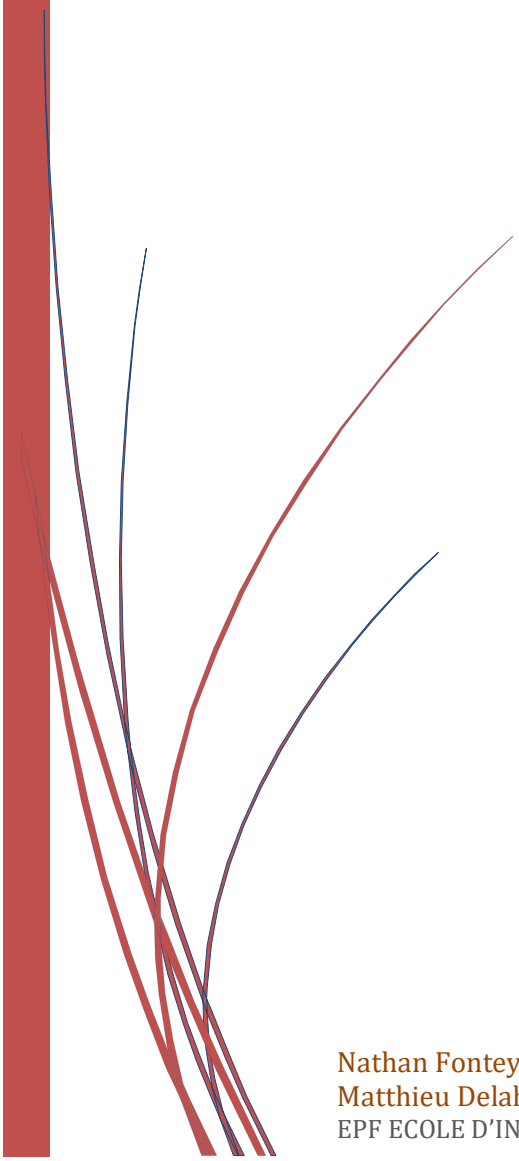
2022-2023

Etat de l'art du projet Muzik Patch Generator



Nathan Fonteyne, Yann Faussurier, Anhkim Chaloupe, Julia Forte,
Matthieu Delahaye, Benjamin Durand, Lucas Juste
EPF ECOLE D'INGENIEUR

Abstract

Ce projet a pour but d'aider l'entreprise Kiviak Instrument dans la conception de fonctionnalités utiles pour le WOFI.

La rédaction de l'Etat de l'Art complet, fournit à l'entreprise est une documentation facilitant la reprise et continuation du projet par d'autres personnes.

Afin de mener à bien ce projet, nous avons eu recours a beaucoup de documentation nous permettant de compléter au mieux notre savoir sur la musique (d'un point de vie scientifique), ainsi que son utilisation dans le monde du Machine Learning et Deep Learning.

But : Créer un algorithme de classification binaire des sons (écoutable / non-écoutable)

Sommaire

Introduction

Définir le champ d'application de l'examen

La **classification musicale** à l'aide de l'apprentissage automatique fait référence à l'utilisation de techniques d'apprentissage automatique pour classer la musique dans différentes catégories ou genres.

Cela peut se faire en formant un modèle d'apprentissage automatique sur un grand ensemble de données d'exemples musicaux étiquetés, où chaque exemple appartient à un genre spécifique ou reçoit une notation spécifique.

Le **modèle** peut ensuite être utilisé pour **prédire** le genre d'une nouvelle musique non étiquetée sur la base des caractéristiques apprises à partir des données d'apprentissage. Le champ d'application d'une revue dans ce domaine pourrait inclure une approche spécifique de l'apprentissage automatique ou une comparaison de différentes approches pour la classification de la musique, un examen de l'efficacité d'une méthode particulière sur un ensemble de données particulier, ou une étude de l'état de l'art de la classification de la musique à l'aide de l'apprentissage automatique.

L'importance de la classification musicale et de ses applications

La classification musicale est le **processus d'organisation** de la musique en **catégories** basées sur certains critères. C'est une tâche importante dans le domaine de la recherche d'informations musicales et elle a de nombreuses **applications pratiques**.

Voici quelques-unes des applications importantes de la classification musicale :

- La recommandation musicale

La classification musicale peut être utilisée pour construire des systèmes de recommandation musicale personnalisés qui suggèrent de la musique aux utilisateurs en fonction de leurs préférences.

- Découverte de musique

La classification musicale peut aider les utilisateurs à découvrir de nouvelles musiques qui sont similaires à celles qu'ils aiment déjà.

- Organisation de la musique

La classification musicale peut être utilisée pour organiser les collections de musique par genre, artiste, album, etc., ce qui permet aux utilisateurs de trouver plus facilement la musique qu'ils recherchent.

- Analyse de la musique

La classification musicale peut être utilisée pour analyser les données musicales et extraire des informations sur les tendances musicales, la popularité et d'autres caractéristiques.

- Éducation musicale

La classification musicale peut être utilisée pour enseigner aux élèves les différents styles et traditions musicaux, et les aider à comprendre comment la musique évolue dans le temps.

Fournir un aperçu des différentes approches de la classification musicale

Il existe plusieurs approches de la classification musicale, qui peuvent être regroupées en deux catégories : les **approches symboliques et les approches audio**.

- Les approches symboliques de la classification musicale

Elles utilisent des représentations **symboliques** de la musique, telles que les partitions ou la notation musicale, pour classer la musique.
Ces approches consistent généralement à extraire des caractéristiques de la représentation symbolique, comme des **motifs mélodiques** ou des progressions d'accords, et à utiliser des algorithmes d'apprentissage automatique pour classer la musique en fonction de ces caractéristiques.

- Les approches de la classification musicale basées sur l'audio

Elles utilisent le **signal audio** de la musique pour la classer.
Ces approches consistent généralement à extraire des caractéristiques du signal audio, telles que des **caractéristiques spectrales ou temporelles**, et à utiliser des algorithmes d'apprentissage automatique pour classer la musique en fonction de ces caractéristiques.

- Les approches hybrides de la classification musicale

Elles utilisent à la fois des **méthodes symboliques et des méthodes audios**. Ces approches peuvent combiner les informations des deux représentations pour améliorer les performances de classification.

