for the music tagging model at scale on the Groover artist database and all the stored music clips. This includes setting up the model to be trained. This will be a distributed task over most of the tech team but the model will still need to be monitored during this time. So, the overall Estimated timeline for the project is shown in the following figure (Figure 2.17)

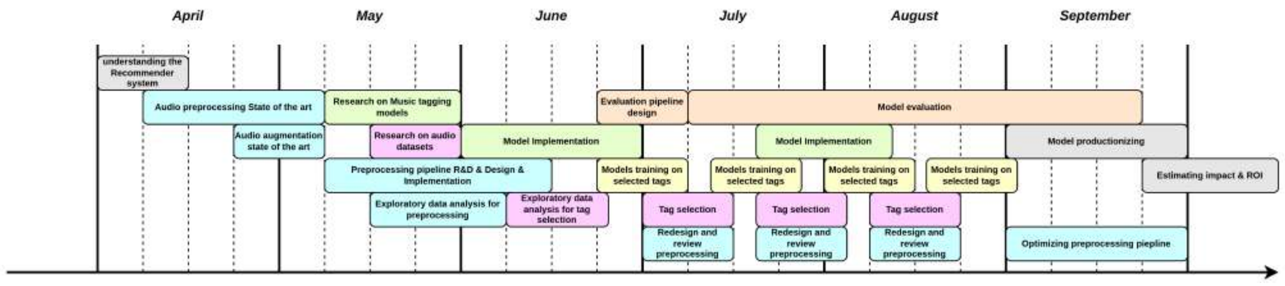


Figure 2.17.: Estimated timeline of the project

As can be seen, this will be far from a linear project. Mutliple reconsideration based on model performance, current model implementations, future model implementations, and model training will have to be undertaken, and back-and-forths between tasks will be inevitable. One of the objectives of the internship will also be to minimize these back and forths.

2.3.4. Developing the research culture at Groover

Another sub-objective for this internship will be to develop the research culture and presence at Groover, by writing articles and pursuing continuing research presentations throughout the internship. These are covered succintly is this section, as they are not a main part of the internship.

Paper talks: Audio and Deep Learning biweekly paper reviews

One of the goals of the internship was to start and maintain biweekly paper presentations on state of the art of machine learning and audio tagging. These presentations were cycled between the members of the data team and their goal was to maintain state of the art knowledge amongst the team members. These papers would also be rewritten as short vulgarization articles for medium, an open blogging platform where the machine learning community is quite developed. Below the links for the presentations that were created by myself:

- **Diffusion models presentation**
- **Codified audio language models presentation**
- **Contrastive learning of musical representations presentation**

Writing articles for a deeper research culture at Groover

To develop the online research presence of Groover, both in-depth articles and vulgarzation articles are written on state of the art evolutions in machine learning and the relevant tasks at groover (Recommender systems and Audio processing Machine Learning algorithms). A non-negligible part of the relationship was spent on writing an in-depth article on the role of the transformer architecture [44] on recent machine learning applications. Not only is it revolutionary and fundamental in today's machine learning landscape, but also relevant to some of the models presented in Chapter 4 4. The article can be found below:

Transformer in-depth article

Part I.

State of the art of research in audio music tagging

A word on the music tagging task and learned signal processing

This section will be focused on exploring Audio preprocessing techniques and machine learning models towards music tagging. This preface does not delve into detail into the specifics of the models explored later on but rather proposes a novel view of machine learning applied to audio as a learnable framework for more black-box audio processing blocks and serves as context with regards to the use of these music tagging models. Though it might be intuitive to consider these as two disctinct tasks of the same pipeline, we propose a different outlook : rather than exploiting learned representations of audio such as spectrograms through a learned model that only serves the purpose of "analyzing" these coefficients in a learned way, we propose that these machine learning models can be considered in and of themselves a learned representation extractor with regards to audio signals.

For instance, consider the canonical music tagging pipeline, which we can summarize here. usually, audio signals of fixed length are used to generate mel-spectrograms in a first preprocessing step. These melgrams are passed through a convolutional neural network which learns the tagging coefficients directly (*nota : As we will see later on, some audio models directly use raw audio to learn tagging coefficients*). A traditional Music tagging task would look like this (Figure 8.2)

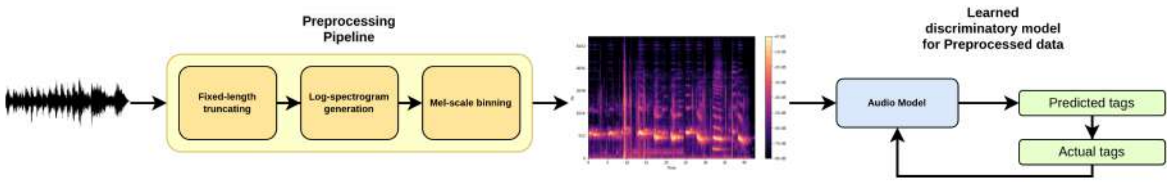


Figure 2.18.: traditional audio tagging task pipeline

In a traditional view, the preprocessing pipeline and the inference and learning pipeline are mostly disconnected. We propose a different view, that Audio models are learned latent representations of audio data based on the extracted data. Only the last few layers of the concerned deep learning models can be considered as classifiers used to discriminate from a learned representation of the audio which is computed in a full pipeline as opposed to before, two different pipelines (figure).

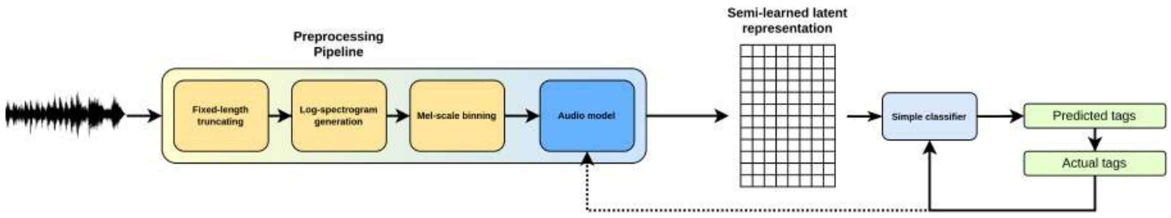


Figure 2.19.: new audio tagging task pipeline

Essentially, we intuite that past a certain point in training, contribution to the higher-level layers of the audio models are minimal compared to shifts in the weights of the simple classifiers which serve as latent representation discriminators based on the data extracted from the spectrograms (or other representations). This is corroborated by the value of pre-training (exhibited for instance in [4], [36]) in learning useful music representations which can be adapted for any downstrem task. By training directly on a downstream tagging-task we learn to generate a useful representation for the task of music tagging, which can eventually be generalized to other audio-oriented tasks.

An audio model appended after a traditional preprocessing pipeline can thus serve as what we now call a **latent representation extractor**, and generate what we now call **learned latent representations**. Much as spectrograms are frozen and deterministic representations of an audio signal relating to spectra and temporal evolution, Frozen audio models without their classifier heads are determinstic representations of an audio signal. We argue that the value of these learned representations can be taken out of the Music Information Retrieval domain to other audio domains, thus showing their use in other signal processing tasks (Much like machine learning has helped determinstic methods in fluid mechanics and Computational Fluid Dynamics for instance [42])

3. Audio preprocessing techniques

The basic raw audio data is in waveform format, or an array of floating samples with a given sampling rate (notated hereforth *sr*.) Preprocessing as a whole for audio includes various techniques which are applied to this raw audio before being fed through an inference model. This can include **segmenting, applying a transformation so the audio is in a new representation space, log-scaling, etc**

We do not consider data augmentation in the first part of this section as this is more relevant to model training. However, audio augmentation is an ensemble of techniques (reverb, delay, saturation, noise...) used to shift the input audio in the representation space to prevent overfitting in the mode during training. Preprocessing in and of itself is the pipeline applied before training which aids in giving the model an appropriate representation of the data.

3.1. Signal-based techniques

Signal-based techniques are defined as techniques applied directly to the raw audio signal (array of 1D floating point values) with the goal of making them more viable for machine learning tasks.

3.1.1. Filtering and filter banks

Filtering a signal a signal applies a convolution operation the weights of which reduce or augment the spectral composition of a signal at various frequencies [11], [7]. Common techniques for music classification tasks are **20Hz-20kHz band pass filters** [1] applied on all audio waveforms to conform to the human hearing range. For instance, when using data labels produced by humans, considering spectra components outside of the human hearing range does not make sense as they were not taken into account when producing labels.

Other, more case-specific techniques might be of use when considering specific applications . A few examples:

- Filtering for a given band of frequencies for instrument classification (for instance, filtering out lows for string instrument classification)
- Filtering out rumble frequencies (20-50Hz) and shimmer frequencies (19kHz - 20kHz) for percussive instrument classification
- Sound event classification (example Urban8K dataset [34] audio event classification task) where interesting frequencies and/or event spectral fingerprints might not lie within the human hearing range.

Librosa provides a bank of available filters at the documentation for librosa. Kapre also provides a filter bank layer for easier application during training and inference, as well as GPU-processed filtering.

3.1.2. Resampling

Sample rate represents the rate at which information is readily available within an audio signal. **The industry standard for sample rate is 44.1kHz**, which allows for a Nyquist frequency slightly above the human hearing threshold. However, it is possible to resample a signal to any given frequency.

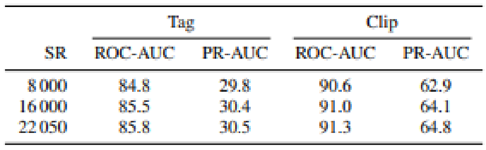
While this reduces space constraints and computation times for model inference (by effectively reducing the size of the input vector), it also is congruent to loss of information. For instance, resampling a 44.1kHz audio clip to 16kHz would lead to loss of more than half the data points.

In practice, the need to resample might arise when dealing with noisy real-world datasets. For instance, in **Rethinking CNN Models for Audio Classification** [30] The various datasets used to benchmark the model

[1]A note on musical recordings : Usually, mastering engineers already apply band-pass filtering to their master track: removing shimmer and rumble frees up space for other frequencies in a song which are more audible and agreeable to the human ear. However, this is not necessarily an industry standard, just good practice. So, while applying a band-pass might be redundant in some musical application cases, there usually is not 100% certainty that it will be.

on any given task have varying sample rates between themselves and between clips of the same dataset. The Urban8K dataset has sample rates varying between 16kHz and 44.1 kHz, and is uniformized to 22.5kHz. [2]

In terms of influence of sample rate on model performance, it is often minimal, and evolves as expected : a lower sample rate leads to lower performance due to the loss of information, be it on raw audio or transformed spectrograms with lower sample rates. (See table below taken from **Contrastive Learning of Musical Representations** [37]



	Tag		Clip	
SR	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC
8 000	84.8	29.8	90.6	62.9
16 000	85.5	30.4	91.0	64.1
22 050	85.8	30.5	91.3	64.8

Figure 3.1.: Performance evolution with sampling rate for contrastive learning of musical representations [37]

The performance hit is marginal, yet significant for industrial applications. So it is important to bear in mind sample rate and not dismiss it right away without at least checking for adjustment necessity.

Conceptual approaches to sample-rate conversion include: converting to an analog continuous signal, then re-sampling at the new rate, or calculating the values of the new samples directly from the old samples. The latter approach is more satisfactory, since it introduces less noise and distortion.[3] Two possible implementation methods are as follows:

If the ratio of the two sample rates is (or can be approximated by) a fixed rational number L/M: generate an intermediate signal by inserting L 1 zeros between each of the original samples. Low-pass filter this signal at half of the lower of the two rates. Select every M-th sample from the filtered output, to obtain the result. [29] Treat the samples as geometric points and create any needed new points by interpolation. Choosing an interpolation method is a trade-off between implementation complexity and conversion quality. Commonly used is the windowed sinc function for audio. The two methods are mathematically identical: picking an interpolation function in the second scheme is equivalent to picking the impulse response of the filter in the first scheme. Linear interpolation is equivalent to a triangular impulse response; windowed sinc approximates a brick-wall filter (it approaches the desirable "brick wall" filter as the number of points increase). The length of the impulse response of the filter in method 1 corresponds to the number of points used in interpolation in method 2.

Method 2 will work in more general cases, e.g. where the ratio of sample rates is not rational, or two real-time streams must be accommodated, or the sample rates are time-varying.

Both kapre [9] and librosa [27] provide resampling functionality for audio signals. The default filtering window used for the resampling interpolation algos used by both kapre and librosa is a kaiser window, which can be defined as :

$$w_0(x) = \begin{cases} \frac{1}{L}\frac{I_0 \pi\alpha\sqrt{1-(2x/L)^2}}{I_0\pi\alpha}, |x| < L/2 \\ 0, |x| > L/2 \end{cases} \tag{3.1}$$

where:

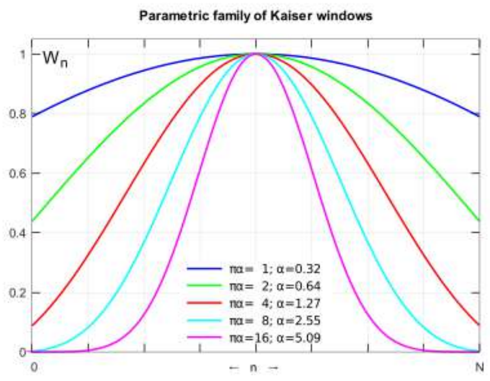
- I_0 is the zeroth-order modified Bessel function of the first kind,
- L is the window duration, and
- α is a non-negative real number that determines the shape of the window. In the frequency domain, it determines the trade-off between main-lobe width and side lobe level, which is a central decision in window design. Sometimes the Kaiser window is parametrized by β, where $\beta = \pi\alpha$.

The shape of which can be seen figure 3.2a. The fourier transform of this window is given by:

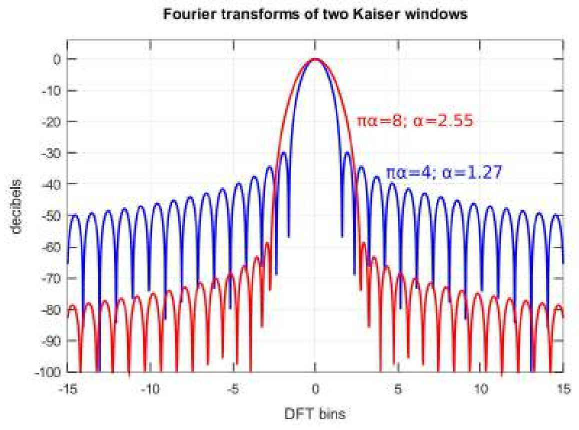
[2]Note on musical applications: As there is an industry standard in place on streaming platforms, it is safe to say that most clips will have at least a sample rate of 44.1. However, high quality audio (96kHz is possible, as well as 122kHz) also exists and so a check is necessary to ensure fixed-length input to the model even if no space-saving related resampling is considered.

$$W_0(f) = \frac{\sin(\sqrt{(\pi L f)^2 - (\pi\alpha)^2})}{I_0 \pi\alpha \sqrt{(\pi L f)^2 - (\pi alpha)^2}} \tag{3.2}$$

The shape of which is shown figure 3.2b with various parameters:



(a) Kaiser windows for various parameters



(b) Kaiser window spectra for various parameters

And the equivalent window function for a discrete signal (digital audio signal processing) is given by :

$$w(n) = L w_0(L/N(n - N/2)) \tag{3.3}$$

having $N+1$ the length of the window. both kapre, librosa, and torchaudio [55] use fast and best variations of the kaiser window to conduct resamplig. Interpolation and decimation to obtain the target sample rate are both conducted using the discrete kaiser window and since the default resampling algorithm is used in the entirety of state of the art, we assume that use of the default kaiser window in this work is appropriate and does not require further investigation.

3.1.3. Mono

Mono is the operation of time-averaging the signals of the two channels of stereo signals (L-R). This reduces the input dimensionality of the signal, at the cost of music-relevant information. Indeed, stereo spacing is often an artistic choice and is relevant when humans are listening to music. So, it follows that reducing input to mono distances the model from the artistic vision of the mixing engineer, which is integral to track characteristics.

Canonical music classification datasets adopt various approaches to this issue, as for instance the **GTZAN [41] and UrbanSound8K [34]** datasets provide mono samples. The FMA dataset [10] provides Stereo clips. Both for loading and other operations, Librosa and Kapre provide stereo support. [52] does not provide emphasis on the use of either mono or stereo.

Per the difficulty of finding any good practice recommendation of mono vs stereo input clips in the current literature, it can be said that the choice is rather empirical, or even model-based. If the model used for the classification task does not support multichannel input, then mono is adopted. However, intuition dictates that when available, stereo should be used, as perhaps spatial features can be learned by the model. At any rate, librosa proposes a handy mono function should the need arise to make a track mono for analysis, though kapre does not:

```
import librosa
y = librosa.to_mono(y)
```

3.1.4. Smoothing / Time averaging

Smoothing, or time-averaging a signal, is a preprocessing technique used to smooth out the envelope of an audio signal, and thus generate a new vector with the average envelope of the signal over the given time frame. This is mostly useful for onset detection though, and has little application in tagging tasks, bar its implication in tempo and other rythmic features detection (these modules are built into librosa as will be seen later on).

Obviously, high-frequency details are lost when smoothing out the signal.

3.2. Spectral techniques

The general purpose of spectrograms is described in Section 3.2.1. They are 2D representations of an audio signal which show the energy distribution of the signal along both the time and frequency axis, making them biologically relevant feature maps of the signal.

Their setup can vary, and modern classification tasks rarely use pure STFTs as input. More performant models have been elaborated by using log-frequency spectrograms, log-magnitude spectrograms, MFCCs and Mel-spectrograms.

3.2.1. Discrete Fourier Transform & Short Time Fourier Transform

Discrete Fourier transform and short time Fourier transform are the respective time-unaware and time-aware analogs of the Fourier transform. this transformation allows for computation of the power distribution of the given sample over the sound frequency spectrum.

Discrete Fourier Transform (DFT) DFT is used to extract spectral representations from a discrete digital signal. recall the expression of the discrete fourier transform of a signal y into its spectrum Y:

$$Y_k = \sum_{n=0}^{N} y_n e^{\frac{-2i\pi nk}{N}} \tag{3.4}$$

based on state of the art, the DTFT is very rarely used in music classification tasks (no papers mention it). This makes sense intuitively as DFT does not represent the evolution of the spectral components of the signal over time, but only over the globaility of the signal, and il also non-discriminant towards unsignificative values. It is relatively high-dimensional data for not much information. The following figure shows the DFT for one of the tracks in the FMA dataset [10] (Figure 3.3) :

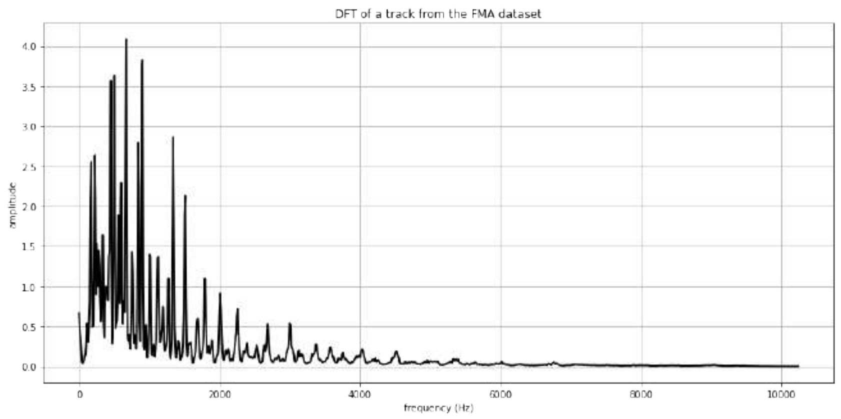


Figure 3.3.: DFT of track from the FMA dataset

Short time Fourier transform (STFT) The rawest kind of Spectrogram, with a linear frequency scale and linear amplitude scale. it's a 2D representation of the signal which is used as input for convolutional neural networks towards music classification tasks. However, it has since been dethroned by the Mel-spectrograms due to the more biologically-plausible representation it provides.

Its strengths stem from the fact that it is a representation of the evolution of the spectral components of the signal over time. Looking at the expression for DFT, the spectral components for the signal are computed over the whole signal. To paliate this, Gabor ([12]) proposes computing the DFT of the whole audio over chunks of the signal to get timewise evolution of the spectral components : This can be achieved through multiplying the signal by a window function centered around various points of the signal. To mitigate artifacts created by analyzing directly-adjacent chunks of the signal, the chunks of the signal are taken as overlapping. The following is the expression of the STFT for a discrete signal with a window function w

$$Y(k,m) = \sum_{n=0}^{N-1} w(n)y(n+mH)e^{\frac{-2i\pi kn}{N}} \tag{3.5}$$

Where $y(m + H : N - 1 + m + H)$ is a segment of y, H is the hop length, in samples (by default in python audio libraries, half of the $N_f ft$), w is the window function. m is the frame index, k is the frequency bin index.

STFTs are often computationally expensive as they provide high dimension 2D representations (For instance, the following STFT spectrogram computed for the same track as figure 3.3 (figure 3.4) with a $n_f ft$ of 2048, a H of 1024 and a Window length of 1024 samples, with a $16kHz$ sample rate, is og shape (1025, 9258), which yields 9.5M samples for 3 minutes of audio.

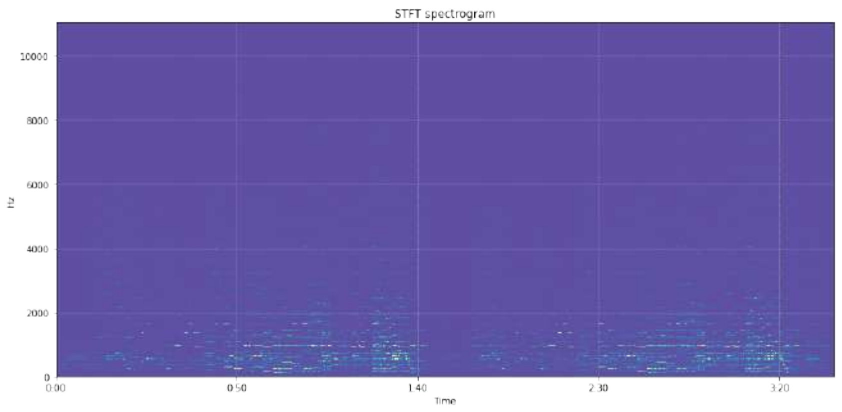


Figure 3.4.: STFT of track from the FMA dataset

The number of FFT bins dictates the spectral resolution, at the expense of temporal resolution. To alleviate this trade-off and prevent artifacts in the fft generation, the use of windows is necessary: A window is a function which multiplies the convolution boundary to smooth out the obtained spectra. Some options for window functions are the following:

- Hanning:

$$w_0(x) = \begin{cases} 1/L\cos^2(\frac{\pi x}{L}), |x| < L/2 \\ 0, |x| > L/2 \end{cases} \tag{3.6}$$

- Hamming
- Black

The default used by many studies and notably the **Librosa** module is the hanning window, which is sufficient for most audio processing applications. The stft function is readily available in librosa and kapre. [3]

[52],[7], [8] [6], [30] All mention that MFCCs outperform STFTs and melgrams outperform MFCCs by a respectable margin. STFTs are seldom used in recent studies per their large size and usually lesser performance.

[3]Note : the STFT is a dual representation : magnitude and phase. Often in classification tasks the phase is neglected (especially when dealing with convolutional neural networks). However, in tasks relative to speech and music generation, phase is a non-negligible component of the signal (eg : in speech generation tasks where phase is neglected, the generated vocals come out flat and robotic).

3.2.2. Mel Frequency Cepstrum Coefficients (MFCC)

Mel Frequency cepstrum coefficients (MFCCs) are a set of features which can be used to describe the overall spectral envelope of a signal. They are interesting in terms of audio classification tasks as they provide a low-dimension representation of the signal. They are based on the mel frequency scale which mimics human perception of frequencies: the following figure (Figure 3.5) shows the mapping function from Hz to mels [2]:

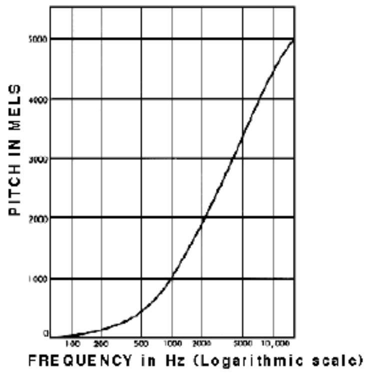


Figure 3.5.: Mel Scale for Audio Frequencies

The calculation of the pitch in mels based on the frequency f is defined as:

$$f_{mel} = 1000\frac{\log_{10}(1 + f/1000)}{\log_{10}(2)} \tag{3.7}$$

The coefficients are obtained by taking the outputs of N_f log-scale triangular filers of a signal into Mel-frequencies and applying a discrete cosine transform. The formula for the cepstral coefficients is given below:

$$c_i = \sum_{n=1}^{N_f} S_n \cos\left(i(n-0.5)\frac{\pi}{N_f}\right), i \in [0...L] \tag{3.8}$$

Where L cepstral coefficients are wanted, and S_n is the log-energy output of the n^{th} triangular filter. The MFCC calculation using librosa [27] for the track in figures 3.4 and 3.3 is given in the following figure (3.6)

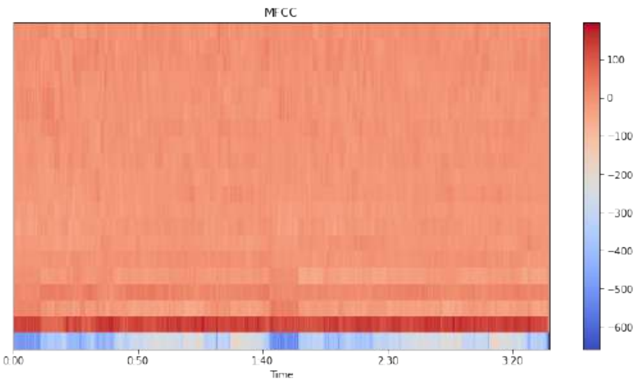


Figure 3.6.: MFCCs for the track from the FMA dataset

in [6], auto music tagging is conducted using among others MFCCs as features. MFCCs have since been outperformed by Mel spectrograms in all music classification tasks, as we will see in the next section.