Représentation des données

Les différentes façons dont les données musicales peuvent être représentée

- Waveform

Le terme “audio brut” (waveform) est souvent utilisé pour faire référence à une forme d'onde codée en utilisant la modulation par impulsions codées (PCM).

Cela consiste à **échantillonner une forme d'onde** continue dans le temps et l'amplitude. Il représente la forme d'onde sous forme de **séquence de nombres**, chaque nombre représentant un échantillon d'amplitude à une fréquence d'échantillonnage choisie.

Pour que cette séquence discrète d'échantillons capture toutes les informations nécessaires, la fréquence la plus élevée du signal doit respecter le théorème de Nyquist-Shannon selon lequel seules les fréquences inférieures à la moitié de la fréquence d'échantillonnage peuvent être reproduites à partir du signal échantillonné.

Une fréquence d'échantillonnage typique pour les applications audio est de 44,1 kHz. Chaque nombre réel est attribué à la valeur approximative fixe dans un ensemble discret fini de nombres. Un son d'une durée d’une seconde échantillonnée à 44,1kHz génère 44100 échantillons. Cette représentation est considérée comme extrêmement informative même pour les réseaux d'apprentissage profond.

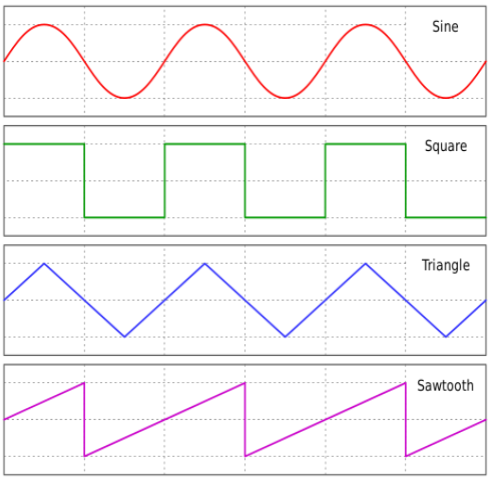


Figure 1: Waveform - Wikipedia

- Spectrograms

Un spectrogramme est une **représentation visuelle du temps/fréquence du son**. Un spectrogramme peut être obtenu via la Transformée de Fourier à Court Terme (STFT), où la Transformée de Fourier est appliquée à des segments chevauchants de la forme d'onde. Le spectrogramme utilise uniquement les valeurs absolues de la STFT, en écartant la phase. Ce type de spectrogrammes a été utilisé dans de nombreux articles.

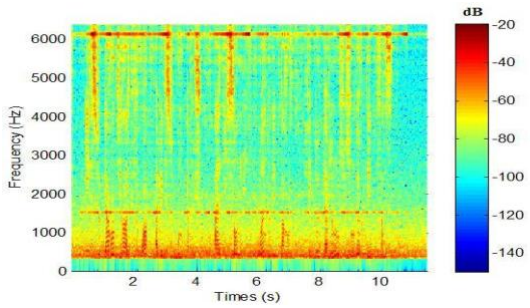


Figure 2: Figure 2: https://www.researchgate.net/profile/Faiz-Adil

- Mel spectrogramme

Notre projet se base sur la subjectivité de l'oreille humaine pour définir l'écoutabilité ou non d'un son. Nous avons donc choisi d'utiliser une échelle permettant de refléter comment l'oreille humaine perçoit le son.

En l'occurrence nous utilisons l'échelle de Mel, une échelle de fréquences basée sur la perception humaine. Elle se mesure en mels.

Elle a été conçue de telle façon que 1000 Hz correspondent à 1000 mels et qu'une variation constante dans le domaine des mels soit perçue comme une variation constante de fréquence (en Hz) par les auditeurs. En effet, au-delà de 500 Hz, l'oreille humaine ne perçoit plus le changement d'octave comme un doublement de la fréquence. Cependant, si on se place sur une échelle de Mel, un changement d'octave sera effectivement perçu comme un doublement de la fréquence.

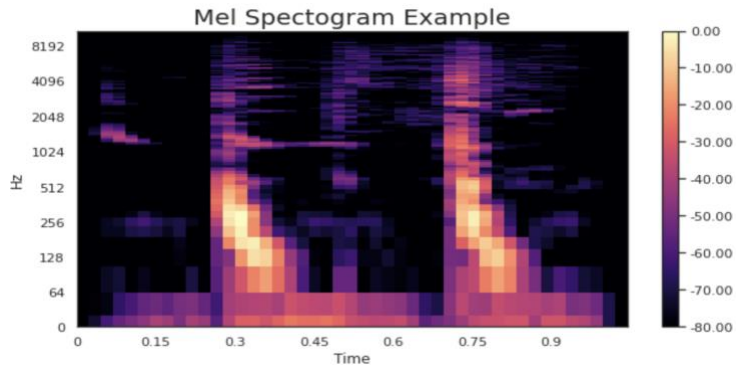


Figure 3: https://fr.wikipedia.org/wiki/Échelle_des_mels

- Acoustic Features

Pour surmonter la richesse d'informations acoustiques présentes dans une forme d'onde sonore, diverses études extraient des **caractéristiques perceptuelles** du signal d'origine. Ces caractéristiques acoustiques peuvent être représentées par des entrées de phonème, une fréquence fondamentale et des caractéristiques spectrales ou de multiples informations telles que la vélocité, l'instrument, la hauteur et le temps.
D'autres implémentations ont inclus des coefficients spectraux ou une variété de caractéristiques linguistiques et acoustiques.

- Embeddings

Les embeddings ont été initialement introduits en traitement du langage naturel (NLP) afin de convertir un mot ou une phrase en un vecteur de valeurs réelles. Cette approche a aidé le processus de texte dans les applications d'apprentissage profond en insérant la propriété d'embeddings plus proches dans l'espace vectoriel pour encoder des mots de sens similaire.
La même approche a été adoptée par le traitement du son pour réduire la dimensionnalité du signal, améliorer la synthèse de timbre ou même générer une **représentation plus interprétable** pour extraire efficacement des paramètres pour un synthétiseur.

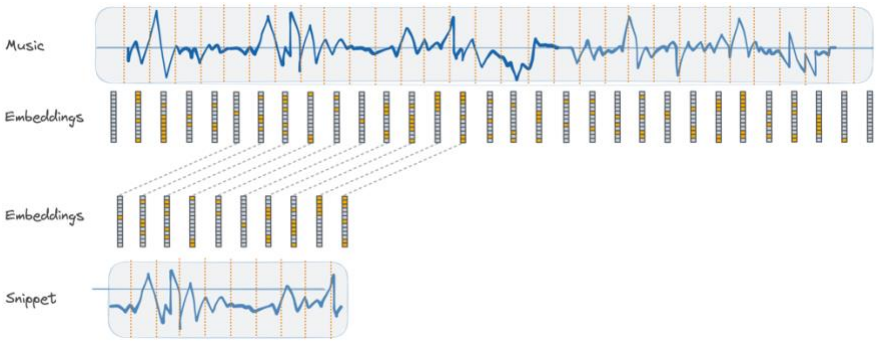


Figure 4: Music classification – embeddings

- Symbolique

Dans le traitement de la musique, le terme symbolique fait référence à l'utilisation de représentations telles que l'interface de données musicales (MIDI) ou les feuilles de pianos. MIDI est un standard technique qui décrit un protocole, une interface numérique ou le lien pour l'opération simultanée entre de multiples instruments de musique électroniques. Un fichier MIDI montre les notes jouées à chaque instant. Habituellement, ce fichier consiste en des informations sur l'instrument joué, la hauteur et sa vélocité. MidiNet est l'une des implémentations les plus populaires utilisant MIDI pour générer des morceaux de musique.

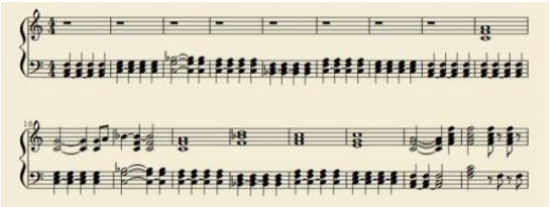


Figure 5: Music classification - symbolique

Les avantages et inconvénients de chaque représentation

	Avantages	
	En général	Machine Learning / Deep Learning
Waveform	Représentation visuelle claire du signal qui évolue dans le temps.	Contient de nombreuses informations sur le signal : la fréquence, la phase et l'amplitude.
	Identifie des caractéristiques spécifiques du signal : les changements brusques d'amplitude ou les périodicités.	Identifie des modèles et des tendances pas nécessairement visibles dans d'autres types de données.
	Détecte et élimine le bruit des signaux audio.	Largement disponibles et facilement collectées à l'aide de capteurs et d'autres dispositifs de mesure.
Spectorgams	Compréhension plus intuitive du contenu en fréquence d'un signal (signaux audios)	
	Révèle des modèles et tendances pas toujours visibles dans d'autres types de données.	
	Facilement générés à l'aide d'algorithmes de transformée de Fourier rapide (FFT).	
Acoustic features	Capturent des informations importantes sur le timbre, la hauteur et l'intensité d'un signal audio.	
	Caractéristiques acoustiques relativement faciles à extraire à l'aide d'une variété d'algorithmes et d'outils.	
	Identifie des modèles et des tendances dans les données audios qui peuvent ne pas être visibles dans la forme d'onde brute.	
Emebddings	Capturent des informations importantes sur la structure et le contenu des signaux audio, et peuvent être utilisés pour représenter les données audios d'une manière compacte et efficace.	
	Générées à l'aide de diverses techniques : la réduction de la dimensionnalité et les approches basées sur les réseaux neuronaux.	
	Utilisés en entrée d'un large éventail de modèles d'apprentissage automatique, notamment les algorithmes de regroupement, les modèles de classification et les systèmes de recommandation.	
Symbolic	Représentation précise et détaillée d'une composition musicale : la hauteur, la durée, la dynamique etc.	Compacte et facilement manipulable. Simplicité à extraire des caractéristiques et effectuer un prétraitement.
	Facilitée à éditer et manipuler à l'aide de logiciels informatiques ou d'autres outils.	Lisible par l'homme : facilite la compréhension et le débogage des modèles.
	Utilisée et comprise dans l'industrie musicale, ce qui facilite le partage et la communication des idées musicales.	Adaptée aux tâches qui nécessitent la compréhension de la structure et de la signification de la musique, comme la génération et la transcription de musique.

	Inconvénients	
	En général	Machine Learning / Deep Learning
Waveform	Ne fournissent pas d'informations sur le contenu en fréquence du signal.	*Mêmes inconvénients que pour : Acoustic features, embeddings et spectrogrammes*
	Difficiles à analyser pour les signaux fortement non périodiques ou présentant des motifs complexes.	
	Sensibles à l'échelle et à la résolution des axes, ce qui rend difficile la comparaison de formes d'onde provenant de sources différentes.	
Acoustic features + Embeddings + Spectrogrammes	Difficulté à capturer toutes les informations pertinentes d'un signal audio et à l'interpréter, en particulier si le signal est complexe ou comporte plusieurs composantes.	
	Nécessite des calculs intensifs, en particulier pour les grands ensembles de données.	
	Peu adapté au traitement des données d'intégration, en particulier si les données sont très variables ou présentent des dépendances complexes.	
Symbolic	Difficulté à prendre en compte toutes les nuances et subtilités d'une performance musicale : le timbre, l'articulation et le phrasé.	Dépourvue de la richesse des informations temporelles présentes dans les formes d'onde audio, qui peuvent être importantes pour des tâches telles que la classification musicale et le transfert de style musical.
	Difficulté avec les musiques non-écrites en notation standard, comme la musique improvisée ou la musique de cultures non occidentales.	Nécessite une annotation manuelle ou l'utilisation d'algorithmes de transcription musicale automatique, ce qui peut prendre beaucoup de temps et être source d'erreurs.
	Certains algorithmes d'apprentissage automatique peuvent ne pas être bien adaptés au traitement de données musicales symboliques, en particulier si les données sont très variables ou présentent des dépendances complexes.	Limite l'éventail des styles musicaux et des instruments dont un modèle peut tirer des enseignements, car la représentation peut ne pas saisir avec précision certaines caractéristiques musicales présentes dans les formes d'onde audio mais pas dans la notation.