3.2.3. Mel-spectrograms

Mel-spectrograms are a combination of STFT and MFCC, in that they are obtained by averaging the STFT values over mel-bins, which yields a lower-dimension representation which is more adapted to human perception of sound. Melgrams are the default feature input for convolutional classification models, especially on tagging. [7] [8] [6] [30] All use melgrams as their input feature, and more specifically [6] [8] [7] [48] use them as part of a tagging task, which is relevant to our use case. The following figure (Figure 3.7 shows the mel-spectrogram for the track used in all other figures:

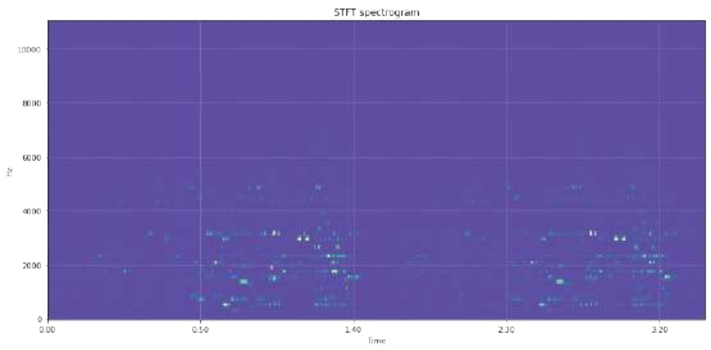


Figure 3.7.: Melgram of track from the FMA dataset

For instance, this melgram of the same track, computed using the same FFT hyperparameters as in 3.4 yields a feature map of shape (128,9285), which represents a 90% reduction in input size dimension (much better for storage, inference time). Furthermore, Melgrams have been shown to outperform or reach the same performance as STFTs in [7] (see figure 3.8):

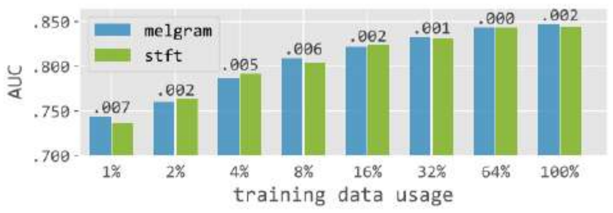


Figure 3.8.: Performance of mel spectrograms versus STFT on tagging tasks versus training data

And in [6] as seen in the following table:

Method	Input	ROC-AUC
FCN-3,	mel-spectrogram	.852
FCN-4,	mel-spectrogram	.894
FCN-5,	mel-spectrogram	.890
FCN-4,	STFT	.846
FCN-4,	MFCC	.862

Table 3.1.: Input performance on music tagging as seen in [6]

Implementations of melgrams are readily available in both Librosa and Kapre. Per their ease of use, low dimensionality and high performance, mel-spectrograms are often considered to be the benchmark for industry applications in most music classification tasks and specifically music tagging tasks [4]. However, we scrutinize some

[4]Furthermore, it is possible to generate stacks of spectrograms and melgrams as inputs by producing various spectrograms with varying hop widths/window sizes. This allows to capture features at various time and frequency scales, which can be beneficial to the model learning them. However, no studies compare the results obtained for such variations.

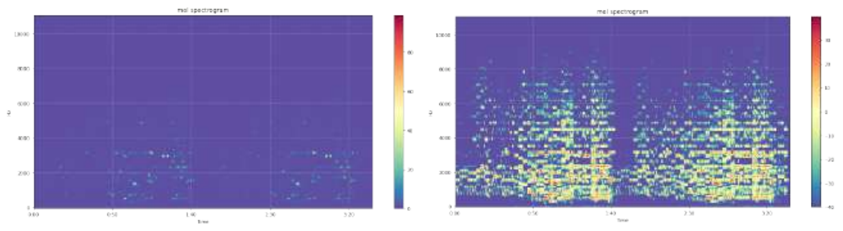
additional prepcrocessing techniques in Annex A.2 (**Constant Q melgrams, Chroma Variants, Spectral Envelopes, Tempograms**).

3.3. Scaling techniques

It is well known that data scaling is essential for good performance of most machine learning models. This prevents gradient explosion for skewed data distributions. In simple classification systems, all features are independently scaled to reach unit normal distribution. In computer vision, image values are scaled for R,G,B channels. For audio preprocessing, the data for STFTs can be scaled using log-magnitude scaling:

3.3.1. log-melgrams

Due to our non-linear perception of Loudness [43], it makes more sense from a biological standpoint to represent loudness in 2D time-frequency inputs as decibel values, i.e taking the log of the modulus of the STFT transform. Here are two Spectrograms representing the same signal, one of them being log-scaled and the other one not (figure 3.9):



(a) Raw mel spectrogram (b) Log-scaled mel spectrogram

Figure 3.9.: raw and log-scaled melspectrogram

As can be seen, the magnitude scale is highly compressed, which usually represents a significant gain in performance for a machine learning model, especially regarding image data. The reduced scale prevents gradient explosion. The following shows the distribution of the binned magnitudes for the non-log scaled stft and for a log-scaled stft (Figure 3.10) taken from [7]. These histograms are averaged over 100 songs from the million song dataset.

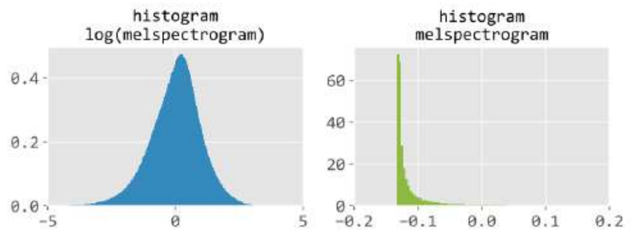


Figure 3.10.: Log-scaled and non-log scaled bins for the melgrams in [7]

In terms of performance on audio tagging tasks, [7] shows that log-scaled melgrams always outperform regular melgrams, the same goes for log-scaled STFTs versus standard STFTs (Figure 3.11).

All studies mentioning melgrams use log-scaled melgrams and some even explicitly state that log-melgrams are the norm. There is no upside to using linear scaled melgrams at all regarding audio classification. Furthermore, it is also possible to apply α-log scaling to the magnitude of the melgrams, which is obtained by applying the following transformation :

$$X \rightarrow \log(X + \alpha) \tag{3.9}$$

the exploration of the epsilon parameter however, is non-existent. Studies arbitrarily apply it as a large or small number, or not at all. We decide to keep it as 0. Librosa and Kapre both implement handy default

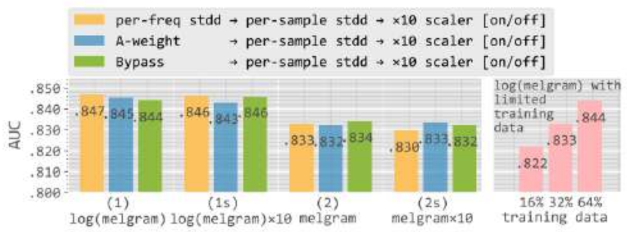


Figure 3.11.: performance of non-scaled STFTs and melgrams vs scaled STFTs and melgrams on music tagging tasks [7]

behaviours regarding this log-scaling, which is set to true by default. Furthermore, it is possible to apply 0-centering through the appropriate function parameters.

3.3.2. Frequency scaling

Per human's non-linear perception of frequencies (we naturally perceive higher frequencies as louder, frequency weighting can provide marginal performance gains when looking at classification models. There are two main techniques [7]

- **Scaling per frequency band** : Also known as spectral whitening. Each frequency band is scaled using mean and standard deviation in the frequency band. This yields a flat spectrum which resembles that of white noise (hence the name).
- **A-weighting** International systems exist to weigh spectra magnitude values as a function of the frequency for various applications. : A-weighting attempts to represent human perception of sound (higher frequencies are felt as louder) C-weighting is for industrial applications and noise regulation laws, and dips off in the very low end (rumble) and very high end (shimmer) Z-weighting is a flat filter (See figure 3.12) [47].

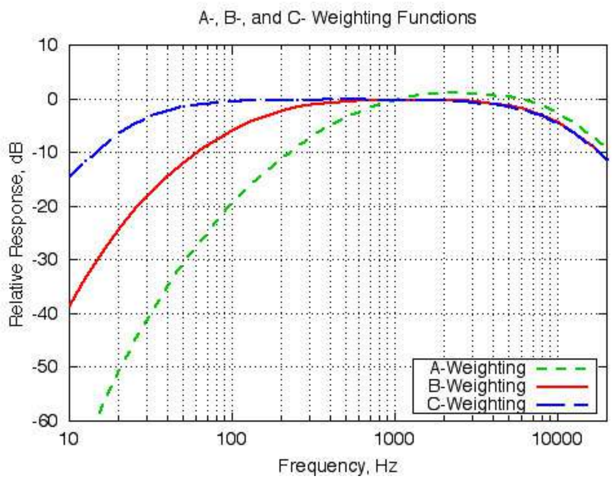


Figure 3.12.: International standards for A,C,Z weighting [47]

Such filters can be applied to the signal as part of preprocessing. However, the performance gain depends on the context and is often marginal. Figure 3.11 shows ROC-AUC performance in [7] for the two described techniques as well as bypassing frequency weighting. There is no true advantage in the case of music tagging to applying frequency weighting to the signals.

3.4. Splitting

3.4.1. Fixed length input splitting

Often, the input to audio classification models is in the range of the 5-10 seconds [7], [52] length (which still represents around 200000 samples). Some musical datasets have gone up to 30s ([41]), but few feature whole length songs.

It can be difficult and not necessarily task-relevant for models to capture relevant features at whole song-levels. This is why a common preprocessing technique is to split the audio clip into fixed input segments which are then voted upon or average (in what is called multi instance learning) to determine the class of the whole sample or can be stacked to create a multi-dimensional vector (think RGB channels for computer vision).

Additionally, splitting the input into different sample sizes (2s, 10s and 30s for instance) can be beneficial towards capturing audio features at different time scales. This is effectively increasing the receptive field of the network.

It is shown in [52] that the maximum length for inputs in current music classification models is 30s, with many being around 3-5s [6], [8], [32], [22], [49]. Splitting adds another advantage, which is that of data augmentation : one 10-second long song yields 5*44100 training samples, which makes for huge training potential with minimal labeling requirements (it does require high storage space though).[5][6]

3.4.2. Padding

Obviously, all audio clips are not of perfect multiple length regarding the desired input length. Zero-padding to attain fixed input length is a common technique in other domains as well (sequence prediction, computer vision, NLP...) and so we do not elaborate on it here, as its effects are similar to those in those other domains and are presumed to be inoffensive.

3.5. Audio augmentation techniques

As mentioned previously, audio augmentation techniques are a set of techniques used to shift the audio sample in the representation space : Audio augmentation, a subset of techniques of general data augmentation, changes the input slightly when training the model to prevent the model learning on the exact distribution of the training set - acting as a regularization technique. Here we summarily discuss audio augmentation techniques and their use in state of the art.

Random cropping and filtering have been discussed previously in the context of pure audio processing. They can also, however, be used as audio augmentation techniques [52]. By random cropping audio inputs, the representation of a given track changes every time it is fed to the model. Similarly, using high-pass and lowpass filters on a track with varying probabilities acts as a disruptor of the original signal (different features are seen. This is used in [36] and [48] using randomized filter cutoffs.

Delay and reverb Delay and reverb are techniques used to modify the acoustic space of the signal. Delay is applied my delaying the signal by a randomly generated delay time (in ms) and adding the delayed signal multiplied by a reductive factor to the original signal. Reverb is generated using impulse response's room size, reverbation and damping factor, drawn from a uniform distribution. Again, these are used in [36] [24] [48]

Polarity inversion and pitch shifting relate to the frequency characteristics of the waveform. Polarity inversion of the waveform is obtained by multiplying it by -1, and pitch shifting is applied by common algorithms provided by librosa, torch augmentations, audiomentations, etc. Usually, pitch shifting occurs by increments of semitones in the 12-tone equal temperament range within 1 octave of the original.

[5] An extension of this is to split the song into relevant segments (for instance, verse chorus and bridge) as from an artistic standpoint it makes sense to distinguish them, so to does it from a tagging standpoint.

[6] Removing the beginning of songs can be an added benefit (approx 15-20s) to remove intro parts that do not reflect on the true character of the song (spoken intros, soft intros to punchy songs, etc)

Noise and Gain Automation reduce the prominence of the original signal by either applying noise to it (which "blurs out" small range audio features to the human ear) or randomly reducing the gain, or volume, of the track over the course of the signal. Again, these are applied in [36] and [48]

[48] Shows by an ablation study that even without student training, audio augmentations improve the performance of the model by reducing overfitting. Generally, these augmentations are applied stochastically in series (stacked one upon the other), meaning that not all signals going through the pipeline will be subject to the same modifications. We consider using audio augmentations later on in the project when all base models are trained on a non-augmented dataset.

4. Music tagging models

This section focuses on exploring the architectures and results of the various prominent models in state of the art music tagging. Each models' architecture and strengths will be overviewed, as well as its performance on the dataset used in the relevant papers' experiments. A last subsection will be reserved for model comparison on a standardized split as seen in [51].

Papers from literature use three input representation lengths : 3s, 15s, and 30s. Though input preprocessing and length are standardized in our case, preprocessing steps vary slightly from paper to paper, so these preprocessing steps and hyperparameters will also be considered in this section.

4.1. Fully Convolutional Network

This architecture was implemented in [6], and was the first use of convolutional neural networks for end-to-end music automatic tagging. Each convolution layer of size $H \times W \times D$ learns D features of $H \times W$, where H and W refer to the height and the width of the learned kernels respectively. The kernel size determines the maximum size of a component it can precisely capture.

Base architecture

The general stucture of the architecture proposed in [6] is a stack of convolutional kernels with a dense head which is referred to as the classification head. The convolutional front-end acts as a feature extractor while the dense fully-connected back-end acts as a discriminator head. [6] proposes 5 variants of the base network (**FCN** - fully convolutional network): FCN 3-4-5-6-7 referring to the number of convolutional blocks in the front-end of the network.

Each convolutional block is comprised of a convolutional layer, a batch Normalization layer and a Max Pooling layer to allow for feature size reduction throughout the process. based on performance on the MagnaTagATune dataset ([19]), FCN-4 is selected as the final model. The architecture for this model is shown below (Figure [**FCN**]):

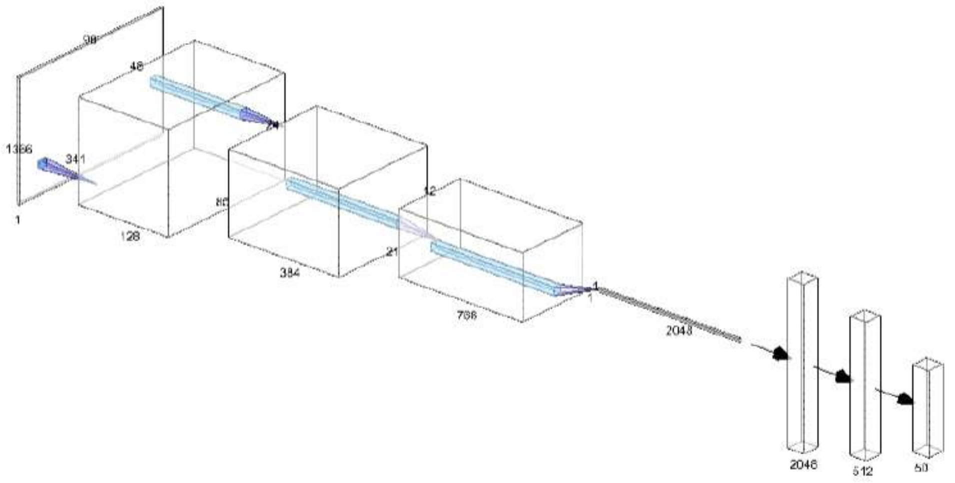


Figure 4.1.: Architecture of the FCN model

Preprocessing

Multiple experiments were run in [6], including STFTs, MFCCs and Mel-Spectrograms:

> "A pilot experiment demonstrated similar performances with 12 and 16 kHz sampling rates and melbins of 96 and 128 respectively. As a result, the audio in both datasets was trimmed as 29.1s clips (the shortest signal in the dataset) and was downsampled to 12 kHz. The hop size was fixed at 256 samples (21 ms) during timefrequency transformation, yielding 1,366 frames in total. STFT was performed using 256-point FFT while the number of mel-bands was set as 96. For each frame, 30 MFCCs and their first and second derivatives were computed and concatenated."[6]

This yields an input shape of 1366×96 for input mel spectrograms

Performance

The models for this paper were evaluated against previous works using the MagnatagATune dataset, against different input representations on the magnatagatune dataset, and against all the k-variants of the different FCN-k architectures on the million-song datasets. The obtained results are shown in the tables below 4.2:

Methods	**ROC-AUC**
The proposed system, FCN-4	**0.894**
2015, Bag of features and RBM	0.888
2014, 1D convolutions	0.882
2014, Transferred learning	0.88
2012, Multi-scale approach	0.898
2011, Pooling MFCC	0.861

(a) Results of [6] compared to past work

Methods	**Representation**	**ROC-AUC**
FCN-3,	mel-spectrogram	.852
FCN-4,	mel-spectrogram	**.894**
FCN-5,	mel-spectrogram	.890
FCN-4,	STFT	.846
FCN-4,	MFCC	.862

(b) results of [6] varying representation and model

Methods	**Representation**	**ROC-AUC**
FCN-3,	mel-spectrogram	.786
FCN-4,	—	.808
FCN-5,	—	.848
FCN-6,	—	**.851**
FCN-7,	—	.845

(c) Results of [6] on the million song dataset

Table 4.1.: Results from [6]

4.2. CRNN

[8] Introduces A Convolutional Recurrent Neural Network (CRNN) for audio music tagging with a convolutional front-end acting as a feature extractor and a Recurrent Neural Network acting as a temporal feature aggregator. The intuition here is that since music is an inherently sequential media (a sequence of chords, notes, lyrics, structures, etc.) It is beneficial to include a temporal learning structure to the successful convolutional one.

Base Architecture

[8] Proposes a structure with the same convolutional front-end as [6], with less temporal pooling to keep a set of temporal features at the end of the front-end. A two-layer recurrent neural network (RNN) is then stacked on top with 32 units per layer to extract temporal features (As shown in figure 4.2) [1]

[1]The two RNN layers are represented as Flat blocks along the temporal axis in the above figure

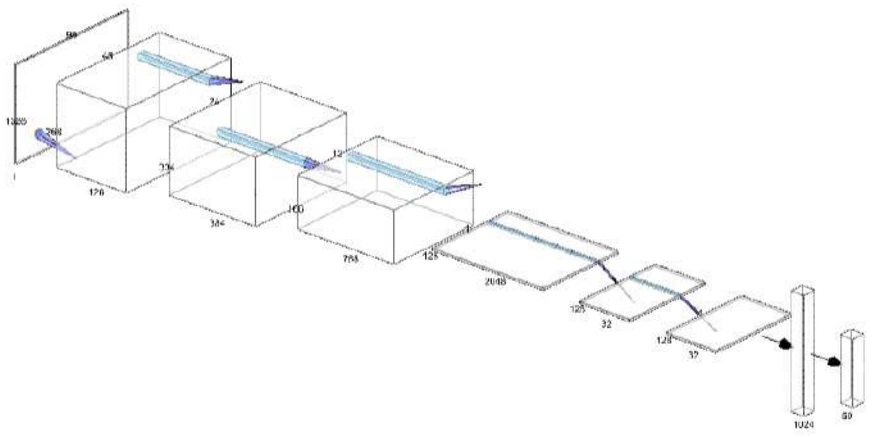


Figure 4.2.: Architecture of the CRNN model

Results

Input representations were only log-Mel-Spectrograms as their effectiveness has been demonstrated in [6]. The preprocessing hyperparameters are kept the same from the previous study. CRNN is compared to 3 other models (k2c1, k1c2, k2c2), which respectively use time-convolution filters, frequency-convolution filters, and rectangular filters (shown in figure 4.3):

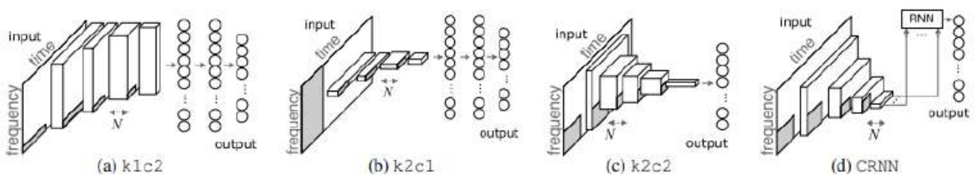


Figure 4.3.: Models implemented in [8]

Results are evaluated on the million-song dataset for all 4 models and compared to SOTA (state of the art). All models are also evaluated using different parameters scalings (less filter depth) and compared (Shown in figure 4.4):

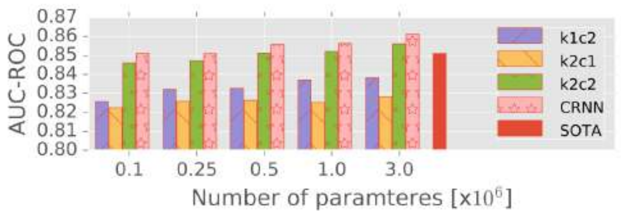


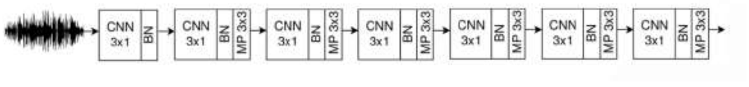
Figure 4.4.: Results from [8]

CRNN effectively beats SOTA on music tagging tasks and represents the last used model with an input length of 30s in literature.

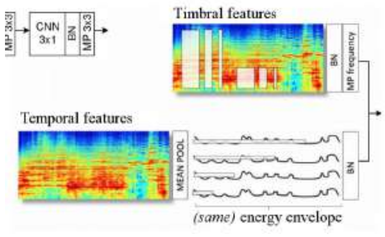
4.3. MusiCNN

Architecture

MusiCNN, implemented in [32], aims to take advantage of domain knowledge to improve music tagging performance. MusiCNN is adaptable to two input representations : Melgrams and raw audio. To achieve this, it is comprised of a common back-end for both architectures and a convolutional architecture front-end : in 1D for raw audio and 2D as previously for 2D melgrams. These two front-ends can be seen in the following figures (??)



(a) Front end for pure audio in [32]



(b) Front end for melgrams in [32]

The front-end for raw audio is a series of 1D convolutional filters used to extract relevant features from pure audio. The front-end for Spectrograms, on the other hand, takes from domain knowledge of musical analysis that music is a combination of temporal and timbral features [32], and thus is designed with this in mind as two branches concatenated, one a series of vertical (timbral) filters and the other a series of horizontal (temporal) features of varying lengths (to capture features at different scales). It is shown that having multiple filter scales is more beneficial than having one. In both representation architectures, the input from the front-end is fed into a common back-end, shown in figure 4.6

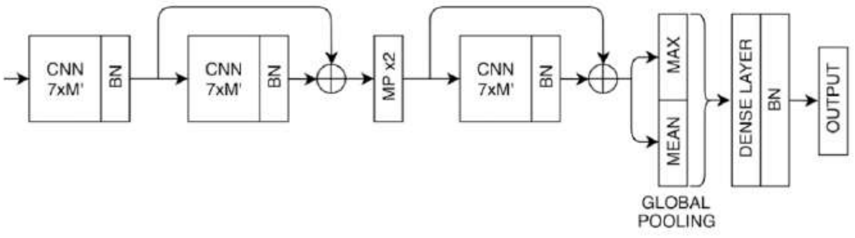


Figure 4.6.: MusicNN backend [32]

Results

Input representations are taken as chunk-level samples of songs (3s input length). The audio is resampled to 16kHz and the parameters for melgram computing is 512 fft bins with a hop value of 256.

Multiple experiments are run for this paper, including:

- experiments on a proprietary 1.2M song dataset on which the raw audio model performs better than the spectrogram one (table 4.2a)
- experiments on the million song dataset comparing to previous work including previously mentioned models here (4.2b)
- results on the MTAT dataset [19] compared to previous work, where Musicnn comes close to previous results with high reduction in the size of the model (4.2c)

MusicNN demonstrates that domain knowledge in both music and signal processing is a valuable insight when conceptualizing model architectures for audio tasks as it can enhance performance while greatly reducing model size. It also demonstrates that using 128 mel bins instead of 96 is preferable for performance.

1.2M-songs	**trained on**	**ROC-AUC**	**PR-AUC**
Baseline	1.2M	91.61%	54.27%
Waveform	1M	**92.50%**	**61.20%**
Spectrogram	1M	92.17%	59.92%
Waveform	500k	91.16%	56.42%
Spectrogram	500k	91.61%	58.18%
Waveform	100k	90.27%	52.76%
Spectrogram	100k	90.14%	52.67%

(a) Results of [32] on their proprietary dataset

MSD	**ROC**	**PR**	**#params**
Models			
Waveform	87.41	28.53	5.3M
SampleCNN [22]	88.12	-	2.4M
SampleCNN multi level and scale [21]	88.42	-	-
Spectrogram (ours)	88.75	31.2	5.9M
CRNN [8]	86.2	-	3M
Multi-level and multiscale [25]	88.78	3M	-

(b) Results of [32] compared to past work on the million song datasset

MTAT	**ROC**	**PR**	**#**
Waveform	AUC	AUC	param
SampleCNN [22]	90.55	-	2.4M
SampleCNN (128mels) [22]	88.56	34.38	2.4M
MusiCNN raw audio	84.87	-	-
MusiCNN raw audio (128mels)	85.58	29.59	194k

MTAT	**ROC**	**PR**	**#**
Spectrogram	AUC	AUC	param
Fully convolutional [6]	89.4	-	22M
Fully convolutional (128mels) [6]	89.99	37.6	450k
MusiCNN	89.30	-	191k
MusiCNN (128mels)	89.07	34.92	220k

(c) results of [32] on the MTAT dataset

Table 4.2.: Results from [22]

4.4. Sample-level CNN

SampleCNN [22] and SampleCNN+Squeeze [17] propose a novel approach to music tagging by 1) considering raw audio waveforms as their input (requires no preprocessing), and 2) using strided convolutional filters with very high granularity to decompose audio signals at into high frequency features. This allows for learning low-dimensional filters to apply onto the audio signal to extract relevant features.