Datasets existants utilisés pour la classification musicale et leurs caractéristiques

Il existe plusieurs dataset existants qui sont couramment utilisés pour les tâches de classification musicale. En voici quelques exemples :

- GTZAN dataset

Composé de 1000 audio tracks, chacun d'une durée de 30 secondes, organisé en 10 genres (blues, classical, country, disco, hiphop, jazz, reggae, rock, metal, and pop).

- Million Song Dataset

Collection d'un million de chansons fournie par The Echo Nest. Il comprend des métadonnées pour chaque chanson, comme le nom de l'artiste, l'année de sortie et les étiquettes de genre. Les données audios ne sont pas incluses, mais le jeu de données fournit un identifiant unique pour chaque chanson qui peut être utilisé pour récupérer les données audios à partir d'autres sources.

- Free Music Archive dataset

Se compose de plus de 50 000 pistes audios et des métadonnées qui les accompagnent, y compris les étiquettes de genre, fournies par la Free Music Archive. Les données audios sont fournies au format MP3.

- ISMIR 2004 Genre Classification dataset

Se compose de 1000 pistes audios organisées en 10 genres (blues, classique, country, disco, hiphop, jazz, reggae, rock, métal et pop). Les pistes durent entre 30 secondes et 3 minutes et sont fournies au format MP3.

- FMA Small dataset

Sous-ensemble de l'ensemble de données Free Music Archive, composé de 8 000 pistes et des métadonnées correspondantes. Il s'agit d'un sous-ensemble équilibré, avec un nombre égal de pistes de chaque genre. Les données audios sont fournies au format MP3.

Chacun de ces datasets a ses propres caractéristiques et peut être mieux adapté à certains types de tâches de classification musicale. Il est important de tenir compte de la taille, de l'équilibre des genres et du format audio du jeu de données lors de sa sélection pour une tâche particulière.

Extraction de caractéristiques (features) :

Les différents types de caractéristiques qui peuvent être extraites des données musicales

Pour entraîner un modèle ML, nous devons d'abord **extraire les caractéristiques utiles d'un signal audio.**
L'extraction de caractéristiques audio est une étape nécessaire du traitement du signal audio, qui est un sous-domaine du traitement du signal. Il s'agit du traitement ou de la manipulation des signaux audio.
Il supprime les bruits indésirables et équilibre les plages de temps et de fréquence en convertissant les signaux numériques et analogiques.

Un signal audio est une représentation du son. Il code toutes les informations nécessaires à la reproduction du son. Les signaux audios se présentent sous deux types de base : **analogique et numérique**.

- L'analogique désigne l'audio enregistrer à l'aide de méthodes qui **reproduisent les ondes sonores originales**, comme les disques vinyles et les cassettes.

- L'audio numérique est enregistré en prenant **des échantillons de l'onde sonore** d'origine à une fréquence spécifiée, appelée fréquence d'échantillonnage, comme les CD et les MP3.

Caractéristiques audio communes utiles pour la modélisation

Les **caractéristiques audios** sont des **descriptions du son** ou d'un signal audio qui peuvent être introduites dans des modèles ML pour construire des systèmes audios intelligents.

Les applications audios qui utilisent ces caractéristiques comprennent la classification audio, la reconnaissance vocale, l'étiquetage automatique de la musique, la segmentation audio et la séparation des sources, l'empreinte audio, le débruitage audio, la recherche d'informations musicales, etc.
En général, les caractéristiques audios sont classées en fonction des aspects suivants :

- Level of Abstraction :

Caractéristiques de haut niveau, de niveau moyen et de bas niveau des signaux musicaux.

- Temporal Scope :

Les caractéristiques du domaine temporel peuvent être: instantanées, segment-level, et globales.

- Musical Aspect :

Propriétés acoustiques qui comprennent le battement, le rythme, le timbre (couleur du son), la hauteur, l'harmonie, la mélodie, etc.

- Signal Domain :

Caractéristiques dans le domaine temporel, le domaine fréquentiel ou les deux.

- ML Approach :

Caractéristiques triées sur le volet pour la modélisation ML traditionnelle ou extraction automatique de caractéristiques pour la modélisation par apprentissage profond.

Comment catégoriser les caractéristiques audios à différents niveaux d'abstraction

Ces grandes catégories couvrent principalement les signaux musicaux plutôt que l'audio en général :

- High-level

Ce sont les caractéristiques abstraites qui sont comprises et appréciées par les humains. Elles comprennent l'instrumentation, la tonalité, les accords, la mélodie, l'harmonie, le rythme, le genre, l'humeur, etc.

- Mid-level

Ce sont des caractéristiques que nous pouvons percevoir. Il s'agit notamment de la hauteur, des descripteurs liés au rythme, des débuts de note, des modèles de fluctuation etc.

- Low-level

Ce sont des caractéristiques statistiques qui sont extraites de l'audio. Elles ont un sens pour la machine, mais pas pour les humains. Par exemple: l'enveloppe de l'amplitude, l'énergie, spectral centroid, spectral flux, zero-crossing rate, etc.

Portée temporelle des caractéristiques audio (Temporal scope)

Ce type de catégorisation s'applique à l'audio en général, c'est-à-dire à l'audio musical et non musical :

- Instantané

Ces fonctions nous communiquent des informations instantanées sur le signal audio. Elles considèrent de minuscules morceaux du signal audio, de l'ordre de quelques millisecondes.

- Segment-level

Ces caractéristiques peuvent être calculées à partir de segments du signal audio de l'ordre de quelques secondes.

- Global

Il s'agit de caractéristiques générales qui fournissent des informations et décrivent l'ensemble du son.

Caractéristiques du domaine du signal pour l'audio

Les caractéristiques du domaine du signal comprennent les caractéristiques les plus importantes ou plutôt descriptives de l'audio en général :

- Time domain

Elles sont extraites des formes d'onde de l'audio brut. Exemples: Zero crossing rate, amplitude envelope, and Root Mean Square (RMS).

- Frequency domain

Ils se concentrent sur les composantes de fréquence du signal audio. Les signaux sont généralement convertis du domaine temporel au domaine fréquentiel à l'aide de la transformée de Fourier. Exemples : Band energy ratio, spectral centroid, and spectral flux.

- Time-frequency representation

Ces caractéristiques combinent à la fois les composantes temporelles et fréquentielles du signal audio. La représentation temps-fréquence est obtenue en appliquant la transformée de Fourier à court terme (STFT) sur la forme d'onde du domaine temporel. Exemples : Spectrogramme, mel-spectrogramme et transformée en Q constant.

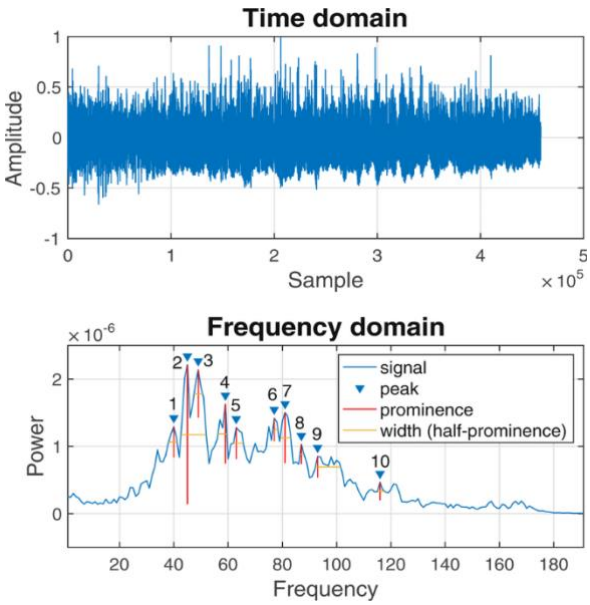


Figure 7: Caractéristiques - Time Domain and Frequency Domain

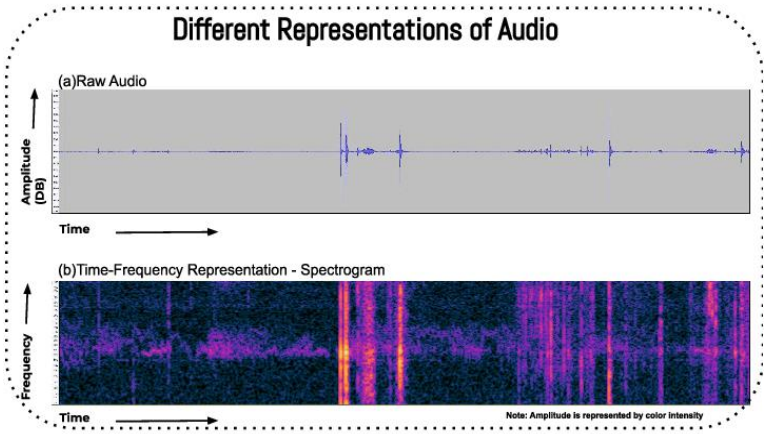


Figure 6: Caractéristiques - Time/Frequency Représentation

Caractéristiques audios dans l'approche ML

L'apprentissage de modèles de ML considère toutes ou la plupart des **caractéristiques des domaines temporel et fréquentiel** comme des entrées dans le modèle. Les caractéristiques doivent être sélectionnées en fonction de leur effet sur les **performances du modèle**.

Parmi les caractéristiques les plus utilisées, nous avons : Amplitude Envelope, Zero-Crossing Rate (ZCR), Root Mean Square (RMS) Energy, Spectral Centroid, Band Energy Ratio, and Spectral Bandwidth.

Le Deep Learning considère les représentations audios non structurées telles que le **spectrogramme** ou les MFCC. Elle extrait les motifs par elle-même. Cette approche est la plus utilisée puisque l'extraction des caractéristiques est automatique. Elle est également soutenue par l'abondance de données et la puissance de calcul. Les caractéristiques ou représentations couramment utilisées qui sont directement introduites dans les architectures de réseaux neuronaux sont les spectrogrammes, les **mél-spectrogrammes** et les coefficients cepstraux de fréquence Mel.

ZCR : Zero Crossing Rate

Le ZCR est un indicateur proportionnel au taux de **changement de signe d'un signal musical**. Il est très utilisé pour la reconnaissance vocale et semble être un critère intéressant pour faire de la classification de musiques.

Spectral centroïde

Le spectral centroïde est une métrique utilisée pour **caractériser les spectres des musiques**. Il s'agit d'un indicateur géométrique de la position du centre de masse du spectre d'un son.

MFCC : Mel-Frequency Cepstral Coefficients

Les MFCC sont des coefficients cepstraux qui correspondent à une **transformation sinusoïdale de la puissance d'un signal**.

Spectral Roll-Off Point

Cette feature est une mesure de **l'asymétrie à droite d'un spectre sonore**. Elle est utilisée par exemple pour différencier des paroles chantées et des paroles normales.

Les techniques et algorithmes utilisés pour extraire ces caractéristiques

Les techniques et algorithmes plus couramment utilisés pour extraire des caractéristiques à partir de signaux audios sont les suivantes :

- La transformée de Fourier à court terme (STFT)

Permet de décomposer un signal audio en un **spectre de fréquences à un instant donné**. En utilisant des fenêtres glissantes sur le signal, on peut obtenir une série de spectres de fréquences qui représentent comment les **fréquences évoluent dans le temps**. Cette technique est souvent utilisée pour l'analyse de la musique et de la parole.

- La transformée en Q constant (CQT)

Cette technique est similaire à la STFT, mais utilise des **fréquences logarithmiques** plutôt que linéaires. Cela permet de mieux séparer les fréquences basses des fréquences élevées, ce qui est souvent utile pour l'analyse de la musique.

- Le mél-spectrogramme

Cette technique utilise une transformée de Fourier à court terme pour décomposer un signal audio en un **spectre de fréquences**, puis utilise une transformation logarithmique pour **représenter ces fréquences** en termes de mél, une unité de mesure de la hauteur de la fréquence qui est plus proche de la façon dont nous percevons les sons.