4.4.1. Base model

Architecture

The base model for sampleCNN is a 1-D convolutional neural network using chunk-level audio lengths (2.6s) as an input but using frame-level convolutional filters (of size 3) on the audio signal. In the terminology of this paper, a frame is comprised of 3 samples. The architecture for samplecnn is shown in figure 4.7 [22]:

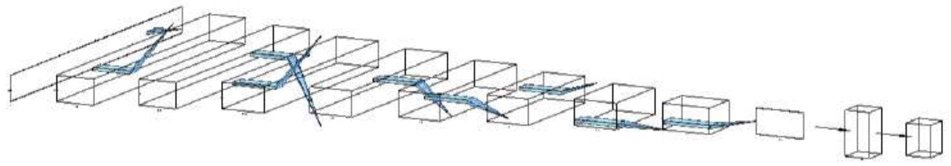


Figure 4.7.: Samplecnn architecture

Sample CNN is compared to an equivalent 2D network at the sample level as well as previous work with frame-level convolutional filters ([8], [6]), and another proposed 2D CNN with low-level convolutional filters (241 samples of filter width). It is evaluated on the million song dataset and the MTAT dataset, as previous works were:

input type	model	MTAT	MSD
Frame-level	Persistent CNN	0.9013	-
(mel-spectrogram)	2D CNN [6]	0.894	0.851
	CRNN [8]	-	0.862
	Proposed DCNN	0.9059	-
Frame-level waveform	1D CNN	0.8487	-
Sample-level waveform	Proposed DCNN	0.9055	0.8812

Table 4.3.: Results from [22]

4.4.2. Adding Squeeze and Excitation

[17] builds upon [22] by taking inspiration from successful 2D convnets (specifically SENets and ResNets) and implement residual connection blocks and squeeze-excitation blocks. Furthermore, it uses multi-level feature aggregation before the dense classification head to ensure the discriminatory dense head sees features from multiple scale levels. This paper focuses only on frame-level raw audio waveforms. The added modifications to the architecture shown in [22] are shown in figure [**samplecnn2**]

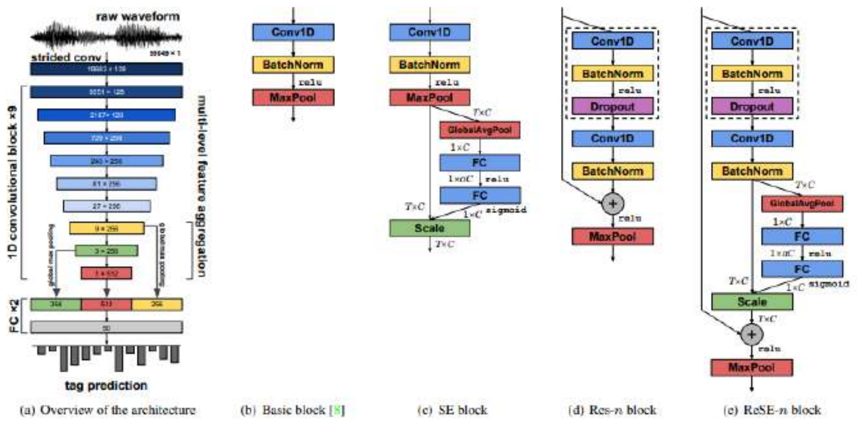


Figure 4.8.: Addition of residual blocks and Squeeze-Excitation blocks to samplecnn [17]

This additional leads to substantial gains in performance compared to previous work and the previous samplecnn on both the MTAT dataset and the Million song dataset (4.4):

Model	MTAT	MSD
Timbre CNN [32]	0.8930	-
2D CNN [6]	0.8940	0.8510
CRNN [8]	-	0.8620
Multi-level & multi-scale [20]	0.9017	0.8878†
SampleCNN multi-features [21]	0.9064	0.8842
SampleCNN [22]	0.9055	0.8812
SE [This work]	0.9111	0.8840
ReSE [This work]	0.9113	0.8847

Table 4.4.: Results for SampleCNN-ReSE [17]

4.5. HarmoniCNN

Intuition and architecture

Similar to [32], [49] aims to incorporate domain knowledge into the very architecture of their proposed model. The harmonic constant-Q transform (HCQT) is a 3-dimensional representation whose dimensions are harmonic(H), frequency (F), and time (T). By stacking standard constant-Q transform (CQT) representations, one harmonic at a time, the output representation (i.e., HCQT) can preserve the harmonic structure while having spectro-temporal locality

In previous work, two learnable sinc functions (i.e., $sin(x)/x$) were used to form each band-pass filter of the first convolutional layer, such that the set of harmonics can be learned. By aligning the convolution band-pass filters in each harmonic, the first layer outputs an H × F × T tensor. When the first harmonic center frequencies are initialized with a MIDI scale, this can be interpreted as an extended, more flexible version of HCQT.

However, the convolution band-pass filter approach to get harmonic spectro-temporal representations requires many convolutions (H × F), including redundant ones (e.g., a 440Hz filter is equivalent to the second harmonic filter of 220Hz). To overcome these efficiency limitations, in this work they replace the convolution band-pass filters of our previous work with an STFT module followed by learnable triangular filters, the so-called Harmonic filters (See figure 4.9).

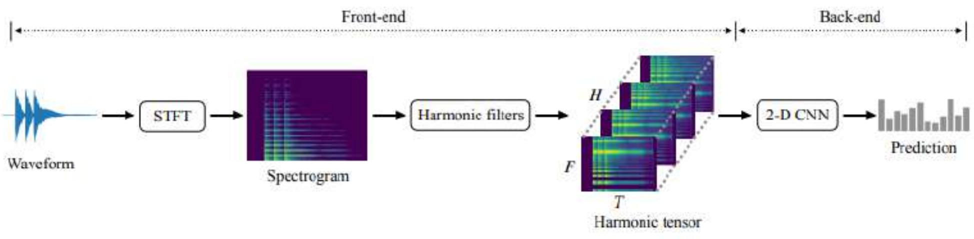


Figure 4.9.: Structure of the harmonicnn network [49]

Where a harmonic filter can be formulated as:

$$\Lambda_n(f; f_c, \alpha, \beta, Q) = \left[1 - \frac{Q|f - nf_c|}{(n\alpha f_c)}\right]_+ \tag{4.1}$$

Where $[]_+$ denotes a rectified linear unit function, and α β, Q are learnable model parameters. The proposed learnable filter banks is the set of harmonic filters [49]:

$$\{\Lambda_n(f; f_c)|n = 1, ..., H, fc \in \{f_c^{\{}(1)\}, ..., f\{(F)\}_c\}\} \tag{4.2}$$

Center frequencies of all the harmonic filters are also learnable paramaeters. This approach differs from previous approaches in that it proposes a F-channel input to the 2D CNN backend instead of a single channel input as before. Furthermore, the approach is novel even in classic computer vision as the channel representations are learned at the same time as the CNN backend.

Results

HarmonicNN is tested on multiple tasks : **music tagging, keyword spotting and sound event tagging**. Only the results for the first task will be presented, though it should be noted that HarmoniCNN generalizes well to in-domain tasks due to the incorporation of domain knowledge to the architecture. STFTs are generated with $n_fft = 512$. HarmoniCNN is evaluated on the MTAT dataset, with the same split as [22], [50], which is the first acknowledgment of a standardized split for inter-model performance comparison on the MTAT dataset. The results are shown in the following table (Table 4.5):

Methods	**Music Tagging**	
	MTAT	
	ROC-AUC	PR-AUC
Musicnn [32]	0.9089	0.4503
Sample-level [22]	0.9054	0.4422
+ SE [17]	0.9083	0.4500
+ Res +SE [17]	0.9075	0.4473
Proposed	0.9141	0.4646

Table 4.5.: results from [49]

4.6. Short-chunk CNN

4.6.1. Two architectures

It is intuited in [49] and shown in [51] that a simple VGG-like CNN trained on short chunks of audio can provide state of the art results on audio music tagging tasks. Though no paper is explicitly dedicated to it, it is implemented in [51] for comparison's sake with other state of the art models.

The architecture for this model is quite similar to that seen in [6] and in figure 4.1 : a series of 2D convolutional layers deepen the input representation while squeezing it horizontally andvertically. However, it differs in that[6] is trained on 29.1 long second audio (song-level evaluation), which means that the time-wise max pooling dimension reduction steps are much more reductive. This is due to the need of the model to have a large receptive field to capture features on all 29.1s. The **Short-Chunk CNN** - as is dubbed this architecture - however, trains on 3.69s long audio excerpts, which greatly restricts the necessary receptive field of the model and allows for smaller max-pooling size (2 versus 4 in [6].)

An additional step to increasing the already state of the art performance of the Short-Chunk model is adding residual skip-connections. These connections, similar to the **Res block** in [17], allow for the flow of original information from one convolutional block instead of only residuals. This alleviates the gradient vanishing problem often found in deep CNNs by shortening the path between information in the network, and shows better results as shown in [51][2]

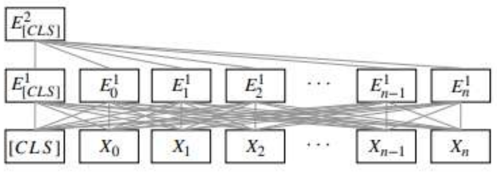
4.7. Convolutional front-end with self-attention

[50] introduces the transformer model for the first time in music auto tagging. Transformers are a class of ML models that rely only on the self-attention mechanism, which is a mechanism allowing them to learn the relationships between features of input representations. This has proven invaluable in fields such as Natural Language Processing, where transformer models have revolutionized the landscape of the domain. Transformer models are highly skilled at dealing with sequential data, and at interpretability. [50]'s goal is to build upon these innate capabilities by introducing a transformer encoder back-end as a temporal feature aggregator for a convolutional front-end for music tagging. The intuition resembles that of [8] in that a temporal feature aggregator on top of a proven feature extractor is a good idea for an inherently sequential media. It differs mostly in two ways :

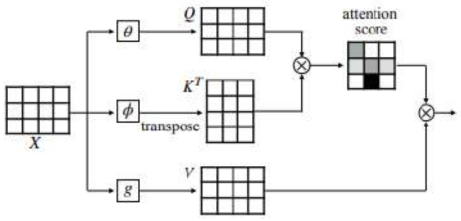
- The feature aggregator is, as discussed, no longer a RNN but a transformer. This, as shown in literature [44] allows for faster computation times within the model due to parallelization and shorter paths between inputs in the model, leading to less gradient vanishing (important for sequences of long lengths)
- The length of the audio used for training in [50] is between 5 and 15s, with [51] citing it as 15s, which represents a lower length than the 29.1s of [8]

The self-attention back-end as well as a schematic of the self-attention mechanism (which can also be found in [44]) are shown in the following figure (4.10):

[2]As this model was only implemented in a paper that recaps most known music tagging models to date and their performance on one same standardized dataset, these results will be presented in section 4.12 along with those of the paper which are of most interest, the comparison results.



(a) CNSSA transformer backend with two layers of self-attention encoders [50]



(b) Self-attention mechanism in CNSSA [50]

Figure 4.10.: Architecture proposed in [50]

Results

[50] implements two front-ends to go with the novel back-end, musicnn's frontend (2D) [32] and samplecnn's front-end (1D) [22], and compares results with their respective back-ends and the transformer back-end on MTAT and Million song:

		MTAT		MSD	
Front-end	Back-end	AUROC	AUPR	AUROC	AUPR
Raw	CNNL	90.62	44.20	88.42	-
Raw	Att (Ours)	90.66	44.21	88.07	29.90
Spec	CNNP	90.89	45.03	88.75	31.24
Spec	Att (Ours)	**90.80**	**44.39**	**88.14**	**30.47**

Table 4.6.: results for CNSSA [50]

4.7.1. Semi-supervised music tagging transformer

Another paper which elaborates on CSSA is [48], implementing a semi-supervised learning version of [50]. The back-end architecture doesn't change, the front end is swapped for a 7-layer short-chunk CNN. Melgrams are extracted using 1024 n_fft and 128 mel bins for 15s audio clips resampled to 22.5kHz.

[48] Uses noisy student training to use the full 1.2M songs of the million-song dataset. Put simply, noisy student training is explianed in [48]:

> First, we train a teacher model T with a conventional supervised learning approach. Then, we train a student model S with two types of losses. The first loss, L_1, is coming from the typical supervised approach with labeled data as done for the teacher model. The other loss, l2, is from unlabeled inputs and the corresponding pseudo-labels provided by the teacher model . In order to make the student model perform beyond mimicking the teacher model, data augmentation is applied.

Through knowledge distillation (i.e when the student model is smaller than the teacher model), [48] reaches state of the art performance on a new proposed split on the million-song dataset with 9 times less parameters than the largest previous model. The results on the conventional split are shown in the following table (4.7):

Models	**ROC-AUC**	**PR-AUC**
FCN	0.8742	0.2963
Musicnn	0.8788	0.3036
Sample-level	0.8789	0.2959
Sample-level+SE	0.8838	0.3109
CRNN	0.8460	0.2330
CNNSA	0.8810	0.3103
Harmonic CNN	0.8898	0.3298
Short-chunk CNN	0.8883	0.3251
Short-chunk ResNet	0.8898	0.3280
Transformer (proposed)	0.8916	0.3358
Transformer (proposed) + DA	**0.8972**	**0.3479**

Table 4.7.: Results for the semi-supervised CNSSA [50]

4.8. SpecTNT

[24] Implements the first fully-transformer model towards musical audio tasks. SpecTNT, or Spectral Transformer in Transformer, uses nested transformer models to extract both the time-feature sequences and frequency-feature sequences of the input mel spectrogram. To do this, positional encodings and frequency class tokens are appended to extracted features from a convolutional front-end (as shown in figure 4.11)

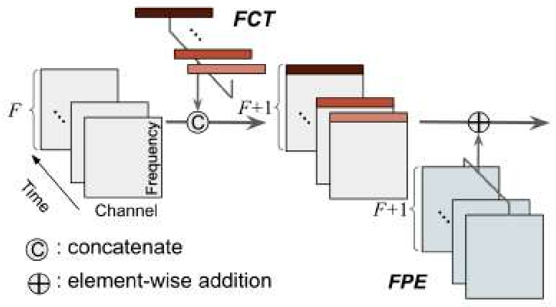


Figure 4.11.: Positional encoding and frequency token appending to input features [24]

Channels are extracted using a convolutional front-end and the resulting Time-Frequency-channel representation of shape (T, F, K) is fed into the SpecTNT architecture. At each encoder, temporal and spectral embeddings are used as input for the self-attention mechanism of the spectral encoder, the output of which is concatenated to the original time encoding and used as input for the time encoder - Hence the name transformer in transformer. This is shown in figure 4.12

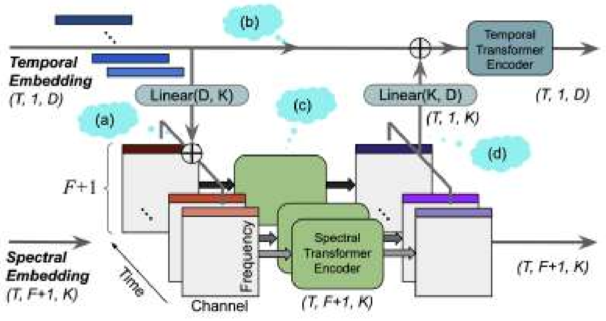


Figure 4.12.: Data flow through the spectnt model [24]

SpecTNT is trained on the million song dataset and compared to CNSSA [50] and ShortChunk-res [51], attaining state of the art on the given music tagging task as shown in table 4.8, but also on other tasks of chord recognition and melody extraction, showing global pertinent representations learned for music audio related tasks:

Method	**ROC-AUC**	**PR-AUC**
Short-chunk CNN + Res	91.55	37.08
CNNSA	91.57	37.09
SpecTNT	92.08	38.62

Table 4.8.: Results on music auto tagging for SpecTNT

4.9. Recap of results and input representations

Many models have been explored in this section, but few of them present reproducible results on a given data split with a consistent architecture. This section focuses on recapping both inputs (representations, hyperparameters, and lengths)[3] and results on a canonical split for training and evaluation on MTAT, MSD, and the MTG-Jamendo dataset [3] proposed by [51]. The following table recaps the input of all models:

Model	**input length**	**# of Mel**	**training**
FCN	29.1s	96	song-level
FCN (128)	29.1s	128	song-level
Musicnn	3s	96	chunk-level
Musicnn (128)	3s	128	chunk-level
Sample-level CNN	3.69s	-	chunk-level
CRNN	29.1s	96	song-level
CRNN	29.1s	128	song-level
Self-attention	15s	128	chunk-level
Harmonic CNN	5s	-	chunk-level
Short-chunk CNN	3.69s	128	chunk-level

Table 4.9.: inputs for most presented models

Note: some audio models are missing, notably SpecTNT - written after the release of [51] - and multi-level multi-scale samplecnn [21] as well as some other long-outperformed models from literature. These models were all evaluated on a common split of MTAT, MSD and MTG-Jamendo. The results are reported within the following table for the task of music tagging.

Methods	**MTAT**		**MSD**		**MTG-Jamendo**	
	ROC-AUC	**PR-AUC**	**ROC-AUC**	**PR-AUC**	**ROC-AUC**	PR-AUC
FCN	0.9005	0.4295	0.8744	0.2970	0.8255	0.2801
FCN (with 128 Mel bins)	0.8994	0.4236	0.8742	0.2963	0.8245	0.2792
Musicnn	0.9106	0.4493	0.8803	0.2983	0.8226	0.2713
Musicnn (with 128 Mel bins)	0.9092	0.4546	0.8788	3036	0.8275	0.2810
Sample-level	0.9058	0.4422	0.8789	0.2959	0.8208	0.2742
Sample-level + SE	0.9103	0.4520	0.8838	0.3109	0.8233	0.2784
CRNN	0.8722	0.3625	0.8499	0.2469	0.7978	0.2358
CRNN (with 128 Mel bins)	0.8703	0.3601	0.8460	0.2330	0.7984	0.2378
Self-attention	0.9077	0.4445	0.8810	0.3103	0.8261	0.2883
Harmonic CNN	0.9127	0.4611	0.8898	0.3298	**0.8322**	**0.2956**
Short-chunk CNN	0.9126	0.4590	0.8883	0.3251	0.8324	0.2976
Short-chunk CNN + Res	**0.9129**	**0.4614**	**0.8898**	**0.3280**	0.8316	0.2951

Table 4.10.: results for all known music tagging architectures[51]

In general, models trained with short audio excerpts (Musicnn, variants of samplelevel CNN, Self-attention, Harmonic CNN, variants of shortchunk CNN) outperform other models trained with relatively longer audio segments (FCN, CRNN). Training with short chunks (instances) is noisier: e.g., an audio excerpt can have a tag guitar if a guitar appears in the song even though the selected excerpt doesn't include guitar sound in it. However, one can expect a much larger number of examples during the training (e.g., 25,877 tracks $\times$ 16 chunks = 414,032 examples). We suspect this brings the performance gain when the model is trained with short chunklevel examples. Furthermore, most of the top 50 tags in the three datasets can be identified only with a short audio excerpt (e.g., instruments, genres). Thus, the model does not need a long sequence of audio to perform its binary classification task. For the top 50 tags in each dataset we experimented with, it is more beneficial to use chunk-level training with short audio excerpts than the song-level training. Short-chunk CNN, short-chunk CNN with residual connections, and Harmonic CNN showed the best results for every dataset. These three models are trained on short audio excerpts (3.69s or 5s).

[3]all audio was resampled to 16kHz and all spectrogram representations used a 512-point 50% overlapping fft window with default window function

Musicnn shows competitive results in MTAT. However, other models (sample-level + SE, self-attention) outperform Musicnn on larger datasets (MSD and MTG-Jamendo). This confirms an intuition that domain knowledge can be beneficial for relatively small datasets, reported in [32]. However, the design choices for parameters of Musicnn restricts the power of the model when it is trained on larger datasets.

These results and intuitions will serve us when selecting our baseline models to train for the proposed custom music tagging task.[4]

[4]Results for CLMR and CALM are not shown here as they were not elaborated on within this internship. However, they are highly interesting models which propose task-agnostic representation learning and are closest to what we would consider as frozen preprocessing models in and of themselves. See page 26 for detailed presentations of these models done over the course of the internship.

5. Music tagging datasets

This section focuses on presenting the available audio datasets with annotations useful for music tagging training. These datasets have been well explored in literature and have been used in the previously described studies. For each dataset, a presentation of the contained annotations as well as summary distribution considerations will be explored, as cleaning and data quality & selection will be reserved for a later section.

Note : Though the million song dataset has been used in many of the studies previously cited, it is no longer publicly available in its audio form today. It is only comprised of audio metadata and extracted features (via echonest), and does not contain any audio files. The only remaining audio from the dataset in circulation is in the form of 30s long preview highlights of each song, which were once publicly available for download - not anymore. They are now only in the possession of organizations which have used them in the past and are not publicly licensed for use anymore. Hence, we do not use the Million Song Dataset.

5.1. MagnaTagATune dataset

The MagnaTagATune dataset [19] is derived from a game called TagATune, where users annotate songs in an attempt to guess if they and the person they are playing with are listening to the same audio. It contains multi-label annotations of 188 tags by genre, mood, and instrumentation for 25,877 audio segments, each 29.1 long. The audio is in the MP3 format (32 Kbps bitrate and 16 kHz sample rate). 16% of the available annotated clips do not have any tags, which means we can discard them. The top 20 and top 50 tags are shown below (5.1):

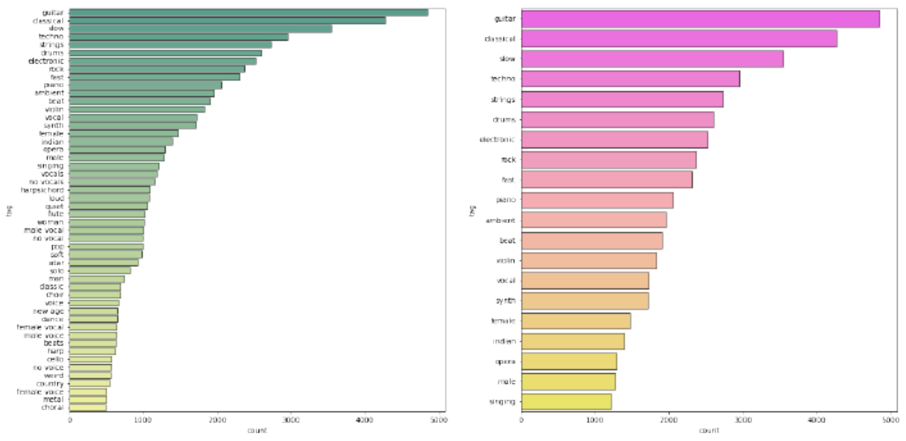


Figure 5.1.: Top tags by count in the MTAT dataset

And the distribution of tags per song is given below (figure 5.2)

5.2. FMA dataset

The Free Music Archive [10] is a relatively new dataset (created in 2017) which has grown over the years. It is comprised of 106,574 songs with 16,341 distinct artists, and tags relative to genre, mood, theme, instruments, feel, years. The FMA dataset is available in multiple subsets:

- FMA Small (`https://os.unil.cloud.switch.ch/fma/fma_small.zip`): 8,000 tracks of 30s, 8 balanced genres (GTZAN-like) (7.2 GiB)
- FMA medium (`https://os.unil.cloud.switch.ch/fma/fma_medium.zip`): 25,000 tracks of 30s, 16 unbalanced genres (22 GiB)

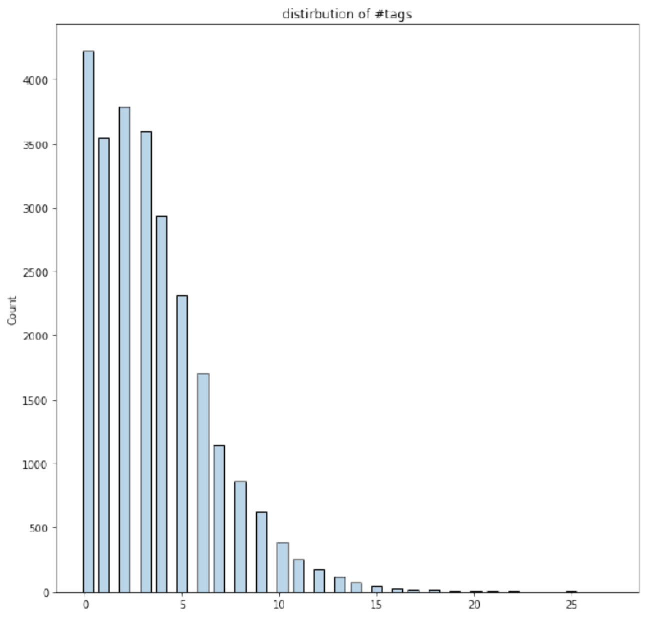


Figure 5.2.: distributions of tag per song in the MTAT dataset

- FMA large (`https://os.unil.cloud.switch.ch/fma/fma_large.zip`): 106,574 tracks of 30s, 161 unbalanced genres (93 GiB)
- FMA full (`https://os.unil.cloud.switch.ch/fma/fma_full.zip`): 106,574 untrimmed tracks, 161 unbalanced genres (879 GiB)

Due to available disk space considerations;, we use FMA large for this study, counting on other datasets to provide the extra needed data. Are also provided a file and a genres file with respectively all tags for tracks and the genre hierarchy. Genres in the FMA dataset are organized hierarchically, into subgenres and subsubgenres. Genres are exclusive and do not have a parent genre. Subgenres have a parent genre which is a genre and a top genre which is that same genre, and subsubgenres have a parent genre which is a subgenre and a top genre which is a genre:

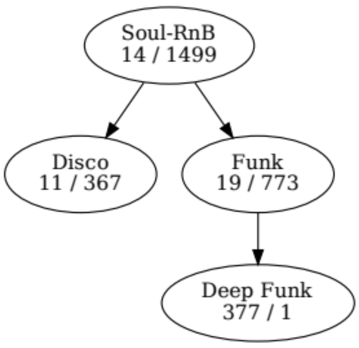
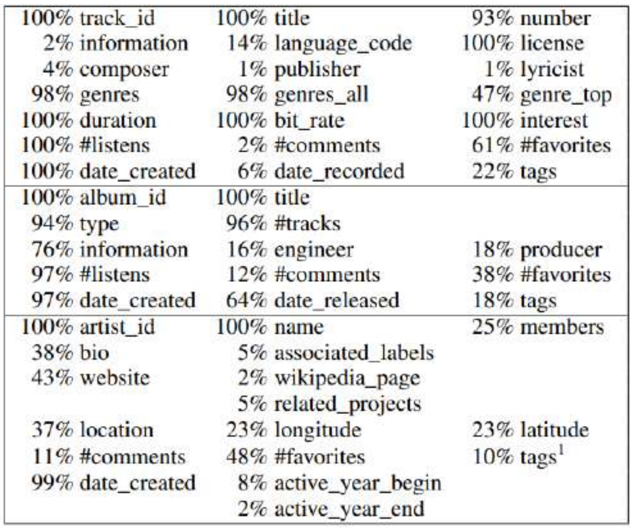


Figure 5.3.: Example of genre hierarchy in the FMA dataset

The paper provides some additional information regarding this dataset (see figure **??**) Where the percentage values next to the column name indicates the percentage of available data. All the necessary data for us is close to 100%. We will only be focusing on track metadata as albums and artists pass on their tags to the tracks:



100% track_id	100% title	93% number
2% information	14% language_code	100% license
4% composer	1% publisher	1% lyricist
98% genres	98% genres_all	47% genre_top
100% duration	100% bit_rate	100% interest
100% #listens	2% #comments	61% #favorites
100% date_created	6% date_recorded	22% tags
100% album_id	100% title	
94% type	96% #tracks	
76% information	16% engineer	18% producer
97% #listens	12% #comments	38% #favorites
97% date_created	64% date_released	18% tags
100% artist_id	100% name	25% members
38% bio	5% associated_labels	
43% website	2% wikipedia_page	
	5% related_projects	
37% location	23% longitude	23% latitude
11% #comments	48% #favorites	10% tags[1]
99% date_created	8% active_year_begin	
	2% active_year_end	

Figure 5.4.: Missing data in the FMA dataset

We explore the genre(B.1a)-subgenre(B.1b)-subsubgenre(5.5) distribution here for subsubgenres as an example and in annex B for the rest to save space here. Since we use FMA-large, all the tracks are of length 30s.