- L'analyse cepstrale

Utilise une transformée de Fourier inversée pour obtenir un signal qui est basé sur les **décalages de phase** plutôt que sur l'amplitude des fréquences. Cela permet de mettre en évidence certaines caractéristiques de la parole, telles que le débit et les tonalités.

- L'analyse des ondelettes

Utilise une transformée d'ondelette pour décomposer un signal en une **série de décompositions à différentes échelles**. Cela permet de mettre en évidence des caractéristiques qui varient à différentes échelles de temps et de fréquence.

Chacune de ces techniques à ses propres avantages et inconvénients, et il peut être nécessaire de combiner plusieurs techniques pour extraire les caractéristiques souhaitées d'un signal audio.

Les méthodes d'extraction de caractéristiques existantes et leurs performances

- Méthodes basées sur les fréquences : Transformée de Fourier à court terme (STFT), Transformée en Q constant (CQT), Mél-spectrogramme
- Méthodes basées sur les ondelettes : Analyse des ondelettes
- Méthodes basées sur les réseaux de neurones :
    - Réseaux de neurones convolutionnels (CNN) : utilisés pour apprendre des caractéristiques automatiquement à partir des données d'entraînement. Ces réseaux ont été utilisés avec succès pour des tâches telles que la reconnaissance de la musique, la segmentation de la musique et la génération de la musique.
    - Auto-encoder est un type de réseau de neurones qui a pour but d'apprendre une représentation efficace des données en utilisant une couche cachée réduite. Ils sont souvent utilisés pour l'analyse de données non structurées telles que les audio.

Notation des sons

Afin de pouvoir exploiter les fichiers audios fournis par l'entreprise et pouvoir mener à bout la mission qui nous a été donnée (**classification binaire** des sons en écoutable / non-écoutable), nous avons décidé de constituer nous même une **base de données de samples labellisés.**

Pour cela, nous avons utilisé la base de de son fournie par notre client, cette base est constituée de **10 000 samples allant de 1 à 3 secondes**, trouvés sur internet. Les sources composant cette base de données sont très diverses, nous pouvons y trouver des musiques classiques, des musiques populaires, du hip hop ou encore des extraits de podcast.

Afin d'avoir une marge de manœuvre sur le système de notation, nous avons décidé de **noter chaque son avec une note comprise entre -2 et 2** (2 étant le meilleur score). à partir de chacun de nos scores nous pouvons alors calculer la **moyenne de nos scores** pour attribuer à chaque son une note finale.

Afin de pouvoir travailler tous ensemble sur ce système de notation, nous avons utilisé un **fichier Excel** partagé, chacun pouvait remplir les notes de chaque son à son rythme et les statistiques étaient calculées automatiquement par Excel :

sample 0002	Lucas	Nathan	A.K	Yann	Benjamin	Matthieu	Julia	Moyenne	Médiane	ecart type	Vote à la majorité	Critères	Remarques	Lucas
-008508-;	2	2	2	2	2	2	2	2,000	2	0	TRUE	-2	-008508-;	
-008520-;	2	2	2	1	2	1	2	1,714	2	0,487950036	TRUE	-1	-008520-;	
-008635-;	-1	-2	-1	-2	-2	0	0	-1,143	-1	0,899735411	FALSE	0	-008635-;	
-008728-;	2	1	2	2	2	2	2	1,857	2	0,377964473	TRUE	1	-008728-;	
-008750-;	-2	-2	-2	-2	1	-2	-2	-1,571	-2	1,133893419	FALSE	2	-008750-;	
-008769-;	-2	-2	-2	-2	-2	-1	-1	-1,714	-2	0,487950036	FALSE		-008769-;	
-008775-;	-2	-1	-1	-1	-1	0	0	-0,857	-1	0,690065559	FALSE		-008775-;	
-008795-;	0	2	1	1	1	0	1	0,857	1	0,690065559	TRUE		-008795-;	
-008829-;	1	2	1	1	1	0	0	0,857	1	0,690065559	TRUE		-008829-;	
-008889-;	1	1	1	1	-1	-1	-1	0,143	1	1,069044968	TRUE		-008889-;	
-008918-;	1	1	2	2	1	1	1	1,286	1	0,487950036	TRUE		-008918-;	
-008982-;	2	2	1	2	2	2	2	1,857	2	0,377964473	TRUE		-008982-;	
-009045-;	2	2	2	2	2	1	1	1,714	2	0,487950036	TRUE		-009045-;	
-009051-;	2	2	1	1	2	2	2	1,714	2	0,487950036	TRUE		-009051-;	
-009099-;	2	1	2	1	2	0	0	1,143	1	0,899735411	TRUE		-009099-;	
-009150-;	2	2	2	2	2	2	2	2,000	2	0	TRUE		-009150-;	
-009152-;	1	1	2	1	1	-1	2	1,000	1	1	TRUE		-009152-;	
-009214-;	2	1	1	2	2	2	2	1,714	2	0,487950036	TRUE		-009214-;	
-009234-;	0	2	1	2	2	0	2	1,286	2	0,951189731	TRUE		-009234-;	
-009344-;	-2	-2	-1	-2	-2	-1	0	-1,429	-2	0,786795792	FALSE		-009344-;	
-009345-;	0	2	0	1	1	1	1	0,857	1	0,690065559	TRUE		-009345-;	
-009350-;	1	2	1	1	1	1	1	1,143	1	0,377964473	TRUE		-009350-;	
-009361-;	2	1	1	2	2	2	1	1,571	2	0,534522484	TRUE		-009361-;	
-009387-;	2	2	2	2	2	2	2	2,000	2	0	TRUE		-009387-;	

Figure 8: Capture d'écran du fichier excel partagé du projet

Lors du projet, nous nous sommes rendu compte que la moyenne n'était pas nécessairement le critère le plus pertinent pour évaluer un son à partir de nos notes, nous avons donc rajouté **la médiane**, l'écart type et un système de vote à la majorité (qui s'est révélé obsolète par la suite par la médiane)

La médiane offre une note qui est **plus facilement interprétable par un modèle**, de plus, si une majorité de personne attribue une note positive, la médiane sera positive, ce qui permet d'établir un système de majorité intéressant dans le cadre d'une notation.