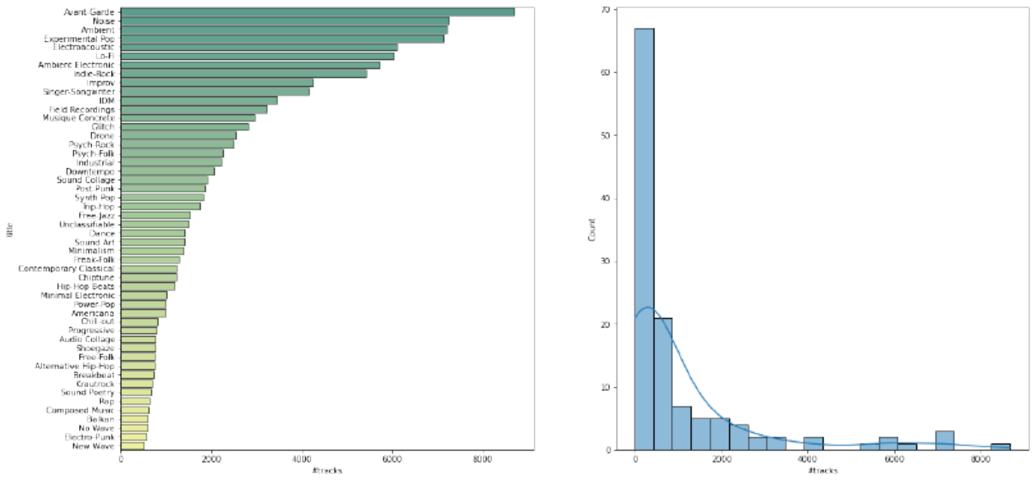


Figure 5.5.: subsubgenre distribution and tracks per label distribution in FMA

5.3. MTG-Jamendo Dataset

The MTG-Jamendo dataset contains audio for 55,701 full songs and is built using music publicly available on the Jamendo 6 music platform under Creative Commons licenses. The minimum duration of each song is 30s, and they are provided in the MP3 (320 Kbps bitrate). The tracks in the dataset are annotated by 692 different tags covering genres, instrumentation, moods and themes.

It weighs approximately 508Gib with no true randomized subset to download. As such, metadata was downloaded but data such as bitrate and/or sampe rate was not obtainable. The MTG-jamendo dataset is organized in tags which can be of three types : genre, instrument and mood/theme. The distribution for these tags is shown below (figure 5.7):

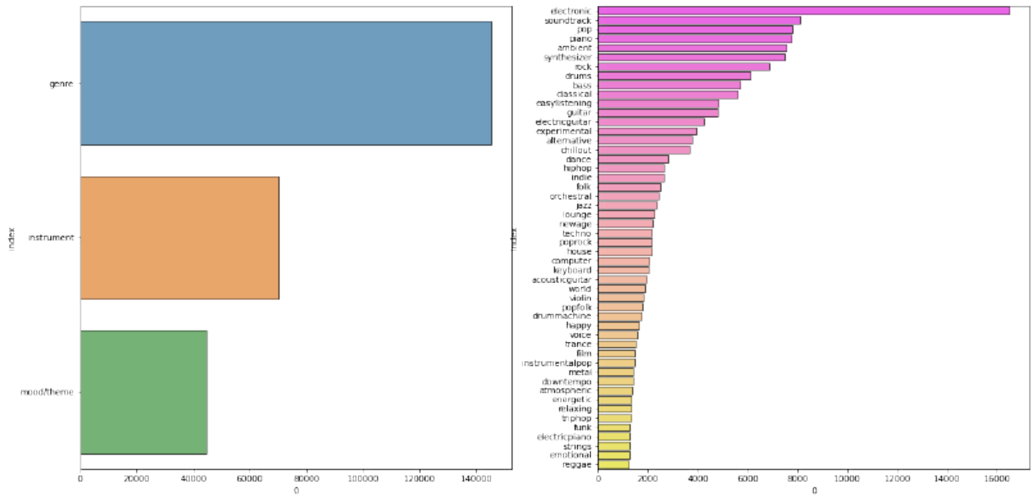


Figure 5.6.: Tag distribution in the MTG-jamendo dataset

Genres are a highly dominant tag type, with more genre tags that instruments and mood/theme combined. All tag types are rather well mixed within the dataset and many mood/theme tags can be seen in the distribution. Other metadata is observed, such as number of tags per track and duration of track:

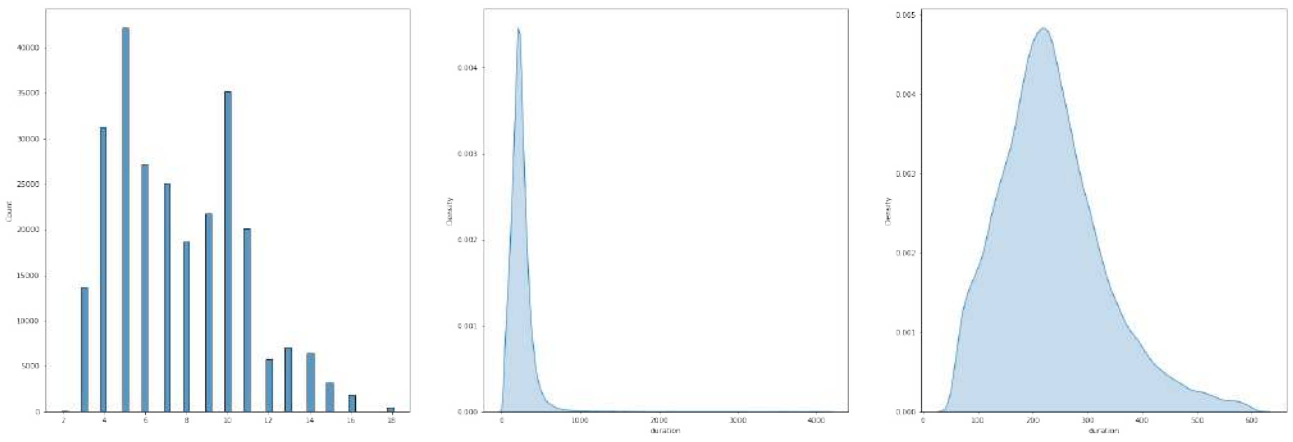


Figure 5.7.: Tag per song distirbution and song duration distribution within the MTG-Jamendo dataset

Part II.

Implementation

This section focuses on implementation of the previously described models on our own custom task. To do this, we firstly further explore the three previously described datasets to identify necessary cleaning steps, as well as desired tags for each round of testing. We build the necessary data wrangling pipeline and address class imbalance distribution issues and dataset noisiness considerations. A consideration of usable models in the face of many restrictions was conducted, which will also be exposed in this chapter, and the exact design of the preprocessing pipeline, as well as the iteration steps over it, will be presented. Finally, training results and evaluations are presented for all tag wrangling types and all selected models.

6. Data exploration and selection

6.1. General purpose exploration and quality of data

In the previous section we had a summary look at the distribution for the tags of each dataset and the format of these tags to understand a bit better how we can work with them. However, simply checking the format is not sufficient. One must also check for missing data, bad quality data, highly skewed data to set up a cleaning process to have the cleanest data possible. This includes audio features such as length, sample rate, no tags, etc.

This section focuses on the process of cleaning up and selecting relevant data from each dataset before moving on to tag selection for all types of tags.

6.1.1. FMA

The distribution for genres has already been explored for FMA, and the available sub-genres and sub-sub-genres can be found in Annex B. Looking at missing data, the following graph shows missing data proportion per column of the tracks dataframe (Figure 6.1):

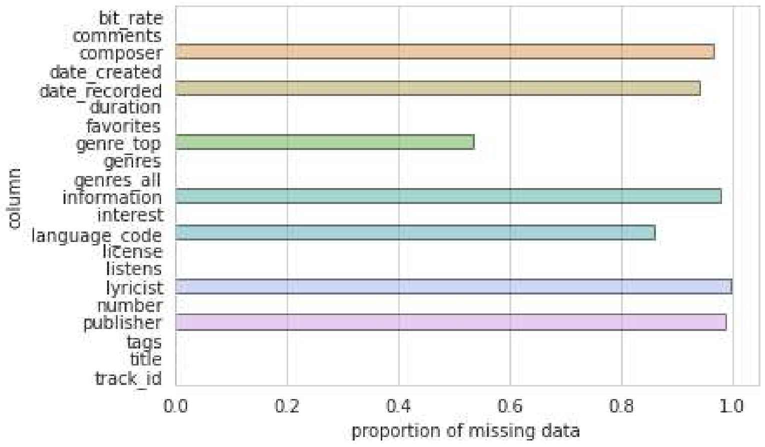


Figure 6.1.: Missing data in the FMA dataset

genre-top, which is the genre most likely to fit the track amongst the previously described top-genres, is missing approx 52% of its data, which means 50% of the dataset has to be discarded if we consider only the top genre. looking at genres and genres-all, which are lists containing the ID of:

- **for genres :** top genres and subgenres
- **for genres-all:** top genres, subgenres and subsubgenres

No data is missing counting null values, but there are empty lists. The same amount of lists are empty for either column, which represents about 2% of the total amount of tracks. Given the size of the dataset, we can throw these unclassified songs out without too much thought. Furthermore, we have to select songs with genres and subgenres which will be relevant to the model (we will not attempt to classify a subgenre which has only 3 tracks). So, we propose the following processing of tracks regarding genres and subgenres:

- Discard any tracks that do not have at least tag in the genreall columns that is either:
  - Within all the top genres
  - Within the top 10 subgenres
  - Within the top 25 subsubgenres
- For tracks that do have at least one tag within those lists, throw out the tags that are not within those lists.

After genre and subgenre considerations, tags are available for the tracks as well. 22% of tracks have a non-empty tag vector. A first glance at the distribution of track count per tag yields the following distribution (figure 6.2):

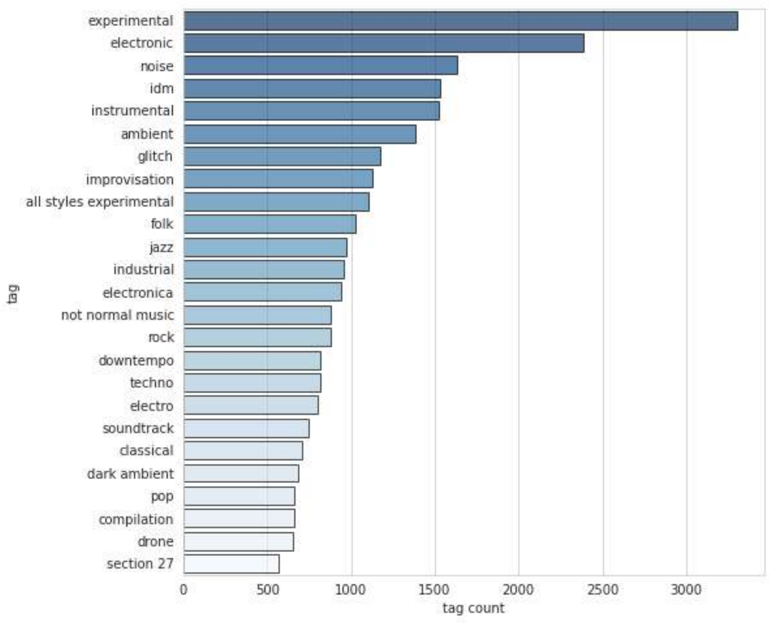


Figure 6.2.: tag distribution for the FMA dataset

Notice that many of the top tags are also genres, subgenres or subsubgenres. Removing these tags non-case-specifically yields the new list of top tags that have not already been considered (6.3):

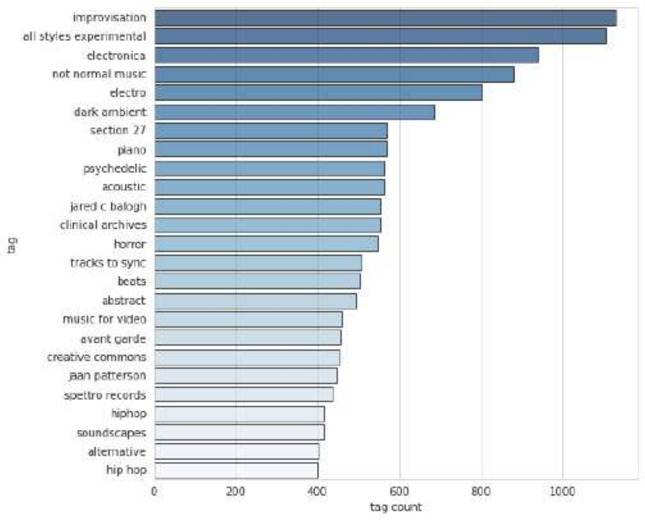


Figure 6.3.: new tag distribution for the FMA dataset

Most of these tags are relevant, and so we also choose to include the taglists of tracks that contain tags from within the top 25 tags to the total label vector of each track. Careful however, as some hand cleaning will have to be done (e.g. hip hop is in the tags list but Hip-hop is in the genres list). So, after all this, we have discarded some tracks and all the tracks that are left have a one-hot encoded label vector of 76 total tags. some stratified subsampling will have to be conducted on popular labels to increase the relevance of unpopular labels with regards to model accuracy. After looking at label generation, the quality of audio as well as outliers regarding audio characteristics is also to be taken into account. In figure 6.4 are shown distributions for:

- track bitrate on all tracks
- Track sample rate
- track duration
- track duration when removing tracks with lengths above 1000 minutes.

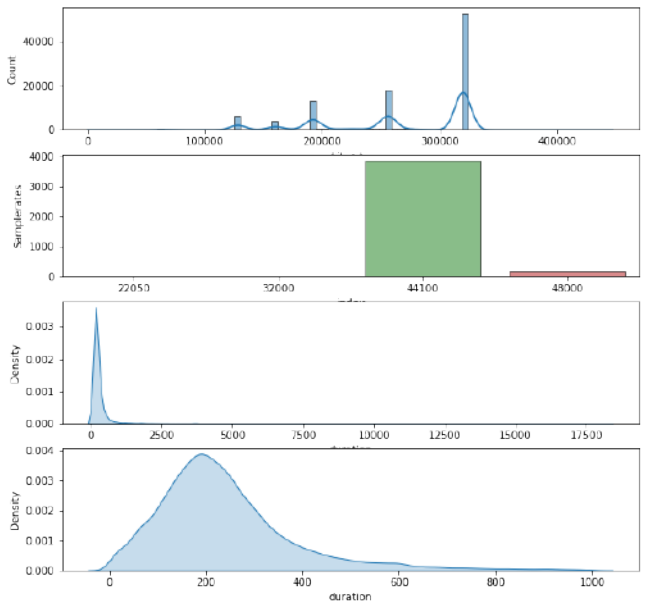


Figure 6.4.: FMA metadata distribution

Most tracks are sampled at 44.1 kHz, which is industry standard. Some other tracks are sampled at 48kHz and 22.5kHz, as well as 32kHz. None of these are far under the sample rates used in the state of the art, so no need to discriminate based on sample rate. Any audio track of duration below 1min and above 10mins (60secs) can be discarded (this represents about 7% of the dataset). Finally, as bitrate has not been shown to influence the performance of audio classification models, we do not filter on bitrate. So, the total selection process for tracks we choose to consider on the FMA dataset is the following :

- Remove tracks of length above 10mins and below 60s
- Downsample tracks with sample rate above 44.1kHz (this is done directly within the preprocessing codebase, no need to do anything in this case)
- Select tracks which only have :
  - genre-all vectors with at least one element within the top-genres, the top 10 sub-genres or the top 25 sub-sub-genres, removing the genre labels that are not within these categories
  - tag vectors that have at least one element within the top 25 tags bar some tags that will be manually cleaned (e.g hip hop or the composer name)

Doing this will yield a clean dataset bar the necessary subsampling which has to be performed before training the model, but this subsampling will be performed later on when all labels from all datasets have been chosen.

6.1.2. MTAT

MTAT has a constant sample rate of 16kHz, which is the lowest sample rate of the three datasets and will thus be our limiting factor for resampling in the preprocessing pipeline. Similarly, all songs are 29.1s long so there is no required filtering on song length. We select the top 100 tags from MTAT to be part of our final dataset. The number of tags per track distribution is shown in the following figure (figure 6.5):

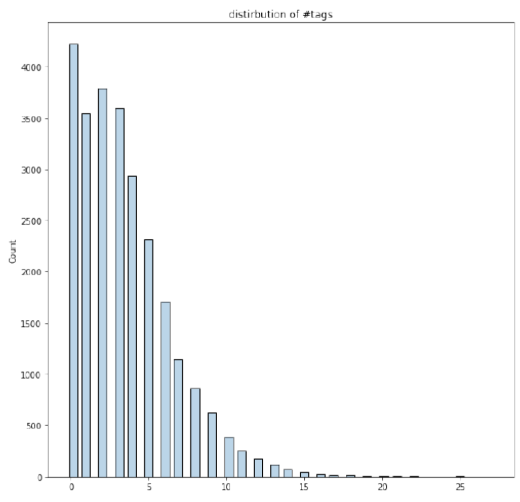


Figure 6.5.: Number of tags per track distribution on the MTAT dataset

Some tracks have more than 15 tags, which are outliers to the rest of the distribution and introduce noise into the dataset, and which we choose to discard. 16% of the available annotated clips do not have any tags, which means we can discard them. We propose the following cleaning pipeline for MTAT: remove songs with 0 tags and more than 15 tags, and discard songs that don't have any tags in the top 100 tags.

6.1.3. MTG-Jamendo

7. Data exploration and selection

For MTG-jamendo, cleaning is rather minimal as well. all songs are the same sample rate (44.1 kHz), so there is no need to oust songs based on sample rate. Since tags are already separated into genres, moods/themes instruments, we select the top 100 genres from the genre section. To check that there are no outliers, the following figure (7.1) shows the dsitribution of tags per track and the duration of the track, which is variable for MTG-Jamendo:

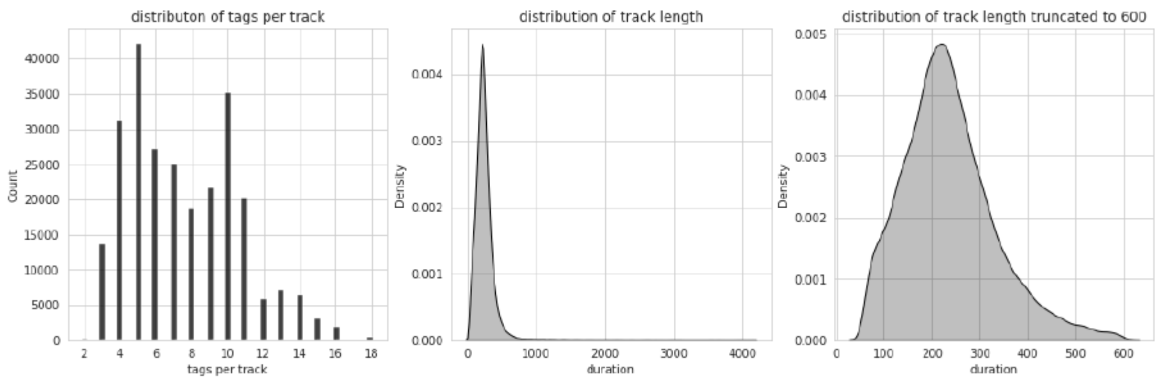


Figure 7.1.: Outlier consideration for the MTG-jamendo dataset

As for MTAT, tracks of length above 600s and below 60s are cut (5% of all tracks). No tracks have 0 tags, which is nice from a data cleaning standpoint.

7.1. Tag selection and distributions before sub/oversampling

We decided at the beginning of the internship to work incrementally with different tag types : Start easy with genres, which are the most mutually exclusive tag set, and allow to get closer to a multiclass single-label task. We then move on to subgenres, then mood/themes/instruments, and finally, we combine all these tags into our own music tagging task relevant to the business use case at groover and different than the typical 50-tag task seen in literature. So, after each round of training and basic fine-tuning for each tag type, we select the tags for the next round of experiments.

Furthermore, tags are noisy in these datasets. for instance, hip-hop and hip hop can bre present within a same dataset. Likewise, male vocals and male singing can be considered as the same tag and are both present in the MTG-Jamendo dataset. This noisiness problem would be very difficult to deal with if we were attempting to classify all tags on all datasets. However, since we select a number of tags to work on at each round of experiments, we peruse the tags from the dataset and do some cleaning by merging tags we consider to be mergeable (for both data augmentation purposes and cleaning purposes). This merging process is also documented in this section.

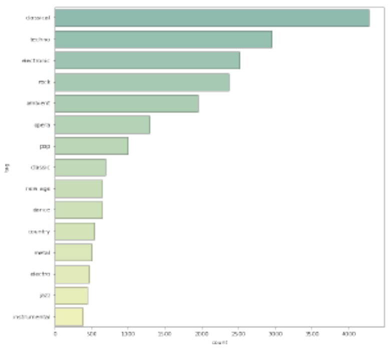
7.1.1. Genres

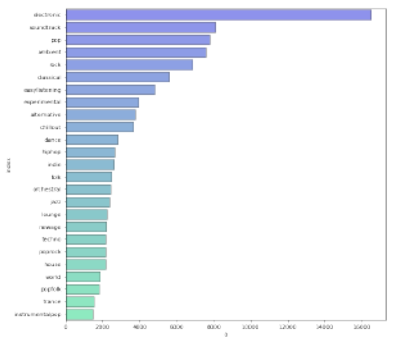
Among the top tags, the genres are isolated a la mano when not specified - for MTAT for instance. The aim here is to choose an array of genres that are both pertinent to the business case at Groover, and represent a broad array of musical genres to not overspecify the model's objective. the genre distributions for MTAT, MTG Jamendo and FMA are shown below (figures 7.2b, 7.2a, 7.2c)

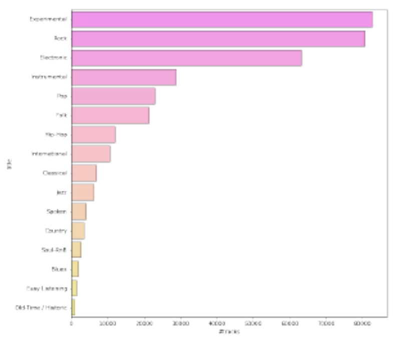
There are many tags here that can potentially refer to very similar things (e.g rock and alternative, hip-hop and rap). Our goal is to merge these tags to encapsulate these concepts better (for genres and for other types of tags as well), but also to reduce noisiness in the datasets (e.g changing 'vocals' to 'vocal' when the 'vocal' tag is already in the dataset tags).

Furthermore, we want to merge the datasets together into one proprietary split of the FMA-MTAT-MTG dataset, which requires some standardization - both in tag taxonomy and format. We use the following processing steps to do this:

- Select relevant tags among each dataset (overlap is possible here)


(a) top genres distribution for MTAT


(b) Top genres distribution for MTG


(c) Top genres distribution for FMA

- filter each dataset for outliers using the procedures outlined in 6.1
- Build a correspondence dictionnary to reduce tags in each dataset by replacing unnecessary tags with relevant ones.
- standardize the format of the tags (remove special characters, convert to lowercase).
- concatenate the individual datasets.

This will yield a clean dataset with only relevant tags and tracks from all three datasets. For reference, this is conducted in the following on only genres tags for the sake of legibility. 19 total tags are selected and reduced using correspondence dictionaries for each dataset. The wanted tags and correspondence dictionaries for genres for all datasets are shown here. The rest are shown in annex B.

```
"FMA": {
    "wanted_genres": [
        "international","blues", "jazz", "classical", "country", "pop", "rock",
        "easylistening", "soulrnb","electronic", "folk", "spoken", "hiphop",
        "experimental", "instrumental",],
    "genre_merging": {
        "classic": "classical", "dance": "electronic", "electro": "electronic",
        "new age": "ambient", "country": "folk", "opera": "classical",
        "techno": "electronic", "spoken": "hiphop", }, ... }

"MTAT": {
    "wanted_genres": ["techno","classical","electronic","rock","opera","pop",
    "classic","dance","new age","ambient","country","metal","electro","jazz",
    "jazzy", "instrumental","foreign","orchestra","folk",],
    "genre_merging": {
        "classic": "classical","dance": "electronic","electro": "electronic",
        "new age": "ambient","country": "folk","opera": "classical",
        "techno": "electronic","jazzy": "jazz","foreign": "international",
        "orchestra": "orchestral",
    },...}


"MTG": {
    "wanted_genres": [
        "electronic","soundtrack", "pop", "ambient","rock", "classical",
        "easylistening","experimental","alternative","chillout", "dance",  "hiphop",
        "indie","folk","orchestral", "jazz","lounge", "newage","techno","poprock","house",
        "world", "popfolk", "trance","instrumentalpop",
        "metal","funk", "blues", "rap", "symphonic","darkambient",
    ],
    "genre_merging": {
        "alternative": "rock", "dance": "electronic",
        "lounge": "jazz", "newage": "ambient",
        "techno": "electronic","poprock": "rock",
        "house": "electronic", "popfolk": "folk",
        "trance": "electronic", "instrumentalpop": "pop",
        "world": "international","funk": "soulrnb",
        "blues": "rock", "rap": "hiphop",
        "symphonic": "orchestral","darkambient": "ambient",
    },...}
```

This yields the following new genre distribution with new genres and old genres specified (figure 7.3):

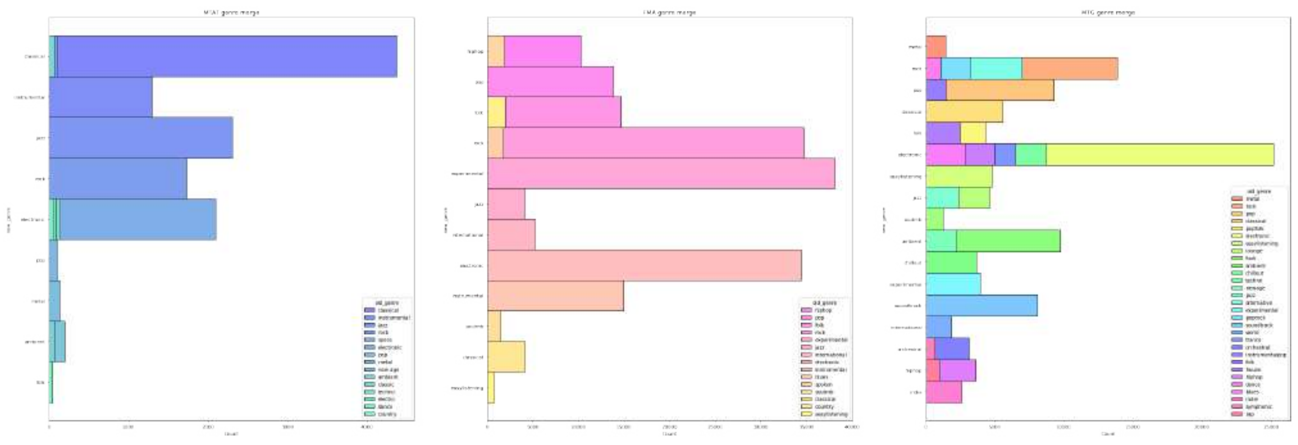


Figure 7.3.: Merging genres in all three datasets

And the genre distribution after dataset merger is shown in following figure (fig. 7.4) Where we can see that despite our best efforts there is still large skewness in favor of some genres, which will require sub/oversampling in future processing considerations

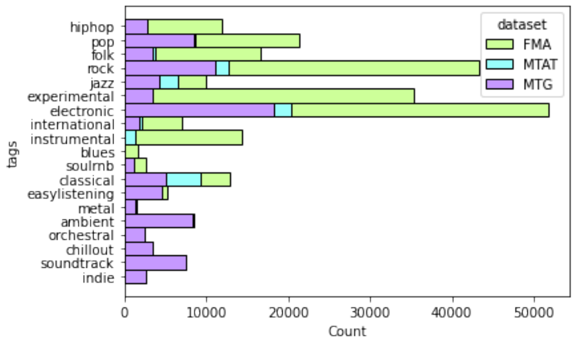


Figure 7.4.: Merging datasets to get final genre distribution

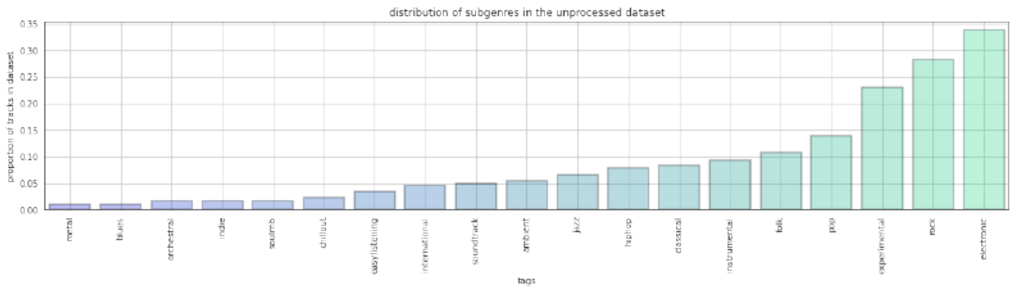
7.1.2. Distribution before resampling for all tags

The same process is repeated for all tag types. At each iteration, relevant tags are taken into careful consideration and the appropriate merging dictionaries are constructed with business appropriacy in mind. A total of:

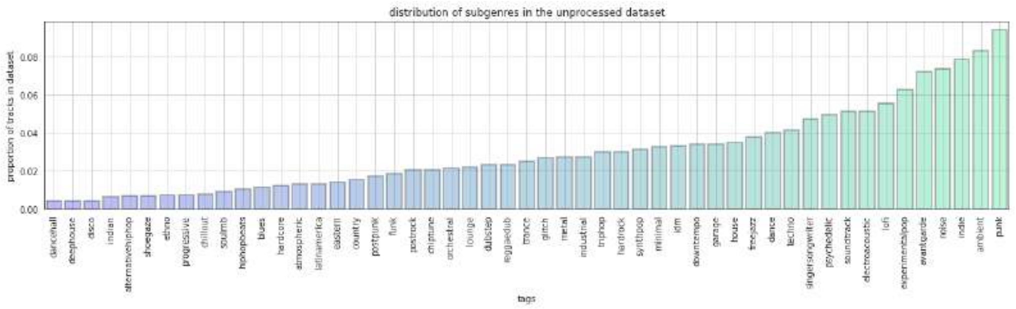
- 19 tags are selected for genres
- 51 tags are selected for subgenres

- 38 tags are selected for moods/themes/instruments
- 80 tags are selected for alltags

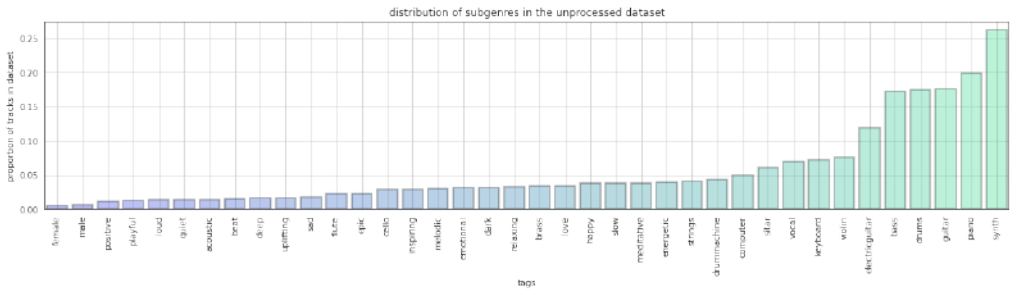
Selection for all tags is slighly different than other selections since we cannot afford to simply add all the previous tags, which would amount to to large of a solution space dimension (113 total tags). We reduce the amount of tags selected as much as possible and end up with 80 tags. Again, all of the wanted tags per type of tag wrangling and per dataset is shown in annex B, as well as the merging dictionaries. The following figures show the final tag distribution before sub/oversampling for each tag type after selection of the aforementioned tags (7.5a, 7.5b, 7.5c, 7.5d). As can be seen in the following figures, all tag types require some form of sub/oversampling to even out the tag distribution, which will be discussed in the following paragraph.



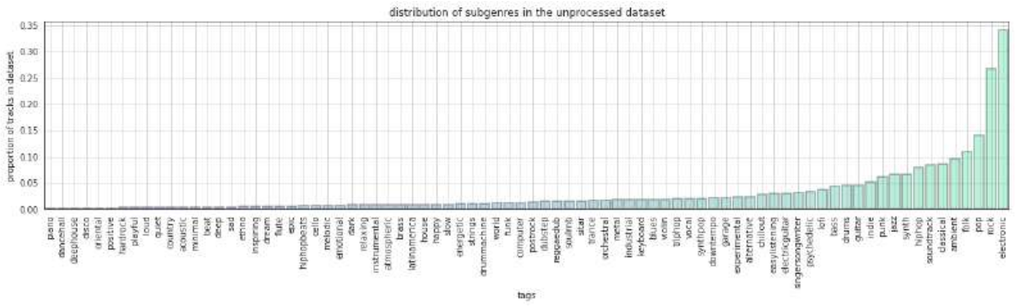
(a) Distribution of genres in merged dataset before resampling



(b) Distribution of subgenres in merged dataset before resampling



(c) Distribution of mood/themes/instruments in merged dataset before resampling



(d) Distribution of all tags in merged dataset before resampling

7.1.3. The need for oversampling and undersampling

The previous section shows that the data is very skewed for some types of tags. For instance, for all tags, "electronic" represents 35% of the total tag volume, which makes the tagging problem for all tags a highly unbalanced problem from a sampling perspective. Notice that in this case, a model which learns to predict electronic 100% of the time would be 35% accurate on a task with 80 tags. This favors model overfitting to the overrepresented tags in the dataset, which is a classic problem in machine learning.

To deal with this, some canonical options exist :

- subsample the majoritary classes before training : this reduces the number of samples associated to the overrepresented classes
- oversample the minority classes : by duplicating some samples of the majority classes, we artifically augment their presence in the dataset.
- attribute higher importance in training to minoritary samples. This skews the model itself towards these underrepresented classes by attributing a higher reward score to them when learning
- creating synthetic data either by performing data augmentation or data interpolation in a representation space for the underrepresented classes.

Our problem requires finesse when adressing this sub/oversampling problem, as it is a multilabel multiclass classification problem. We choose to conduct both subsampling of major classes and oversamping of minor classes. The goal is to flatten out the difference between the count of tags to a given number. for instance, for genres, the minimum number of tracks for a genre is for metal (1500 tracks). The distribution with no sub and oversampling is shown below in figure (7.4) Again, the dataset is highly skewed. To conduct resampling, we still want to conserve original dataset proportions in the new resampled dataset as well as label proportions with comparison to these original datasets. In other words, if metal tags are comprised of 25% MTAT tracks before resampling, we want them to still conform to that metric after resampling). To this end, we conduct stratified sampling based on exploded tags to reduce each tag to 1500 samples (same as the metal tag) and then take the songs from the unsampled dataset which unique identifiers are in the new resampled dataset. This leads to a new distribution shown in figure 7.6

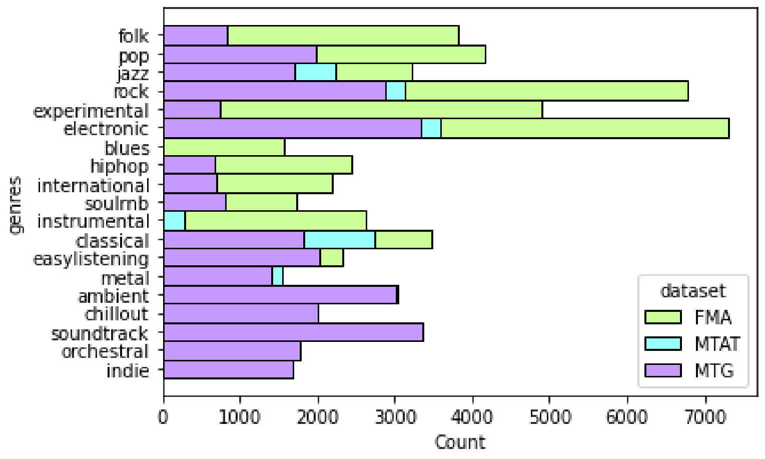


Figure 7.6.: Genre dataset before after resampling

This is also done for the subgenre dataset, the moodthemeinstrument dataset and the alltags dataset. Notice that though the skewness of the dataset has largely diminished **with the new dataset having a max-min ratio of about 6 compared to the previous 33**, the skew has not disappeared altogether. This is due to the multilabl nature of the problem. It would be too artifical of a manipulation to resample only tags with a single label from each genre, and so tracks which are both labeled as metal and rock inevitably contribute to a skewness in favour of rock.

For oversampling, we stratify-resample the datasets as previously while oversampling genres that are under the desired track count to the wanted ratio (e.g if we want 2000 tracks per genre, metal will be resampled to a 1.3 ratio) and preprocess the track as many times as it is resamples. As we wil see later, we introduce a random cropping operation in the preprocessing pipeline which minimizes the artifical aspect of resampling the same track, as the given track is sampled at various starting points each time.

8. Pre-processing pipeline design

In this section we discuss the specifics of the preprocessing pipeline in terms of design and implementation. This includes audio preprocessing but also the wrangling, subsampling and downloading pipeline to smooth out the iterative testing process on multiple datasets and models, while taking into account the possibility of changing input length as well as preprocessing hyperparameters along the way.

In terms of pure audio preprocessing and based on techniques described in Chapter 3, we choose the following steps:

1. Render the file into mono.
2. Resample the file to the target sample rate using the base algorithm described in section 3. Given the data presented in the previous section concerning the available datasets, we choose 16kHz as it is both a limiting factor due to the FMA dataset and a predominant parameter of choice in the state of the art.
3. Clip out the start and end of the tracks (5s) to remove intros and outros
4. Random crop clip to approximate fixed length input (30s)
5. Split audio into overlapping audio clips of target input length computed as $floor(length(s) * sr(Hz))$ with an overlap of half of the target input length (e.g a 10s clip divided into 5s clips with a hop of 2.5s will yield 4 5s clips)
6. Generate log-power mel spectrograms with 128 mel bins based on the state of the art observed in [51]. *Note that for now, only individual melgrams are saved but it could be interesting to look into sequences of melgrams for convolutional RNNs and/or convolutional transformers and more generally sequence classification.*

An example of the preprocessing hyperparameters used to generate a melgram dataset for one of the trained models is shown below :

```
    preprocessing_config = {
    "mono": True,
    "target_sample_rate": 16000,
    "min_sample_rate": 11250,
    "melgram_hop": 1.5,
    "len_split": 3,
    "melgram_params": {
        "window_length": 512,
        "window_stride": 256,
        "n_fft": 512,
        "n_mels": 128,
    },
    "start_cut": 5,
    "end_cut": 5,
    "random_crop": 30,
    "random_crop_occurence": True,
}
```

To conduct the full construction of a dataset ready for training, including train-test-validation split generation, we first wrangle and merge the selected tags for each tag type based on the work explored in the previous section. unique reproducible identifiers are generated based on track names, paths, and random seeding. We then proceed to subsample and oversample the merged dataset as described previously. To save disk space, we only download the required tracks after resampling. These tracks are stored in the AWS EC2 service and are downloaded from the largest version of every dataset described previously. They take up in total about 2TB of space, but only about 30000 tracks are downloaded at a time.

During download, we also delete the local storage of unrequired tracks (*e.g if the split has changed or the wrangling type has changed*). The local audio is then preprocessed into melgrams as described previously, which yields a local dataset of melgrams on which train-test-validation splitting is conducted. The global pipeline is shown in the following figure (Figure 8.1)

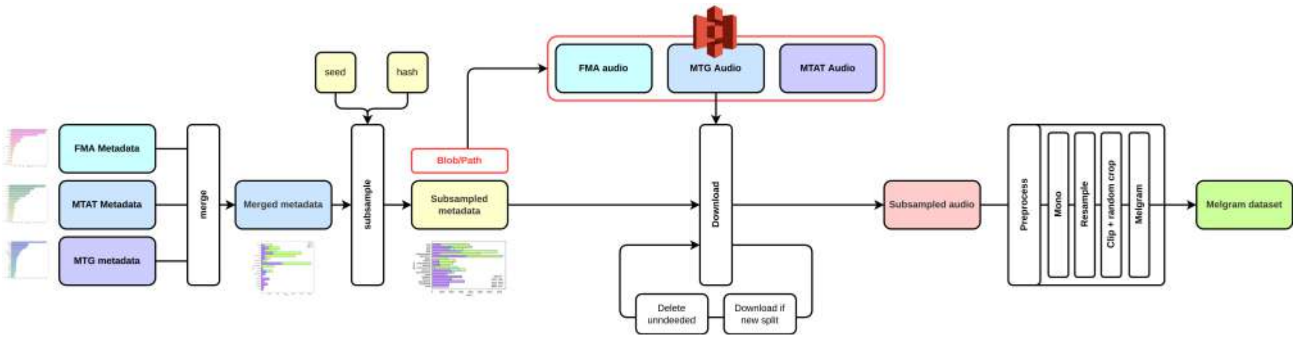
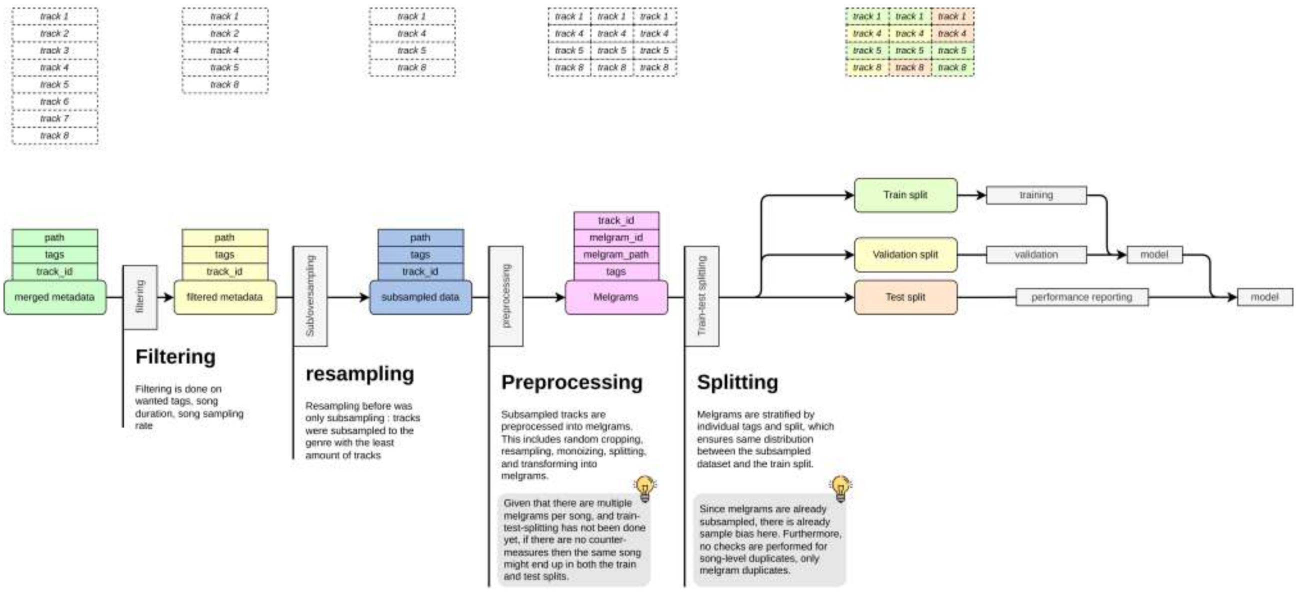


Figure 8.1.: General pipeline : from wrangling to train test splitting

Refactoring the pipeline to avoid sampling bias and data leakage Up till now, the code used to generate the training data and the whole preprocessing pipeline led to some form of data leakage between the train and test sets, meaning that some songs from the train set ended up in the validation set. Furthermore, sub/oversampling was conducted before train/test/validation splitting which is obviously undesirable as it introduces a sampling bias into the results. To summarize the previous full pipeline, the following schematic shows the pipeline from wrangling to training BEFORE refactoring (Figure 8.2).



Filtering

Filtering is done on wanted tags, song duration, song sampling rate

resampling

Resampling before was only subsampling : tracks were subsampled to the genre with the least amount of tracks

Preprocessing

Subsampled tracks are preprocessed into melgrams. This includes random cropping, resampling, monoizing, splitting, and transforming into melgrams.

Given that there are multiple melgrams per song, and train-test-splitting has not been done yet, if there are no counter-measures then the same song might end up in both the train and test splits.

Splitting

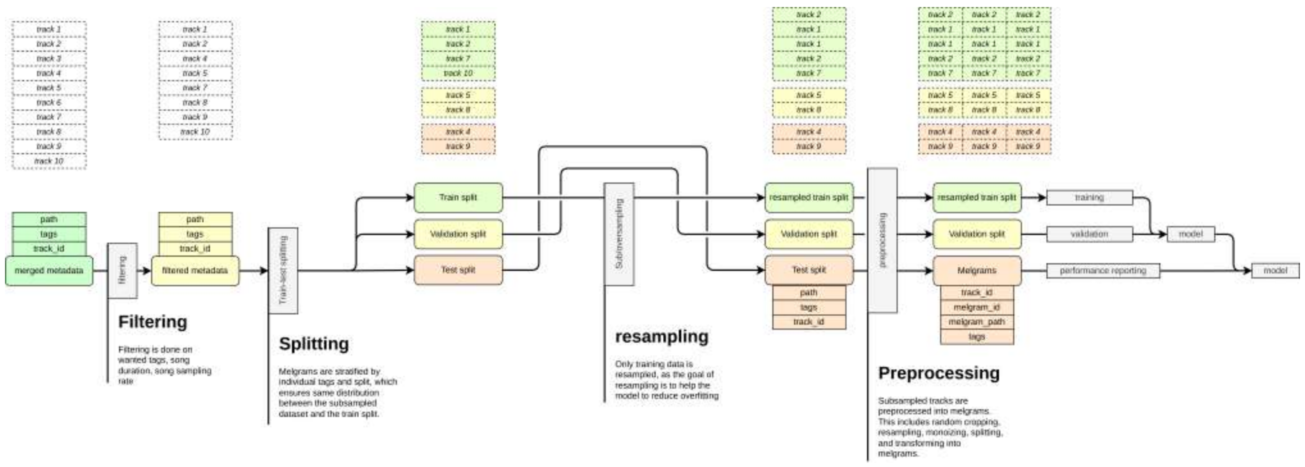
Melgrams are stratified by individual tags and split, which ensures same distribution between the subsampled dataset and the train split.

Since melgrams are already subsampled, there is already sample bias here. Furthermore, no checks are performed for song-level duplicates, only melgram duplicates.

Figure 8.2.: Full pipeline before refactoring for leaking and sampling bias

One can see how both the issues of sample bias and information leakage arises for the test and validation set.

We refactor the pipeline to adjust for this problem by moving the subsampling downstream and the train-test-validation split downstream. Furthermore, since we don't want to introduce sampling bias in the validation and test set, we only resample the training set. These changes lead to the following pipeline (8.3):



merged metadata
path
tags
track_id
filtering
filtered metadata
Train-test splitting
Train split
Validation split
Test split
Subsampling
resampled train split
Validation split
Test split
preprocessing
resampled train split
Validation split
Melgrams
track_id
melgram_id
melgram_path
tags
training
validation
performance reporting
model
model
Filtering
Filtering is done on wanted tags, song duration, song sampling rate
Splitting
Melgrams are stratified by individual tags and split, which ensures same distribution between the subsampled dataset and the train split.
resampling
Only training data is resampled, as the goal of resampling is to help the model to reduce overfitting
Preprocessing
Subsampled tracks are preprocessed into melgrams. This includes random cropping, resampling, monoizing, splitting, and transforming into melgrams.

Figure 8.3.: Full pipeline after refactoring for leaking and sampling bias

This major problem being fixed, a quick sanity check shows that after the refacto track-ids are now exclusive to either the train, test or validation sets, and that there is no overlap between the sets. we can also check the result of oversampling only the training set. If all goes well, the test and validation set should have about the same distribution of flattened genres, and the training set should be more equally split. This is shown in figure (8.4). The same graphs for the rest of the tag types (subgenres, moodthemes, alltags) are shown in annex B (B.2, B.3, 7.5d).

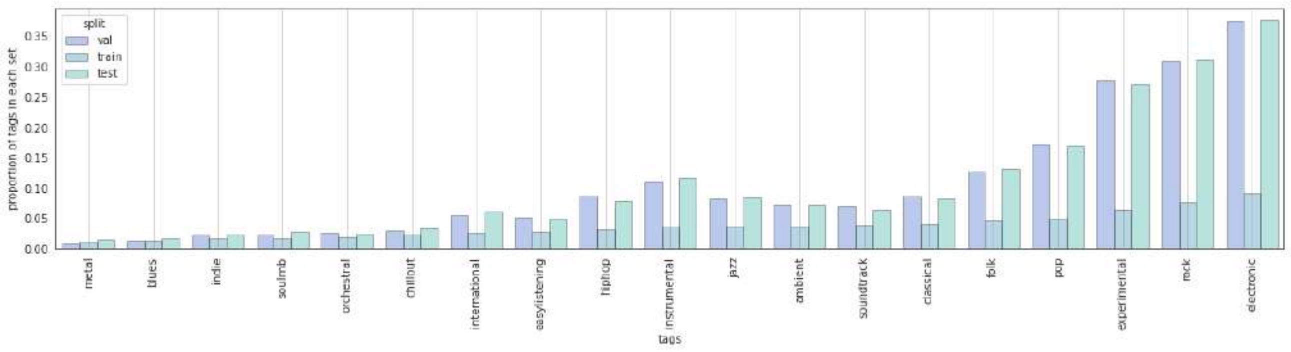


Figure 8.4.: Resampled genres split into train test and validation split. A more equal distribution can be seen in the training set, and an approximately equal distribution in the test and validation set, which is what was sought

9. Preliminary model selection

In section 4, we presented a total of 12 models and their variations which could serve towards the task of automatic music tagging. These models were of variable complexity, size, input length and availability. By availability, we mean that the implementation of some of these models is open-sourced, and some of them are even available pretrained online (e.g MusiCNN and SampleCNN) are available online. Furthermore, an additional constraint is that the tech stack for Groover is mainly based on TensorFlow [26], one of two main python packages for machine learning development along Pytorch [31]. The following criteria are taken into account when selecting the base models we will implement and test on our custom dataset:

- **Complexity of implementation**. If the model is highly complex to implement and not available in open-source implementation, it would take several weeks to implement and to verify its performance by reproducing results, where this work is a benchmark work rather than a results reproduction work.
- **Melgram input representation**. As a conscious choice to simplify the preprocessing pipeline as well as the wrangling pipeline, we choose to focus on models with melgram inputs as our main models. This is with the issue of disk space saving in mind as well as performance shown in state of the art over raw audio models
- **Low input dimension**. Corroborated by state of the art, small input lengths seem to perform better that long chunks (eg FCN and CRNN). This is beneficial from a space saving standpoint, so we choose to prefer models with lower length inputs
- **Open source availability**. This goes in hand with point 1, but an open source reproducible model is always better than rewriting it from scratch. If the model is pretrained, then bonus points.

With these criteria in mind, the following table recaps these criteria for each available model in the state of the art (*Note: Performance on state of the art datasets is important, but as shown in [51]), these performance metrics differ by small margins. As our task dataset has never been used before, and there is no benchmark on it, we remain conservative towards ousting models because they were beaten by a percentage point on a very specific task in state of the art*:

Model	*Input format*	*Input length*	*Open source?*	*pretrained?*	*Implemented in*
FCN	Melgrams	30s	Yes	No	Pytorch
CRNN	Melgrams	30s	Yes	No	Pytorch
MusicNN	Melgrams	3s	Yes	Yes	Pytorch / TF
HarmoniCNN	STFT	3s	Yes	No	Pytorch
SampleCNN	Audio	3s	Yes	No	Pytorch
Ensemble SampleCNN	Audio	3s	No	No	Pytorch
SampleCNN+SE	Audio	3s	Yes	No	Pytorch
ShortChunk	Melgrams	3s	Yes	No	Pytorch
ShortChunk ResNet	Melgrams	3s	Yes	No	Pytorch
CNSSA	Melgrams	15s	Yes	No	Pytorch
Semi-Supervised CNSSA	Melgrams	15s	No	No	Pytorch
SpecTNT	Melgrams	3s	No	No	Pytorch
CALM	Audio	-	Yes	No	Pytorch
CLMR	Audio	-	Yes	No	Pytorch

Table 9.1.: Preliminary model selection

Unfortunately, many models are not available in TensorFlow except for musicNN, which has a different input shape than our own data (due to slightly different input lengths). This and the non-sequential nature of MusiCNN prevents us from using the pretrained version in comparison with the other models. In the above table in green are shown the models that were implemented and trained on our dataset. In light yellow are shown the models that were either partially trained or implemented and not trained. In orange are shown CLMR and CALM, which as discussed previously show great promise in terms of reduction of storage space and latent representation learning, which are scheduled to be implemented before the end of the internship[1].

Note : CNSSA was implemented and was to be trained on all datasets. However, due to the different input length and shape, switching between datasets was time-consuming and preprocessing the whole dataset multiple times was not desirable for the obtention of results before the end of the internship. So, CNSSA was only trained on genres.

10. Model training and evaluation

10.1. training

This section focuses on the training of all the selected models. MusiCNN, Short-Chunk and ShortRes are trained on all tag types. CNSSA is trained only on genres. A first round of training on small subsets of the datasets was run to dial in our training hyperparameters as well as eventual dropout values to prevent overfitting. Naturally, this training on smaller subsets highly overfits but shows that the models are capable of learning the features for our custom task. The global training process, from first run to last, took about 3 months. In between each training run, back-and-forth considerations were had on preprocessing, input size, hyperparameters, etc, which explains the retraining of some models. The following figure (10.1) shows the amount of training runs undertaken for all models, as well as annotations of the different phases and models trained The recorded metric here is ROC-AUC on the train and validation sets.