L'écart type permet de résoudre le problème que certains extraits audios ont qui est qu'ils sont notés avec une note trop dispersée. Ce sont des sons sur lesquels nous sommes en **désaccord** en ce qui concerne la qualité et qui se voient donc attribuer une note qui n'est pas représentative de l'avis des membres du groupe. Pour régler ce problème nous utilisons l'écart type.

Si l’écart type est trop grand, nous considérons que la note est une valeur aberrante et nous le **supprimons de la base d'entraînement** du modèle.
Cette base de données est ensuite exportée sur **python** où elle est traitée avec **pandas** pour que le modèle puisse travailler avec.

Méthodes utilisées

CNN : Convolutionnal Neural Network

Recherches amenant à la création du modèle

Pour créer ce modèle, nous sommes avant tout partis de l'existant en adaptant ce que l'on peut faire de performant actuellement à nos données et notre cas d'usage.

Un des modèles très intéressant que l'on a pu trouver est celui-ci :
https://www.kaggle.com/code/nilshmeier/melspectrogram-based-cnn-classification

Le notebook est décrit de cette manière dessus :

"*Ce notebook traite de la classification ou de la reconnaissance de chiffres parlés à l'aide de réseaux de neurones convolutifs (CNN en abrégé). Le jeu de données utilisé est librement disponible et peut être téléchargé sur la plateforme Kaggle. Il contient un total de 3 000 enregistrements audio de différents locuteurs qui ont prononcé chaque chiffre 50 fois. Le code traite les enregistrements audio, extrait les caractéristiques et entraîne un CNN.*"

Nous avons choisi d'utiliser ce modèle car il nous donnait en premier lieu un exemple de **preprocessing de fichiers audios** vers des Mel spectrogramme et ensuite car les différentes couches utilisées par la suite dans le modèle rejoignent celles utilisées dans d'autres modèles s'approchant plus de notre objectif.

Preprocessing

La première étape va être de **traiter nos données** afin de leur donner le bon format. Pour ce qui est des features, nous devons **transformer les fichiers .wav en Mel spectrogramme** afin de les donner en entrée à notre réseau séquentiel.

Pour faire cela, la première étape consiste à parcourir tous nos fichiers audios, les traiter avec Librosa, les **transformer en spectrogramme** et finalement rentrer toutes les données dans un **dictionnaire** en y associant le nom de fichier correspondant. En effet Librosa est une librairie fournissant les outils essentiels pour traitement des données audio. **Les targets** quant à elles, étant sur un excel, nous devons les importer à partir d'un autre fichier.
Ces éléments n'étant pas dans le même fichier à la base, nous devons faire en sorte d'associer la bonne feature à la bonne Target. Une fois cette étape faite, nous avons d'un côté **une matrice X contenant nos Mel spectrogrammes** et de l'autre un **vecteur y contenant les targets associées.**

Explications du modèle

Pour créer notre modèle nous avons utilisé la bibliothèque Keras de Tensorflow afin de créer un **modèle séquentiel**.

Ce modèle contient des **couches de neurones convolutionnels** afin de trouver des paternes dans les matrices de nos spectrogrammes. Ensuite, le **Max pooling** est utilisé pour réduire les dimensions spatiales du volume de sortie. Vous pouvez d'ailleurs remarquer que, plus le volume spatial de sortie diminue, plus le nombre de filtres appris augmente.
Nous avons ensuite un **dropout**, cela permet d'éteindre une partie des neurones selon une certaine probabilité. L'objectif est de rendre ces derniers plus **indépendants les uns des autres**. Cela permettra de réduire l'overfitting (la différence de précision en testant le modèle sur des données qu'il a déjà vues et sur des données qu'il n'a jamais vu.

Finalement, notre sortie étant **binaire**, nous mettons un seul neurone en sortie pour obtenir le résultat et nous utilisons une sigmoïde comme fonction d'activation.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 87, 16)       160

 max_pooling2d (MaxPooling2D  (None, 64, 43, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 43, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 32, 21, 32)       0
 2D)

 flatten (Flatten)           (None, 21504)             0

 dropout (Dropout)           (None, 21504)             0

 dense (Dense)               (None, 64)                1376320

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 1,381,185
Trainable params: 1,381,185
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 9: Model séquentiel

Analyse des résultats

En premier lieu nous pouvons observer un aperçu de l'une des courbes d'apprentissage. Dans l'exemple que l'on peut observer, nous ne voyons pas de réelle différence entre l'entraînement

et le testing. On pourrait dire à première vu que l'on n'a pas d'overfitting et que notre modèle est aussi performant sur des données connues que sur des données inconnues.
Pour vérifier cela nous allons regarder les statistiques sur **différentes seed**. Nous avons, en effet, à peu près 1000 données ce qui est très peu pour un réseau de neurones. Tester nos résultats sur différentes seed nous donnera ainsi une **estimation plus fiable** de la performance de notre réseau.