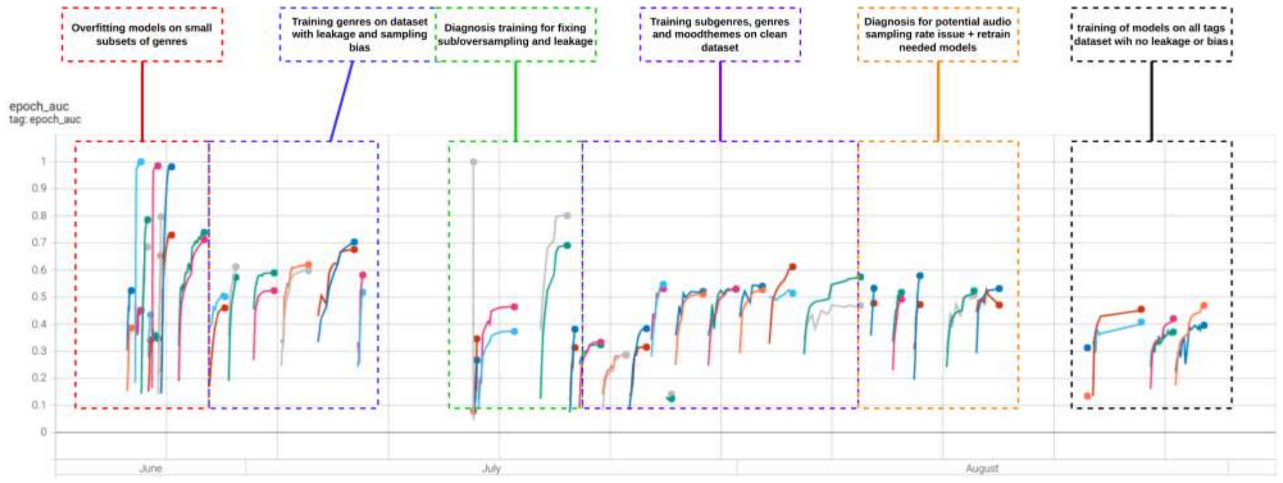


Figure 10.1.: All training runs with various training phases

The first phase allows us to dial in a convolutional dropout value of 0.1 for all convolutional front ends of the models, and per literature a 0.5 dropout layer is added between the dense layers at the end of each network. Phase 1 also allows us to dial in our training hyperparameters, which are the following :

```
training_config = {
"split": {"train_size": 0.85, "val_size": 0.1, "test_size": 0.05, "random_seed": 1},
"training": {"batch_size": 16, "epochs": 100, "steps_per_epoch": None,
"validation_steps": None,"optimizer": Adam(learning_rate=1e-4),"loss": BinaryCrossentropy(),
    "metrics": [AUC(curve="PR"), AUC(curve="ROC")],"max_queue_size": 2,
    "early_stop_patience": 4,"learning_rate_patience": 2,
}}
```

[1]All models which were implemented were implemented in Tensorflow either from scratch or using the pytorch formulations as a template to our own structure. These models can be found at this github repository

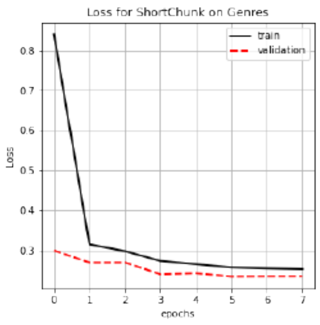
To prevent saddle point sticking, we use an adam optimizer with an original learning rate of 10^{-4}, and a learning rate scheduler which reduces the learning rate by a factor 10 every 2 epochs the validation loss does not decrease. This allows to not overshoot local minima due to gradient momentum. Per state of the art, our problem is a multilabel classification task, which requires a sigmoid activation function on the last layer of each model with and use of the binary cross-entropy loss function:

$$\mathcal{L}(y, gt) = \frac{-1}{N} \sum_{i=1}^{N} gt_i \log(y_i) + (1 - gt_i) \log(1 - y_i) \tag{10.1}$$

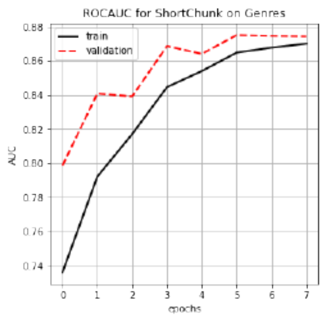
where gt_i is the ground truth label for each sample, and y_i is the model prediction for the given label. This frames our optimization problem. All models are trained for max 100 epochs with an early stopping callback that stops training when validation loss does not go down for 4 epochs. We use a batch size of 16, which is the max value allowed to not overblow the memory of the training GPU (which is a **RTX 2070 QMAX** on a personal laptop). Each model takes about 2 days to train, which explains why it is not possible to iterate endlessly on all these models. However, we manage to train all models in time except for moodthemes on ShortChunk. Here are shown training graphs (monitored on ROCAUC, PRAUC and Loss) for the training of all models on genres. Notice that no overfitting is shown here except for in one case - CNSSA.

ShortChunk

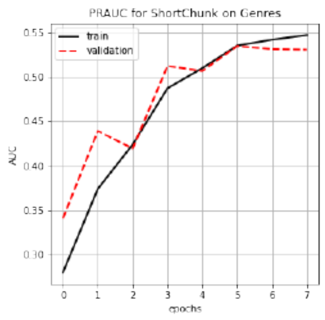
Training for ShortChunk on genres in shown in the following figures 10.2:



(a) Training loss for ShortChunk - genres



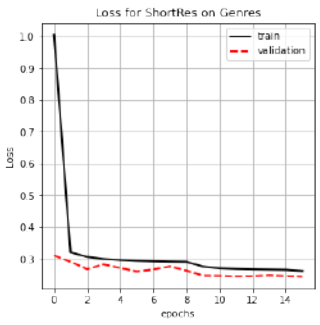
(b) Training ROC-AUC for Shortchunk - genres



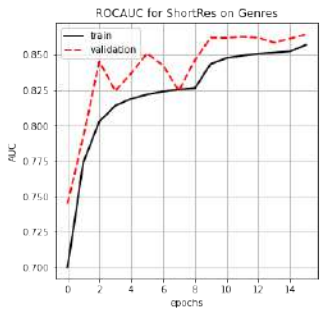
(c) Training PR-AUC for ShortChunk - genres

Figure 10.2.: Training - genres - ShortChunk

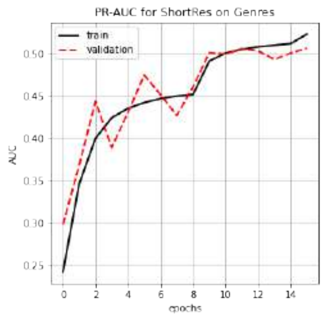
ShortRes



(a) Training loss for Shortres - genres



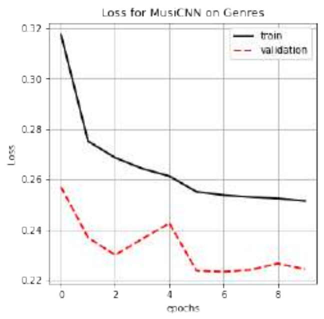
(b) Training ROC-AUC for Shortres - genres



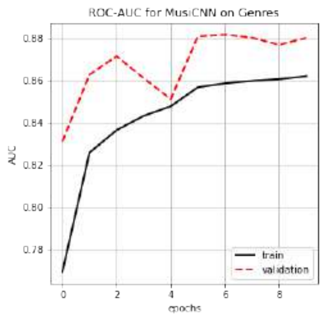
(c) Training PR-AUC for Shortres - genres

Figure 10.3.: Training - genres - Shortres

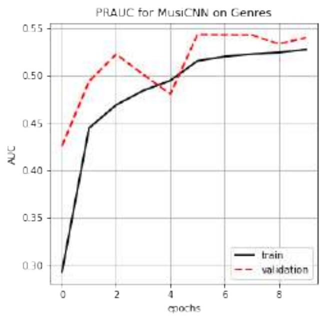
MusicNN



(a) Training loss for MusiCNN - genres



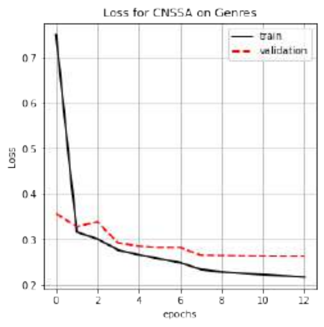
(b) Training ROC-AUC for MusiCNN - genres



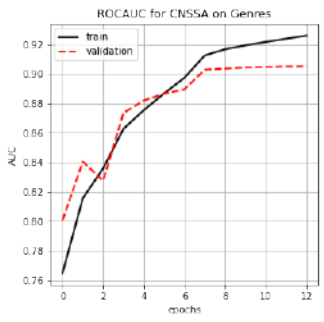
(c) Training PR-AUC for MusiCNN - genres

Figure 10.4.: Training - genres - MusiCNN

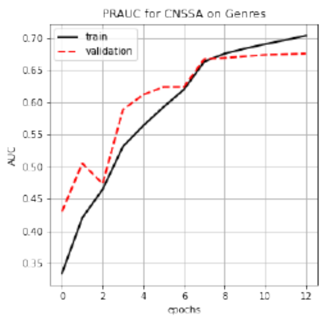
CNSSA



(a) Training loss for CNSSA - genres



(b) Training ROC-AUC for CNSSA - genres



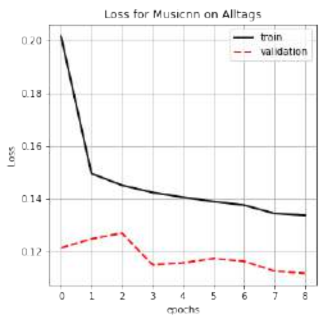
(c) Training PR-AUC for CNSSA - genres

Figure 10.5.: Training - genres - CNSSA

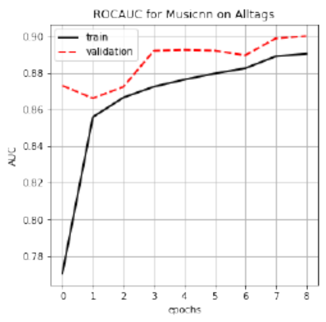
Note that the previous figures (10.5) show a high overfitting on the training set for CNSSA, made obvious by the crossing of the train and validation loss curves even though dropout and regularization is implemented. This as well as higher performance overall is due to the fact that CNSSA was trained with data leakage and sampling bias.

10.1.1. All tags

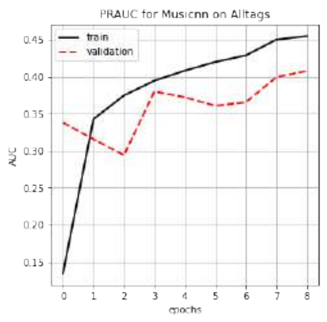
For the sake of brevity, training graphs for other models on the rest of the tag types are not shown here. We show here the training graphs on alltags as this is our final classification objective (Figures 10.6, 10.7, 10.8).



(a) Training loss for MusicNN - Alltags

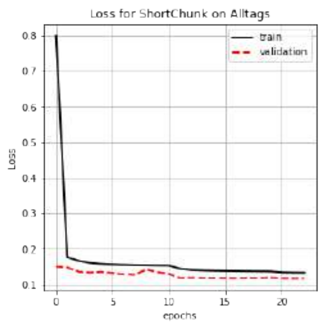


(b) Training ROC-AUC for MusicNN - Alltags

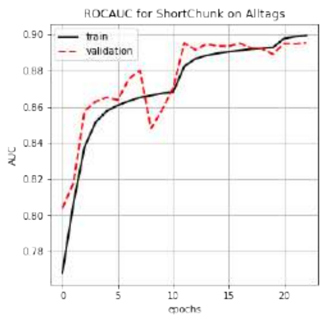


(c) Training PR-AUC for MusicNN - Alltags

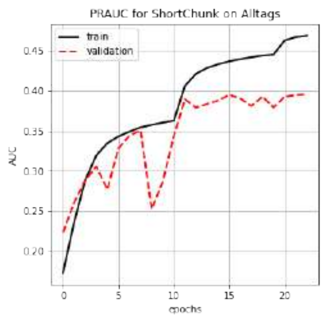
Figure 10.6.: Training - Alltags - MusicNN



(a) Training loss for Shortchunk - Alltags

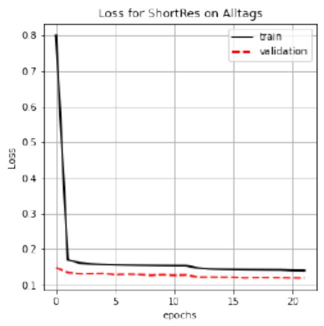


(b) Training ROC-AUC for Shortchunk - Alltags

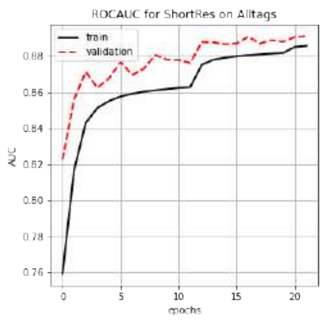


(c) Training PR-AUC for Shortchunk - Alltags

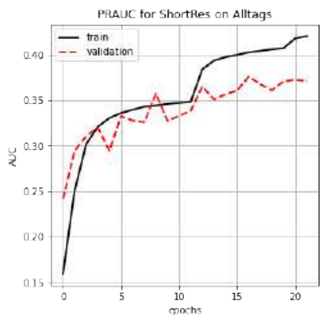
Figure 10.7.: Training - Alltags - Shortchunk



(a) Training loss for Shortres - Alltags



(b) Training ROC-AUC for Shortres - Alltags



(c) Training PR-AUC for Shortres - Alltags

Figure 10.8.: Training - Alltags - Shortres

In all these trainings, though no overfitting can be observed via loss monitoring, all models perform much better on the training set in terms of PR-AUC than on the validation set, while performance based on ROC-AUC remains relatively the same. This is also the case for other tag types with more labels : the higher the label count, the lower the performance of the models based on PR-AUC on the validation set. This hints at a need to relabel and clean the tags that are performing poorly on PR-AUC basis, which we will see in the next section.

10.2. Evaluation and visualization of model results

10.2.1. Key metrics and evaluation levels

In section 2.2.2 We described our macro metrics for music tagging evaluation, which are ROC-AUC, PR-AUC and average cosine similarity. These provide metrics on a global level, but do not distinguish performance between classes, which is important to determine which labels could be better labeled to be cleaner and/or oversampled for better representativity. To this end, we use per-class metrics to evaluate the performance of each model on each dataset:

- **No-average ROC-AUC and PR-AUC**. The global PRAUC and ROCAUC are computed by performing a micro average or macro average of the individual ROCAUC and PRAUC for each class. To visualize the performance of each class on the dataset, we also consider the non-averaged metrics (per-class ROCAUC and PRAUC)
- **Confusion matrices** are used to evaluate multiclass classification problems by comparing the number of true predictions vs for each class versus the actual prediction (see figure 10.9)

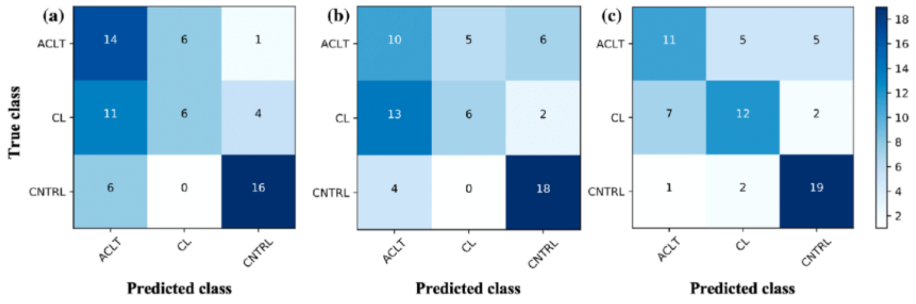


Figure 10.9.: Example of a traditional confusion matrix

This allows to evaluate which classes are missing the mark the most and more importantly which class they are missing the mark on. However, traditinal confusion matrices are best used on single-label multiclass classification problems due to the fact that each prediction can be singled down to a single label which is either correctly predicted or wrongly predicted. In our case, models output a score from 0 to 1 which is not aimed at isolating a single label but rather at ranking them. To solve this problem, we implement a custom confusion matrix in which each cell's value is the sum of predicted scores for that class which were meant for another class:

$$M_{i,j} = \sum_{k \in gt_{k,j}=1} y_{k,i} \tag{10.2}$$

We also normalize each column by the total score of the column, i.e the total score attributed to that class in predictions. So, the custom matrix can be read by column : "for the class *rock*, 25% of the total predicted scores in all the test dataset were attributed to rock, 5% to indie, etc.)

- Finally, we also include **mean Top-K accuracy (MTA_K) and mean soft Top-k accuracy in our test metrics** ($MSTA_K$), which are another custom metric. Top-K accuracy is considered a hit (1) if all ground truth labels are in the top K labels for the given prediction. It is a more harsh metric than soft top-K accuracy, which we consider a hit if at least one of the ground truth top tags is present in the top predictions.

$$MTA_K = \frac{1}{N}|y_i \in \{y_i | \forall k, gt_{i,k} = 1 \rightarrow y_{i,k} \in y_{topk}\}| \tag{10.3}$$

$$MSTA_K = \frac{1}{N}|y_i \in \{y_i | \exists k, gt_{i,k} = 1 \rightarrow y_{i,k} \in ytopk_i\}| \tag{10.4}$$

Where $ytopk_i$ is the k-truncated sorted vector of predictions for the ith sample, y_i is the prediction vector for the ith sample, gt_i is the ground truth label vector for the ith sample.

Multi Instance Learning and song-level evaluation

Previously, he have only discussed generating predictions on individual melgrams. As a matter of fact, all our models generate a prediction on one instance of a melgram, most of which generate predictions on 3s audio. However, depending on the tag, relevant characteristics are not necessarily predominant in the entire music recording. For example, when a song has a tag female vocal, it does not imply that the female vocal appears in every time segment of the song. In essence, this is a multiple instance problem [25]. A song can have many acoustic characteristics (instances) that describe it, but often only specific characteristics (instances) are responsible for the associated music tag. In most cases, we do not have time precise instance-level annotations because the precise labeling can be laborsome, therefore a music tag associated with a song is simply applied to all music excerpts (instances) of the song during training. There are two approaches to handle the multiple instance problem in music tagging. One is to train the model on full songs and produce song-level predictions from a songlevel input. From a given song-level input, the model has to predict relevant tags. Another one is to train the model on short audio chunks (instances) and generate chunk-level predictions, which can be later aggregated (e.g., majority vote, average) in the evaluation phase. We have already decided to conduct chunk-level training on our models, and it is now interesting to evaluate these models on whole songs.

To do this, we generate what is called a **taggram**, which represents the prediction distribution generated by the model on all the instances of the song. For taggram generation, a whole song is split into 3s segments with a given overlap which are fed into the model and the output matrix is a time-series of model predictions (this is shown in figure 10.10 for the track *The nights - Avicii.*

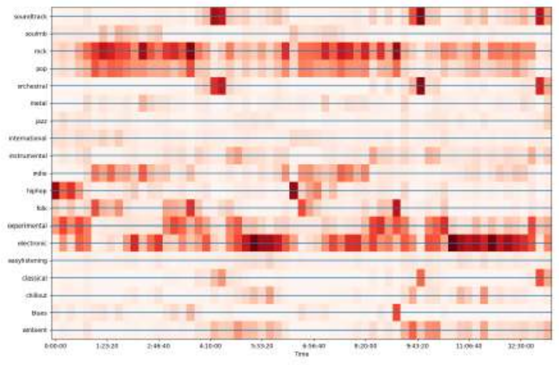


Figure 10.10.: Genre-taggram generated using ShortRes on the song *The nights - Avicii*

Though taggram provide the advantage of being interpretable - they show the evolution of the tags throughout the song : for instance here, the dance choruses are correctly labeled as electornic while the folk-rock intro and verses are labeled as rock, indie, folk - they do not perform song-level evaluations.

The solution is to pool the values of the taggram timewise, by using a pooling strategy (mean, softmax, majority voting...). As this has not been explored in literature yet in terms of the inlfuence of pooling strategy or hop length on music tagging tasks - studies mostly average the predictions into a global prediction - we evaluate all our models on the test dataset at the song level for all tagging types and report results depending on pooling strategy and instance overlap.

10.2.2. Chunk-level metrics

In this section, we focus on results for the melgram-level split for all models. As described in the previous metric section, these models were all evaluated based on all metrics. As their global performance is similar (as it was in the state of the art [51]), and for the sake of brevity, we do not show all the per-class metrics for each model for each tagging type but rather one comprehensive model report per tagging type, to exhibit the model behavior and potential next steps towards improving performance, as all models behave rather similarly and exhibit the same problems within the data.

Genres

Below is shown the performance report for MusicNN on the genres dataset (figure 10.11)

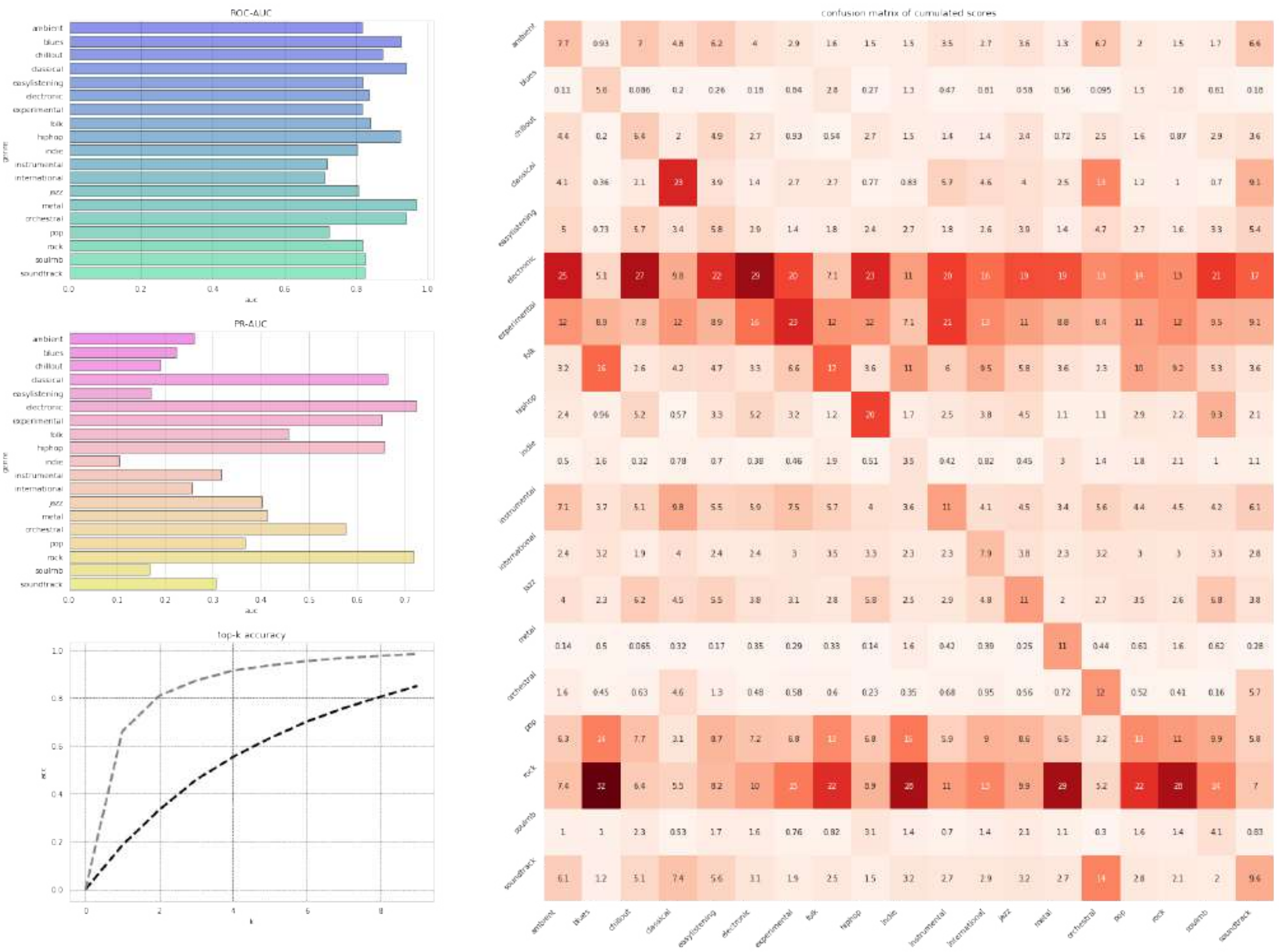


Figure 10.11.: Performance of MusicNN on the genres dataset - Chunk level

As can be seen, ROC-AUC performance is rather consistent across all genres. PR-AUC, however, which represents the hit rate of the model (how many relevant results are retrieved) varies wildly depending on the genre (tag). More specifically, by looking at the confusion matrix we see two horizontal bands that show that the model is skewed towards rock, electronic and pop. This corresponds to the genre distributions shown in the previous sections, and shows the need to consider per-genre resampling. It is reassuring to see, however, that there still is a diagonal of of correct predictions in the confusion matrix, showing that the model does learn relevant features. Furthermore, even though the model is skewed towards overrepresented tags, it makes human-coherent mistakes, such as often confusing *rock* with *indie* or *orchestral* with *sountrack*.

Looking at results on unseen data, we test out the model on the song *Test drive*, an orchestral soundtrack from the pixar animation film *how to train your dragon*. The results in the form of a taggram are shown in figure 10.12:

The model captures the top tags for the song perfectly, and even shows a bridge transition towards the middle of the song with an experimental feel to it. This human validation is corroborated by accuracy at K and soft

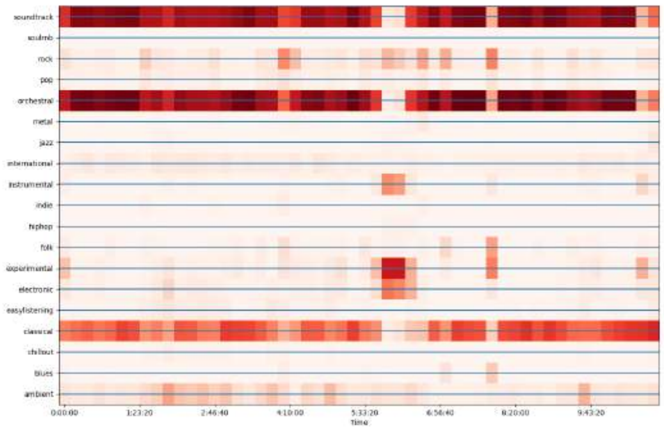


Figure 10.12.: taggram results of shortRes on *Test drive - John Powell*

accuracy at K that show that as early as 5 tags in, at least one of the relevant tags will be present in the results with 95% certainty.

Melgram-level results on the other tag types

In the following figures are shown the results obtained using the best musiCNN model on other tagging types. Though not as clear as in the genres section, the classification metrics and confusion matrix consistently show the model is skewed towards popular tags, and exhibit the same behavioral biases, thus also exhibit a clear road towards performance bettering. Fixing this goes through either data augmentation, attributing lighter wieght to these tags during training, or undersampling these tags even more, specifically on single-label samples. Figure 10.13 shown the performance report for musicnn on subgenres, 10.14 on all tags and 10.15 on moodtheme:

Global results on the melgram split

Results for all models on a standardized split are shown in the table below (10.1):

tags	genres			subgenres			moods/themes			Alltags		
model	**PR**	**ROC**	**Cosim**	**PR**	**ROC**	**Cosim**	**PR**	**ROC**	**Cosim**	**PR**	**ROC**	**Cosim**
Short	0.389	0.829	0.3	0.16	0.79	0.13	0.291	0.810	0.22	0.258	0.799	0.22
ShortRes	0.393	0.831	0.299	0.15	0.79	0.15	**0.31**	**0.819**	**0.26**	0.245	0.801	0.23
MusiCNN	**0.401**	**0.838**	**0.307**	**0.32**	**0.87**	**0.1**	0.26	0.799	0.25	**0.31**	**0.819**	**0.26**

Table 10.1.: Global results for melgram-level evaluation

As intuited in [51], musicnn seems to be a better performer across the board except in the case of moodtheme (we suspect musicnn was trained with an erroneous sample rate when compared to the evaluation sample rate, which could explain the performance delta. On the whole, as shown by the previous figures of class-level behaviour, there is a need for rebalancing classes and potentially exploring single-label data augmentation (or at least minimizing the oversampling of majoritary classes such as *rock* or *electronic*. Overall, results are close to what could be expected based on the state of the art. As our training was on a new dataset, which was potentially noisier and less comprehensive in terms of labels and data quality, and since we did not have time to fine-tune these models as much as possible, it is to be expected that our performance be slightly worse. Not to mention that our tagging tasks were different from those in the literature, and even the alltags dataset had 80 tags in the place of 50 in all papers.

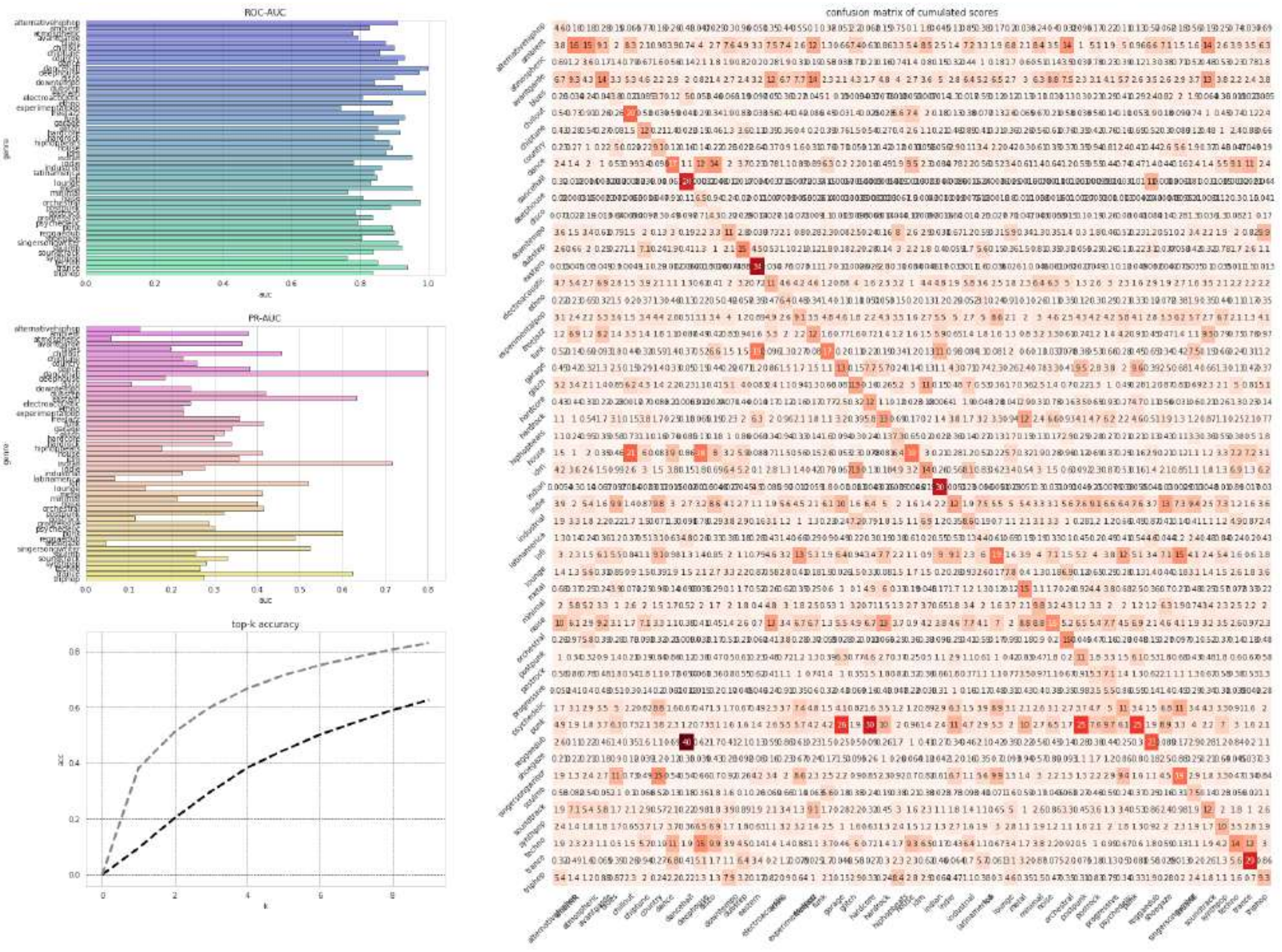


Figure 10.13.: Performance of MusicNN on the subgenres dataset - Chunk level

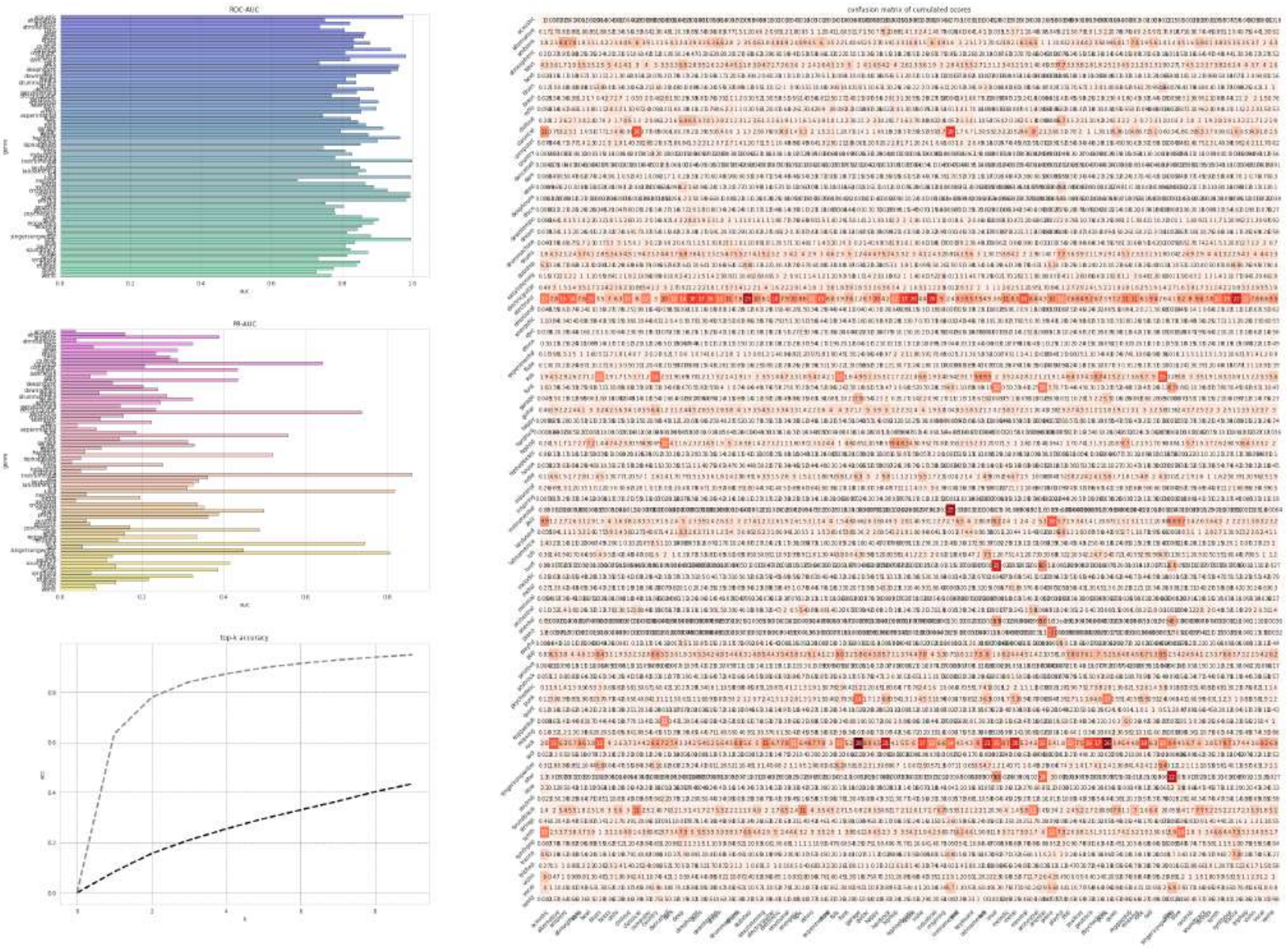


Figure 10.14.: Performance of MusicNN on the alltags dataset - Chunk level

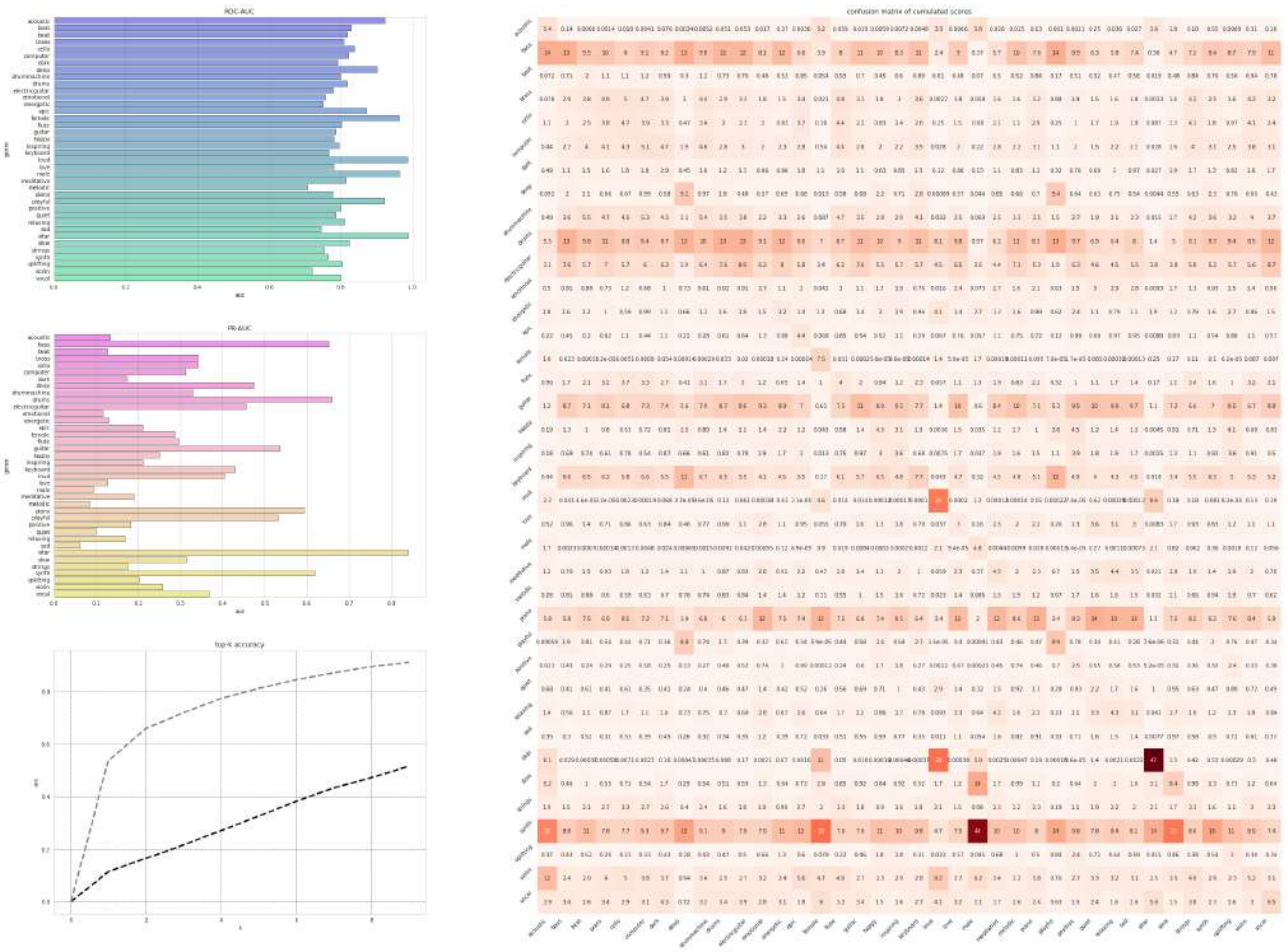


Figure 10.15.: Performance of MusicNN on the moodtheme dataset - Chunk level

Song-level evaluation As described in section 10.2.1, We also evluate the trained models on song-level samples, which allows us to compare CNSSA and other models with various input representation lengths. Taggrams are generated with no overlap between instances (e.g a 21s song is split into 7 3s chunks to be fed to the models and generate a melgram for), and are pooled into a final classification vector using **Timewise average, time-wise majority voting and timewise softmax**. The formula for the softmax activation of a vector is shown below:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{10.5}$$

Which converts vector z into a probability distribution with high discrimination towards low values (low values are rendered very low and high values are close to 1). This type of activation layer is often used in single label classification tasks. Results are reported in table 10.2 for all tag types and pooling methods

genres

Method	Mean			Majority vote			Softmax		
	PRAUC	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**
Short	0.399	0.837	0.6	0.35	0.67	0.33	0.281	0.558	0.403
ShortRes	0.441	0.861	0.618	0.39	0.66	0.35	0.316	0.594	0.499
MusiCNN	**0.448**	**0.876**	**0.634**	0.41	0.72	0.46	0.311	0.608	0.553
CNSSA	0.46	0.89	0.7	0.38	0.73	0.41	0.339	0.621	0.538

subgenres

Method	Mean			Majority vote			Softmax		
	PRAUC	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**
Short	0.233	0.827	0.358	0.238	0.712	0.319	0.19	0.637	0.29
ShortRes	0.266	0.831	0.381	0.249	0.751	0.336	0.19	0.641	0.289
MusiCNN	**0.28**	**0.849**	**0.388**	**0.251**	0.77	0.35	0.21	0.679	0.33

Moodtheme

Method	Mean			Majority vote			Softmax		
	PRAUC	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**
Short	-	-	-	-	-	-	-	-	-
ShortRes	**0.34**	**0.821**	**0.28**	**0.26**	0.731	0.212	0.251	0.632	0.233
MusiCNN	0.31	0.797	0.26	0.253	0.715	0.211	0.236	0.611	0.

Alltag

Method	Mean			Majority vote			Softmax		
	PRAUC	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**
Short	0.26	0.809	0.23	0.233	0.71	0.18	0.245	0.66	0.16
ShortRes	0.291	0.811	0.25	0.258	0. 721	0.2	0.278	0.692	0.2
MusiCNN	0.312	0.823	0.26	0.261	0.729	0.21	0.285	0.691	0.2

Table 10.2.: Song level evaluation

Again, it is shown that MusiCNN substantially outperforms other models on out small dataset, except for CNSSA (which, recall, was trained on the leaky dataset and thus is overestimated in terms of performance). It is also shown that mean pooling is by far the best method for song-level estimation and no other method comes close in terms of performance. Thus, We keep mean pooling as a future reference for implementation at scale. This is also good for artists with multiple tracks as it allows to create an average tag representation over all tracks for the artist without being incoherent with our song level predictions.

11. Future considerations

At the time of this report, All selected models have been trained and evaluated on their respective splits for multiple tagging types. However, some tasks still have to be addressed before moving on to productionizing an audio tagging model:

1. **Firstly, all models still have to be evaluated on a song-level split**, as discussed in section **10.2.1**. This will allow us to not only consider both levels of evaluation to select the final model but also to consider various pooling methods and their outcomes in model performance on our song-level test sets. This will surely happen within the time frame of the remainder of the internship.

2. **The selection and fine-tuning of the final model is an essential task**, but will probably not have the time to occur before the end of the internship - at least the fine-tuning aspect. Fine-tuning requires testing multiple hyper-parameters towards getting better performance. These can be preprocessing hyper-parameters, training hyper-parameters, etc. As each training takes about a day, conducting a grid search is very time-costly and should be approached with strategy. Our training strategy described in Section **10.1** potentially alleviates the need for many steps in the grid search for training parameters as it is generally considered to be optimal in state of the art, but this still remains a lengthy next step.

3. **Integrating the model's predictions into the recommender system at a local level** before bringing into production to check performance gain in also a necessary step, and will potentially start before the end of the internship, but not be finished.

4. **Another bottleneck in model training and iteration, as well as inference, is differed preprocessing**. Currently, as described in section **9**, all preprocessing is done prior to ingestion into the model. This leads to long loading times at inference for whole songs (close to 10s using librosa for 3min long songs), and adds an additional 10hrs of preprocessing to the process when training a new model. Re-implementing the trained models with on-board preprocessing with KAPRE (which is coherent with librosa results) would be a nice step towards reducing the global clunkiness of the process and to prevent long inference times in production.

5. Finally, Many models have been put to the side during the preliminary model selection round described in Section **8**. Specifically, HarmoniCNN [49], [22], CALM [4], and CLMR [36] show strong promise based on state of the art result and the attractiveness of not having to conduct preprocessing at inference time. Implementing these models and refactoring the pipeline to include them into the models trained in this report is a good idea and could possibly lead to better models in terms of performance and more elegant preprocessing steps in the production pipeline. Note also that some techniques were not explored for the models at this point in the internship. Ensemble voting, multi-level and multi scale techniques [25] [20] [21] have shown to increase model performance in state of the art, and could be an easy addition to some of the models implemented here.

   Then, of course, comes the implementation of the model into the production pipeline-the groover website itself. Though this will probably come at a later time, outside of the scope of this internship, it will require adapting the models and working towards a globally faster preprocessing and inference pipeline for better user interaction.

12. Conclusion

The goal of this internship, rooted in the need to elaborate upon an existing recommender system by implementing at-scale audio tagging, was to explore and benchmark state of the art music audio preprocessing techniques and machine learning models towards automatic music tagging.

Through extensive research on audio processing state of the art, we selected appropriate models to train on a dataset that we built ourselves, adapted to our custom task of music tagging at Groover. Through careful selection and consideration of constraints due to length of the internship, implementation complexity, and hardware limitations, we selected appropriate models and trained them on our proprietary dataset. Our trained models performed well, on par with state of the art with room for improvement based on fine-tuning considerations and low quality of data. Furthermore, we exhibited the interest of machine learning music tagging models as learned preprocessing representation extractors, and stipulate that training said models on task-agnostic prediction (such as CALM or CLMR) can lead to learned fine-tunable representation that can serve for other more generalist acoustic tasks (dereverberation, active noise control, acoustic scene classification, etc). We argue that these machine learning models, in and of themselves, constitute new preprocessing algorithms that have the potential to be adapted specifically to the task they aim to solve.

Finally, we showed that coherent and realizable next steps are to be taken before the end of the internship, towards bettering the already trained models, implementing new ones, and pushing our selected trained model, MusiCNN, to production. There is still some work left within the context of the internship, and perhaps some interpretability exploration of these models as representation learners will further showcase these intuitions as interesting next steps for potential research or future exploration.

List of Figures

List of Tables

Bibliography

[1] M. M.Afsar, T.Crump, and B.Far. "Reinforcement learning based recommender systems: A survey". In: *ACM Computing Surveys (CSUR)* (2021).

[2] J. H.Appleton. "The State of Electronic Music: 1972". In: *College Music Symposium*. JSTOR. 1972, pp. 7–10.

[3] D.Bogdanov, M.Won, P.Tovstogan, A.Porter, and X.Serra. "The MTG-Jamendo Dataset for Automatic Music Tagging". In: *Machine Learning for Music Discovery Workshop, International Conference on Machine Learning (ICML 2019)*. Long Beach, CA, United States, 2019. URL: `http://hdl.handle.net/10230/42015`.

[4] R.Castellon, C.Donahue, and P.Liang. *Codified audio language modeling learns useful representations for music information retrieval*. 2021. DOI: `10.48550/ARXIV.2107.05677`. URL: `https://arxiv.org/abs/2107.05677`.

[5] Q.Chen, H.Zhao, W.Li, P.Huang, and W.Ou. "Behavior sequence transformer for e-commerce recommendation in alibaba". In: *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*. 2019, pp. 1–4.

[6] K.Choi, G.Fazekas, and M.Sandler. "Automatic tagging using deep convolutional neural networks". In: *arXiv preprint arXiv:1606.00298* (2016).

[7] K.Choi, G.Fazekas, M.Sandler, and K.Cho. "A comparison of audio signal preprocessing methods for deep neural networks on music tagging". In: *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE. 2018, pp. 1870–1874.

[8] K.Choi, G.Fazekas, M.Sandler, and K.Cho. "Convolutional recurrent neural networks for music classification". In: *2017 IEEE International conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2017, pp. 2392–2396.

[9] K.Choi, D.Joo, and J.Kim. "Kapre: On-GPU Audio Preprocessing Layers for a Quick Implementation of Deep Neural Network Models with Keras". In: *Machine Learning for Music Discovery Workshop at 34th International Conference on Machine Learning*. ICML. 2017.

[10] M.Defferrard, K.Benzi, P.Vandergheynst, and X.Bresson. "FMA: A Dataset for Music Analysis". In: *18th International Society for Music Information Retrieval Conference (ISMIR)*. 2017. arXiv: `1612.01840`. URL: `https://arxiv.org/abs/1612.01840`.

[11] S.Dieleman and B.Schrauwen. "End-to-end learning for music audio". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 6964–6968.

[12] D.Gabor. "Theory of communication. Part 1: The analysis of information". In: *Journal of the Institution of Electrical Engineers-part III: radio and communication engineering* 93.26 (1946), pp. 429–441.

[13] A. A.Gunawan, D.Suhartono, etal. "Music recommender system based on genre using convolutional recurrent neural networks". In: *Procedia Computer Science* 157 (2019), pp. 99–109.

[14] X.He, L.Liao, H.Zhang, L.Nie, X.Hu, and T.-S.Chua. "Neural collaborative filtering". In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 173–182.

[15] B.Hidasi, A.Karatzoglou, L.Baltrunas, and D.Tikk. "Session-based recommendations with recurrent neural networks". In: *arXiv preprint arXiv:1511.06939* (2015).

[16] K.Järvelin and J.Kekäläinen. "Discounted Cumulated Gain". In: *Encyclopedia of Database Systems*. Ed. by L.LIU and M. T.ÖZSU. Boston, MA: Springer US, 2009, pp. 849–853. ISBN: 978-0-387-39940-9. DOI: `10.1007/978-0-387-39940-9_478`. URL: `https://doi.org/10.1007/978-0-387-39940-9_478`.

[17] T.Kim, J.Lee, and J.Nam. "Sample-level CNN architectures for music auto-tagging using raw waveforms". In: *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2018, pp. 366–370.

[18] Y.-J.Ko, L.Maystre, and M.Grossglauser. “Collaborative recurrent neural networks for dynamic recommender systems”. In: *Asian Conference on Machine Learning*. PMLR. 2016, pp. 366–381.

[19] E.Law, K.West, M. I.Mandel, M.Bay, and J. S.Downie. “Evaluation of algorithms using games: The case of music tagging.” In: *ISMIR*. 2009, pp. 387–392.

[20] J.Lee and J.Nam. “Multi-level and multi-scale feature aggregation using pretrained convolutional neural networks for music auto-tagging”. In: *IEEE signal processing letters* 24.8 (2017), pp. 1208–1212.

[21] J.Lee and J.Nam. “Multi-level and multi-scale feature aggregation using sample-level deep convolutional neural networks for music classification”. In: *arXiv preprint arXiv:1706.06810* (2017).

[22] J.Lee, J.Park, K. L.Kim, and J.Nam. “Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms”. In: *arXiv preprint arXiv:1703.01789* (2017).

[23] Z.Liu, Y.Chen, J.Li, P. S.Yu, J.McAuley, and C.Xiong. “Contrastive self-supervised sequential recommendation with robust augmentation”. In: *arXiv preprint arXiv:2108.06479* (2021).

[24] W.-T.Lu, J.-C.Wang, M.Won, K.Choi, and X.Song. “SpecTNT: A time-frequency transformer for music audio”. In: *arXiv preprint arXiv:2110.09127* (2021).

[25] M. I.Mandel and D. P.Ellis. “Multiple-instance learning for music information retrieval”. In: (2008).

[26] MartínAbadi, AshishAgarwal, PaulBarham, EugeneBrevdo, ZhifengChen, CraigCitro, GregS. Corrado, AndyDavis, JeffreyDean, MatthieuDevin, SanjayGhemawat, IanGoodfellow, AndrewHarp, GeoffreyIrving, MichaelIsard, Y.Jia, RafalJozefowicz, LukaszKaiser, ManjunathKudlur, JoshLevenberg, DandelionMané, RajatMonga, SherryMoore, DerekMurray, ChrisOlah, MikeSchuster, JonathonShlens, BenoitSteiner, Ilya Sutskever, KunalTalwar, PaulTucker, VincentVanhoucke, VijayVasudevan, FernandaViégas, OriolVinyals, PeteWarden, MartinWattenberg, MartinWicke, YuanYu, and XiaoqiangZheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[27] B.McFee, A.Metsai, M.McVicar, S.Balke, C.Thomé, C.Raffel, F.Zalkow, A.Malek, Dana, K.Lee, O.Nieto, D.Ellis, J.Mason, E.Battenberg, S.Seyfarth, R.Yamamoto, viktorandreevichmorozov, K.Choi, J.Moore, R. Bittner, S.Hidaka, Z.Wei, nullmightybofo, A.Weiss, D.Hereñú, F.-R.Stöter, L.Nickel, P.Friesch, M.Vollrath, and T.Kim. *librosa/librosa: 0.9.2*. Version 0.9.2. June 2022. DOI: `10.5281/zenodo.6759664`. URL: `https://doi.org/10.5281/zenodo.6759664`.

[28] I.Munemasa, Y.Tomomatsu, K.Hayashi, and T.Takagi. “Deep reinforcement learning for recommender systems”. In: *2018 International Conference on Information and Communications Technology (ICOIACT)*. 2018, pp. 226–233. DOI: `10.1109/ICOIACT.2018.8350761`.

[29] A. V.Oppenheim, J. R.Buck, and R. W.Schafer. *Discrete-time signal processing. Vol. 2*. Upper Saddle River, NJ: Prentice Hall, 2001.

[30] K.Palanisamy, D.Singhania, and A.Yao. “Rethinking CNN models for audio classification”. In: *arXiv preprint arXiv:2007.11154* (2020).

[31] A.Paszke, S.Gross, F.Massa, A.Lerer, J.Bradbury, G.Chanan, T.Killeen, Z.Lin, N.Gimelshein, L.Antiga, A.Desmaison, A.Kopf, E.Yang, Z.DeVito, M.Raison, A.Tejani, S.Chilamkurthy, B.Steiner, L.Fang, J.Bai, and S.Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[32] J.Pons, O.Nieto, M.Prockup, E.Schmidt, A.Ehmann, and X.Serra. “End-to-end learning for music audio tagging at scale”. In: *arXiv preprint arXiv:1711.02520* (2017).

[33] B.Rocca. *Introduction to recommender systems*. `https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada`. June 2019.

[34] J.Salamon, C.Jacoby, and J. P.Bello. “A Dataset and Taxonomy for Urban Sound Research”. In: *22nd ACM International Conference on Multimedia (ACM-MM’14)*. Orlando, FL, USA, Nov. 2014, pp. 1041–1044.

[35] E.Smirnova and F.Vasile. “Contextual sequence modeling for recommendation with recurrent neural networks”. In: *Proceedings of the 2nd workshop on deep learning for recommender systems*. 2017, pp. 2–9.

[36] J.Spijkervet and J. A.Burgoyne. *Contrastive Learning of Musical Representations*. 2021. DOI: `10.48550/ARXIV.2103.09410`. URL: `https://arxiv.org/abs/2103.09410`.

[37] J.Spijkervet and J. A.Burgoyne. "Contrastive learning of musical representations". In: *arXiv preprint arXiv:2103.09410* (2021).

[38] F.Sun, J.Liu, J.Wu, C.Pei, X.Lin, W.Ou, and P.Jiang. "BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer". In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 1441–1450.

[39] J.Tang and K.Wang. "Personalized top-n sequential recommendation via convolutional sequence embedding". In: *Proceedings of the eleventh ACM international conference on web search and data mining*. 2018, pp. 565–573.

[40] G.Tzanetakis and P.Cook. "Musical genre classification of audio signals". In: *IEEE Transactions on speech and audio processing* 10.5 (2002), pp. 293–302.

[41] G.Tzanetakis, G.Essl, and P.Cook. *Automatic Musical Genre Classification Of Audio Signals*. 2001. URL: `http://ismir2001.ismir.net/pdf/tzanetakis.pdf`.

[42] A.Usman, M.Rafiq, M.Saeed, A.Nauman, A.Almqvist, and M.Liwicki. "Machine learning computational fluid dynamics". In: *2021 Swedish Artificial Intelligence Society Workshop (SAIS)*. IEEE. 2021, pp. 1–4.

[43] L. R.Varshney and J. Z.Sun. "Why do we perceive logarithmically?" In: *Significance* 10.1 (2013), pp. 28–31.

[44] A.Vaswani, N.Shazeer, N.Parmar, J.Uszkoreit, L.Jones, A. N.Gomez, L.Kaiser, and I.Polosukhin. *Attention Is All You Need*. 2017. DOI: `10.48550/ARXIV.1706.03762`. URL: `https://arxiv.org/abs/1706.03762`.

[45] H.Wang, N.Wang, and D.-Y.Yeung. *Collaborative Deep Learning for Recommender Systems*. 2014. DOI: `10.48550/ARXIV.1409.2944`. URL: `https://arxiv.org/abs/1409.2944`.

[46] S.Wang, L.Cao, Y.Wang, Q. Z.Sheng, M. A.Orgun, and D.Lian. "A survey on session-based recommender systems". In: *ACM Computing Surveys (CSUR)* 54.7 (2021), pp. 1–38.

[47] *What are frequency weighted filters such as a, C and Z weighting?* `https://www.castlegroup.co.uk/frequency-weighted-filter/`. Apr. 2018.

[48] M.Won, K.Choi, and X.Serra. "Semi-supervised music tagging transformer". In: *arXiv preprint arXiv:2111.13457* (2021).

[49] M.Won, S.Chun, O.Nieto, and X.Serrc. "Data-driven harmonic filters for audio representation learning". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 536–540.

[50] M.Won, S.Chun, and X.Serra. "Toward interpretable music tagging with self-attention". In: *arXiv preprint arXiv:1906.04972* (2019).

[51] M.Won, A.Ferraro, D.Bogdanov, and X.Serra. "Evaluation of CNN-based automatic music tagging models". In: *arXiv preprint arXiv:2006.00751* (2020).

[52] M.Won, J.Spijkervet, and K.Choi. "Music Classification: Beyond Supervised Learning, Towards Real-world Applications". In: *arXiv preprint arXiv:2111.11636* (2021).

[53] S.Wu, Y.Tang, Y.Zhu, L.Wang, X.Xie, and T.Tan. "Session-based recommendation with graph neural networks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 346–353.

[54] X.Xin, A.Karatzoglou, I.Arapakis, and J. M.Jose. "Self-supervised reinforcement learning for recommender systems". In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2020, pp. 931–940.

[55] Y.-Y.Yang, M.Hira, Z.Ni, A.Chourdia, A.Astafurov, C.Chen, C.-F.Yeh, C.Puhrsch, D.Pollack, D.Genzel, D.Greenberg, E. Z.Yang, J.Lian, J.Mahadeokar, J.Hwang, J.Chen, P.Goldsborough, P.Roy, S.Narenthiran, S.Watanabe, S.Chintala, V.Quenneville-Bélair, and Y.Shi. "TorchAudio: Building Blocks for Audio and Speech Processing". In: *arXiv preprint arXiv:2110.15018* (2021).

A. Other spectral features

A.1. Constant Q melgram

constant Q transform is a transformation into the frequency domain from the time domain than can be though of as a series of filters centered around frequencies which are logarithmically spaced between one another, and the spectral width of which is a multiple of the previous filter.

$$\delta f_k = 2^{1/n} \delta f_{k-1}$$

The constant Q transform alleviates the time-frequency trade-off through this approach, and thus provides a higher time resolution for mel-spectrograms , notably in the lower-end of the spectrum. (as seen below in figure A.1):

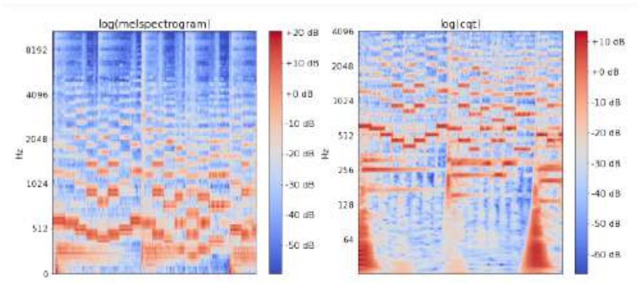


Figure A.1.: Constant Q transform [52]

Despite this resolution gain, not many mentions of their can be found in literature. thus far the CQT has mainly been used when precise frequency resolution has to be obtained (chord and note classification) [30].

A.2. Chroma variants

A chromagram is the equivalent of a spectrogram for which the frequencies are binned within a given note system (librosa includes the indian music system and the 12-note equal temper western music system). Possible variants are chroma CQT and chroma energy normalized scale (CENS), which is more resistant to dynamics, timbre and articulation [27]

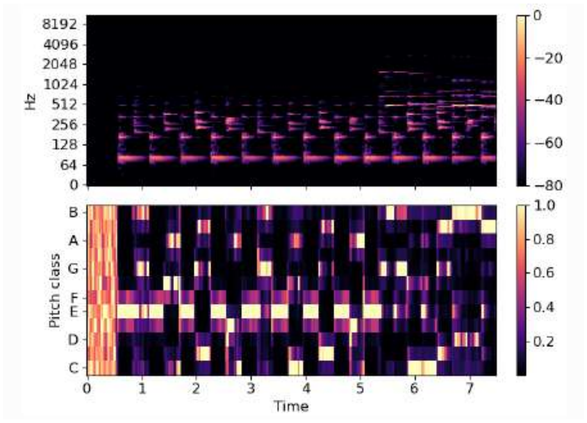
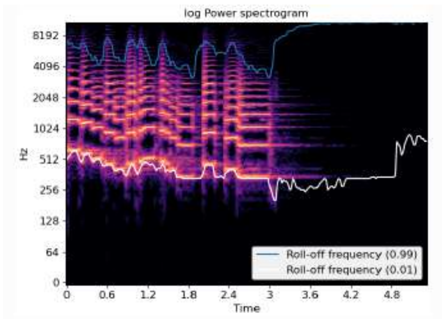


Figure A.2.: Chroma variants [27]

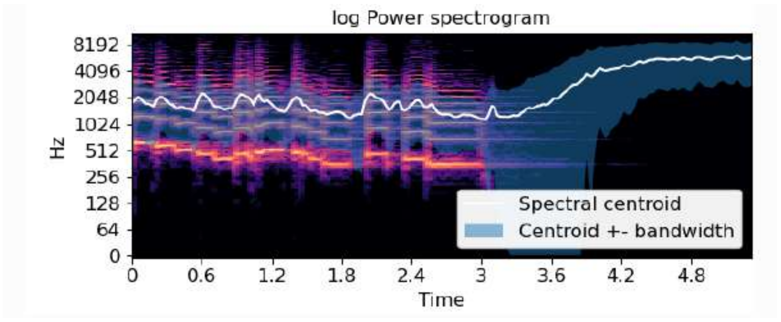
All of these are available in librosa but not in Kapre. For music tagging this is not necessarily a valuable asset (perhaps for key detection but pre-trained models exist for that).

A.3. Spectral Envelopes

Other spectral features are available, such as bandwidth, roll off, contrast, some of which are shown below (A.3a A.3b):



(a) Spectral rolloff



(b) Spectral Centroid and bandwidth

Though useful in classification where spectral global components are key to identification (for instance instrument classification, sound event classification [30]), these representations are not particularly useful as representations of full musical pieces and are not used in any of the papers cited in this report. Most of these features are not musically interpretable but provide good physical representations of the audio clip. Though no studies mention their use as input features of a model, they can be good estimators of mixing and mastering jargon such as 'bright','warm', etc.

A.4. Tempogram

Tempograms and Fourier tempograms are used to estimate the bpm of a track by calculating autocorrelation and estimating onset for the signal. The tempogram itself as an input feature doesn't necessarily serve any purpose. However, the tempo estimation for the track can provide a good estimation of track energy, along with things such as track LUFs, true peak maximum dB, etc. Below is shown a tempogram for reference purposes (A.4:

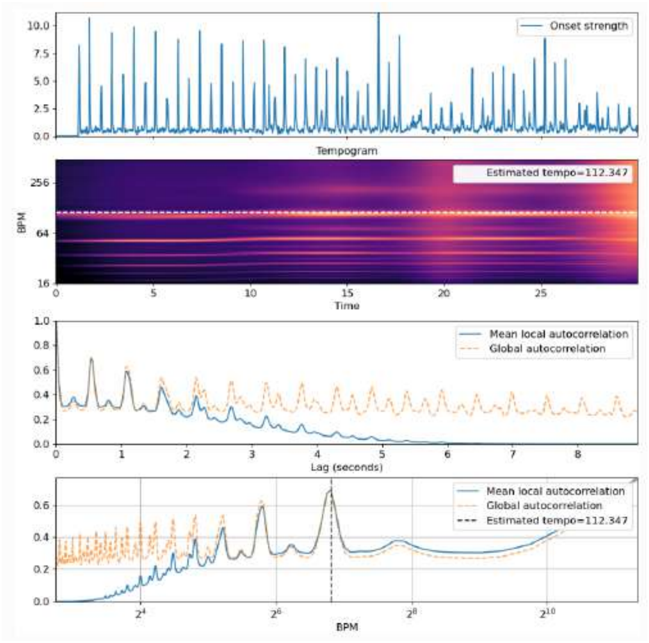
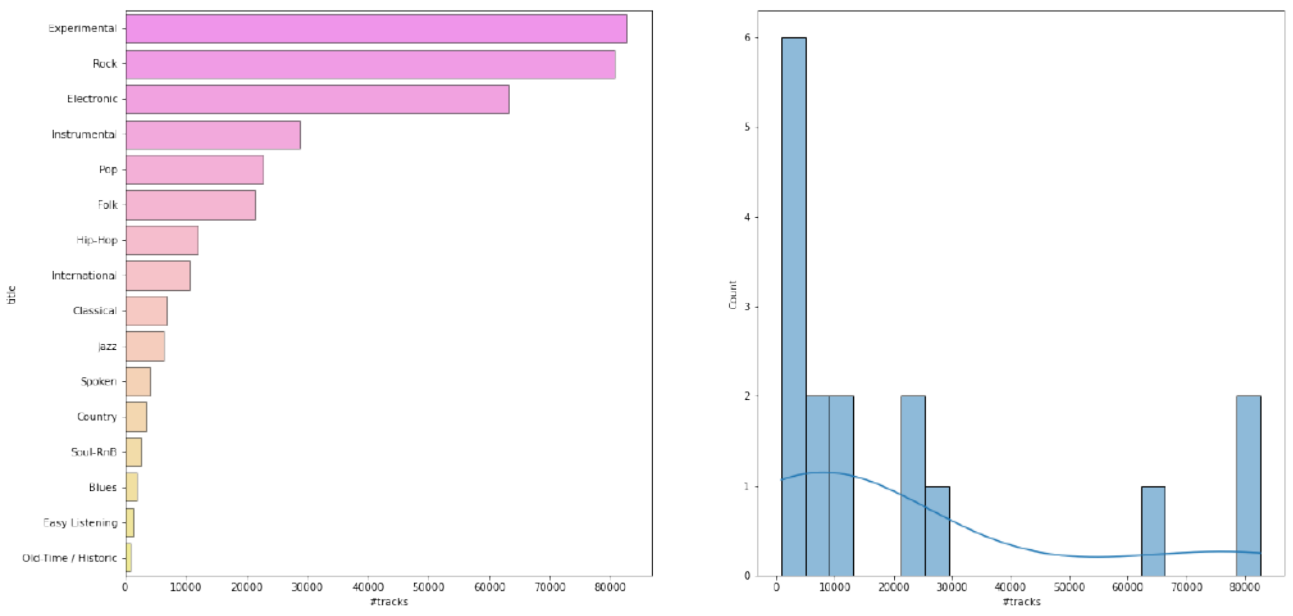


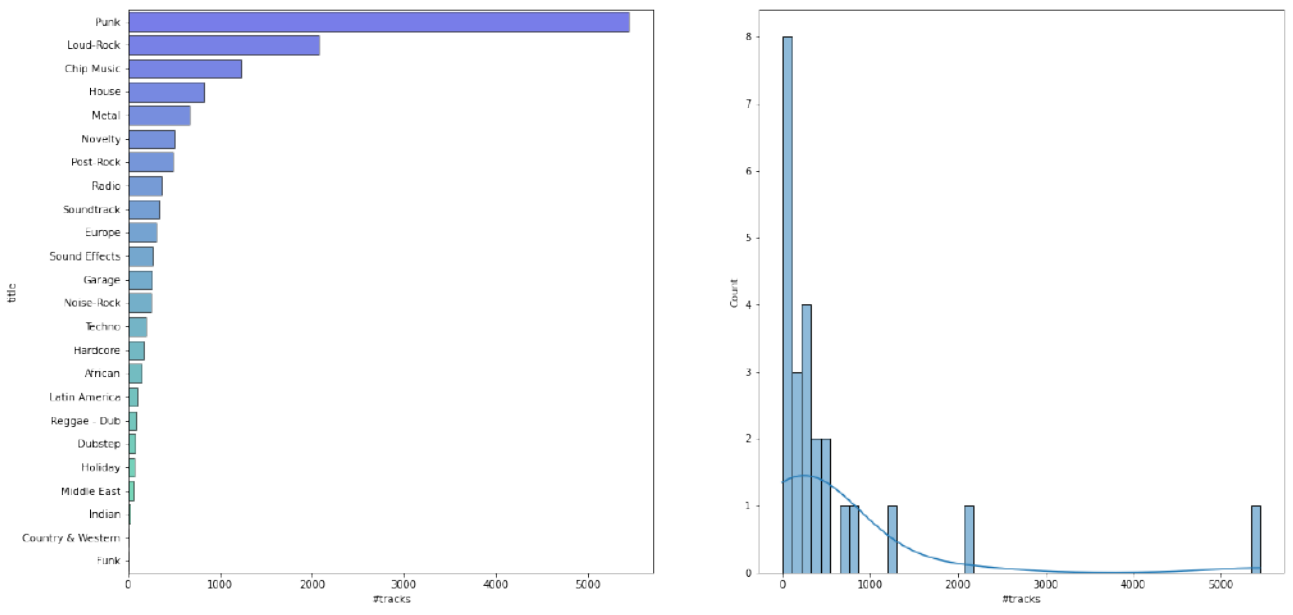
Figure A.4.: Tempogram and fourier tempogram examples

B. Datasets

B.1. Genre distributions for the FMA dataset



(a) genre distribution and tracks per label distribution in FMA



(b) subgenre distribution and tracks per label distribution in FMA

B.2. Final selected tags and merging dictionaries for all tag types

B.2.1. Subgenres

```
{"FMA":
    "wanted_subgenres": [
        "new age","country","techno","punk","loudrock","chipmusic","house",
        "metal","postrock","soundtrack","garage","noiserock","hardcore",
        "african","latinamerica","reggaedub","dubstep","middleeast","indian","avantgarde",
        "noise","ambient","experimentalpop","electroacoustic","lofi","ambient electronic",
        "indierock","improv","singersongwriter","idm","glitch","drone",
        "psychrock","psychfolk","industrial","downtempo","postpunk","synthpop","triphop","freejazz",
        "dance","contemporary classical","chiptune","hiphopbeats","minimalelectronic","americana",
        "chillout","progressive","shoegaze","freefolk","alternativehiphop",
    ],
    "subgenre_merging": {
        "african": "ethno","improv": "freejazz","middleeast": "eastern","psychrock": "psychedelic",
        "psychfolk": "psychedelic","freefolk": "psychedelic","americana": "country",
        "noiserock": "noise","chipmusic": "chiptune","minimalelectronic": "minimal",
        "indierock": "indie","drone": "minimal","loudrock": "hardrock","new age": "ambient",
    }
"MTAT" : {
    "wanted_subgenres": [
        "ambient","new age","dance", "country",
        "techno","beat","indian","newage",
        "beats","eastern","orchestra","hardrock",
        "trance","spanish","electronica","oriental","funky"
        ,"tribal","blues","metal",
        ],
    "subgenre_merging": {
        "orchestra": "orchestral","beat": "hiphopbeats","beats": "hiphopbeats",
        "oriental": "eastern","spanish": "latinamerica","funky": "funk",
        "tribal": "ethno","electronica": "dubstep","newage": "ambient",
        }
    }
"MTG" : {
    "wanted_subgenres": [
        "darkambient","ambient","newage","soulrnb",
        "alternative","dance", "indie","lounge",
        "newage","techno","house","trance","downtempo",
        "atmospheric","triphop","funk","reggae","blues",
        "electropop","singersongwriter","punkrock","latin","industrial",
        "synthpop","minimal","psychedelic","country","rnb",
        "dubstep","soul","hardrock","disco","dub",
        "ethno","deephouse","dancehall","postrock",
        "orchestral","metal",
    ],
    "subgenre_merging": {
        "latin": "latinamerica","punkrock": "punk","dub": "reggaedub",
        "reggae": "reggaedub","breakbeat": "garage","drumnbass": "garage",
        "rnb": "soulrnb","soul": "soulrnb","newage": "ambient",
        "darkambient": "ambient","electropop": "synthpop",
    },

    }
}
```

B.2.2. Moods/Themes/Instruments

```
"MTAT": {"wanted_moodthemes": [
        "guitar","slow","strings","drums",
        "fast","piano","violin","vocal","synth",
        "female","male","singing","vocals","no vocals",
        "heavy", "loud", "quiet", "flute", "woman",
        "male vocal", "no vocal", "sitar", "man", "voice", "female vocal",
        "male voice","cello","no voice","female voice","drum",
        "male vocals","violins","female vocals","bass","string","electric",
        "classical guitar","upbeat","acoustic","male singer","electric guitar",
        "man singing","no singing","percussion","woman singing",
    ],
    "moodtheme_merging": {
        "vocals": "vocal","singing": "vocal","woman": "female",
        "male vocal": "male","no vocals": "no vocal","man": "male",
        "voice": "vocal","female vocal": "female","male voice": "male",
        "no voice": "no vocal","female voice": "female","drum": "drums",
        "male vocals": "male","violins": "violin","female vocals": "female",
        "string": "strings","electric": "electricguitar","electric guitar": "electricguitar",
        "classical guitar": "guitar","male singer": "male","man singing": "male",
        "no singing": "no vocal","percussion": "drums","woman singing": "female",
        "fast": "energetic","upbeat": "energetic","heavy": "dark",
    },
}
"MTG" : {
    "wanted_moodthemes": [
        "piano","synthesizer", "drums", "bass",
        "guitar","electricguitar","computer","keyboard","acousticguitar",
        "violin","drummachine","happy","voice",
        "energetic","relaxing","electricpiano","strings","emotional",
        "melodic","dark","cello","epic","dream",
        "love","percussion","saxophone","inspiring","trumpet","flute",
        "sad","meditative","classicalguitar","uplifting",
        "motivational","deep","beat","romantic","positive",
        "playful","brass" "fun","soft","calm","slow",
    ],
    "moodtheme_merging": {
        "acousticguitar": "guitar","synthesizer": "synth","voice": "vocal",
        "percussion": "drums","classicalguitar": "guitar","rhodes": "keyboard",
        "electricpiano": "keyboard","bongo": "drums","trumpet": "brass",
        "saxophone": "brass","romantic": "love","soft": "quiet",
        "calm": "slow","motivational": "inspiring","dream": "meditative",
    },

}
```

B.2.3. Alltags

```
{"FMA" :
"wanted_alltags": [
        "blues","jazz","classical","country","pop","rock",
        "soulrnb","electronic","folk","spoken","hiphop",
        "new age","country","techno","blues","punk",
        "house","metal","postrock","soundtrack","garage","latinamerica",
        "reggaedub","dubstep","ambient","lofi","african",
        "indierock","new age","singersongwriter","psychrock","psychfolk",
        "industrial","downtempo","synthpop","triphop","dance",
        "hiphopbeats","chillout","freefolk","freejazz",
    ],
    "alltags_merging": {
        "classic": "classical","dance": "electronic","electro": "electronic",
        "new age": "ambient","country": "folk","freejazz": "jazz",
        "opera": "classical","techno": "electronic","spoken": "hiphop",
        "african": "ethno","psychrock": "psychedelic","psychfolk": "psychedelic",
        "freefolk": "psychedelic","americana": "country","indierock": "indie",
        "new age": "ambient",
    }
"MTAT" : {
    "wanted_alltags": [
        "techno","classical","electronic","rock","opera",
        "pop","classic","dance","new age","ambient",
        "country","metal","electro","jazz","jazzy",
        "instrumental","orchestra",
        "folk","ambient","new age","dance","country",
        "techno","beat","newage","beats","orchestra",
        "trance","spanish","electronica","oriental","funky",
        "tribal","blues","metal","guitar","slow",
        "strings","drums","fast","piano","violin",
        "vocal","synth","female","male","singing",
        "vocals","no vocals","heavy","loud","quiet",
        "flute","woman","male vocal","no vocal","sitar",
        "man","voice","female vocal","male voice","cello",
        "no voice","female voice","drum","male vocals","violins",
        "female vocals","bass","string","electric","classical guitar",
        "upbeat","acoustic","male singer","electric guitar","man singing",
        "no singing","percussion","woman singing",
    ],
    "alltags_merging": {
        "classic": "classical","dance": "electronic","electro": "electronic","new age": "ambient",
        "country": "folk","opera": "classical","techno": "electronic","jazzy": "jazz",
        "orchestra": "orchestral","orchestra": "orchestral","beat": "hiphopbeats","beats": "hiphopbeats",
        "spanish": "latinamerica","funky": "funk","tribal": "ethno",
        "electronica": "dubstep","newage": "ambient","vocals": "vocal","singing": "vocal",
        "woman": "vocal","male vocal": "vocal","no vocals": "no vocal",
        "man": "vocal","voice": "vocal","female vocal": "vocal",
        "male voice": "vocal","no voice": "no vocal","female voice": "vocal","drum": "drums","male vocals": "vo
        "female vocals": "vocal","string": "strings","electric": "electricguitar",
        "electric guitar": "electricguitar","classical guitar": "guitar",
        "male singer": "vocal","man singing": "vocal","no singing": "no vocal","percussion": "drums",
        "woman singing": "vocal","fast": "energetic","upbeat": "energetic",
        "heavy": "dark","male": "vocal","female": "vocal",
    }
}
}
```

B.3. Final training and validation distributions after refactoring

B.4. subgenres

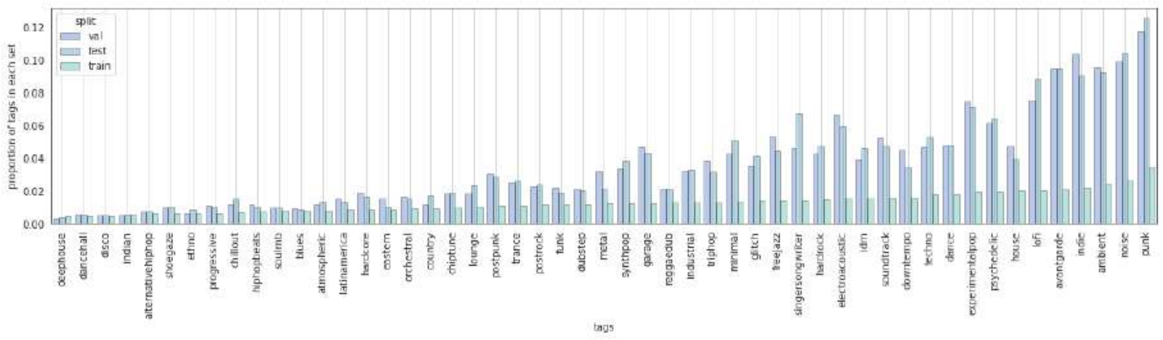


Figure B.2.: Resampled subgenres split into train test and validation split.

B.5. mood/themes/instruments

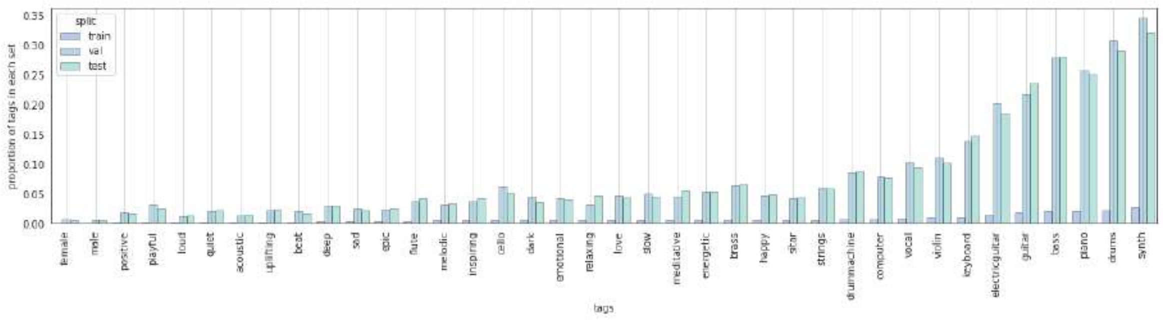


Figure B.3.: Resampled subgenres split into train test and validation split.

B.6. alltags

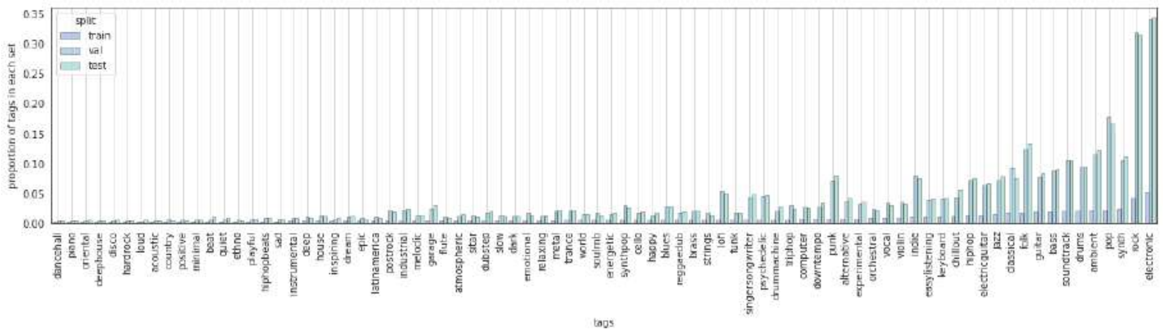


Figure B.4.: Resampled all tags split into train test and validation split.