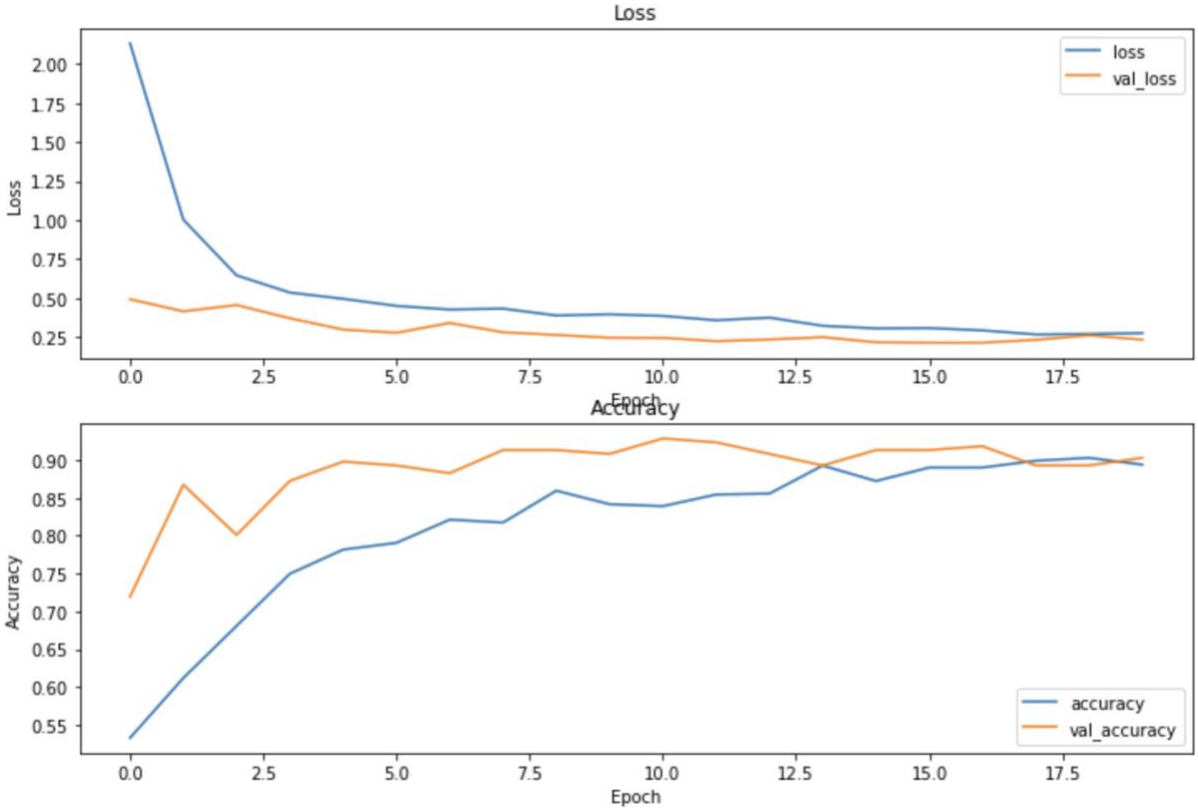


Figure 10: Model Séquentiel – Plots

seed 42 : || loss: 0.3335 - accuracy: 0.8633 - val_loss: 0.2328 - val_accuracy: 0.9235

seed 1 : || loss: 0.3199 - accuracy: 0.8685 - val_loss: 0.2376 - val_accuracy: 0.9184

seed 2 : || loss: 0.4435 - accuracy: 0.8186 - val_loss: 0.2773 - val_accuracy: 0.9286

seed 3 : || loss: 0.3938 - accuracy: 0.8378 - val_loss: 0.2463 - val_accuracy: 0.9286

seed 4 : || loss: 0.4000 - accuracy: 0.8314 - val_loss: 0.2518 - val_accuracy: 0.9133

seed 5 : || loss: 0.3528 - accuracy: 0.8582 - val_loss: 0.2152 - val_accuracy: 0.9286

seed 6 : || loss: 0.3746 - accuracy: 0.8391 - val_loss: 0.2147 - val_accuracy: 0.9337

Figure 11 : Model Séquentiel – Résultats d'entrainement avec différentes seeds

Seed : En Machine Learning, la *seed* désigne l'état d'initialisation d'un générateur de nombres pseudo-aléatoires. Si vous utilisez la même *seed*, vous obtiendrez exactement le même modèle de nombres.

En testant notre réseau sur différentes seed on aperçoit que les **résultats diffèrent selon les seed** et que nous avons visiblement une meilleure accuracy sur notre test que sur nos données d'entraînement.
Cela n'est pas une bonne chose, notre **modèle n'arrive visiblement pas à overfit**, pour essayer d'améliorer cela nous avons essayé de jouer avec le « *learning rate* », l' « *optimizer* » et le *nombre de batches* pour obtenir les meilleurs résultats possibles. Cependant on retombe sur des cas ou le score du test est meilleur que celui du train.

	loss	accuracy	val_loss	val_accuracy
Moyenne :	0,3533063	0,8562313	0,242425	0,9228625
Médiane	0,3629	0,84995	0,23715	0,92605

Figure 11: médiane et moyenne des différentes valeurs pour 15 seeds

Les résultats sont similaires lorsque l'on essaie sur une vingtaine de *seed*.
Au cours de nos recherche nous avons pu voir que, en général pour étudier des matrices de dimension similaire, les chercheurs utilisent environ 90 000 valeurs au total. Avec nos 1000 valeurs nous sommes 90 fois en dessous de ces recommandations.

Nous gardons tout de même en tête que les résultats sont intéressants, en effet notre modèle peut, d'après les statistiques, trouver **9 fois sur 10 sur un son est intéressant ou non**. De plus notre méthode de notation est subjective, et soumise au changement. (Celle-ci se faisant sur la durée, des sons similaires peuvent potentiellement être notés d'une manière différente par la même personne après un certain temps).

Après l'analyse de nos résultats, nous nous rendons compte que l'idéal serait de prendre en compte le fait que nous avons un **faible nombre de données** afin de créer un modèle plus adapté.

XGBoost

Raison du choix du modèle

L'idée ici était d'utiliser différentes features pour essayer d'avoir une précision la plus forte.
Pour cela, on s'est inspiré d'un article sur Machine Learning pour la classification automatique de musiques avec Python, et on a pris les algorithmes avec le plus de précision.
Si on retire le CNN, il reste le random Forest et XGBoost qui sont des algorithmes de **modélisation de prédictions**, et qui ont une bonne précision. Ils sont tous deux utilisés pour créer des modèles de prédiction à partir de données d'entraînement en utilisant un **processus d'apprentissage automatique supervisé**. Cependant, ils ont quelques différences clés.

Méthode d'apprentissage

Random Forest utilise une méthode d'apprentissage par bootstrap aggregating, également connue sous le nom de "bagging".
XGBoost, d'autre part, utilise une méthode d'apprentissage par "boosting".
Le boosting est une technique d'apprentissage automatique utilisée pour améliorer la précision des modèles de prédiction. Il fonctionne en **ajoutant des modèles de manière itérative**, en commençant par des modèles simples et en ajoutant des modèles de plus en plus complexes jusqu'à ce qu'une précision satisfaisante soit atteinte.
Chaque modèle ajouté est conçu pour **corriger les erreurs** commises par les modèles précédents, ce qui permet d'obtenir des prédictions de plus en plus précises au fil du temps.
Le bagging, également connu sous le nom de bootstrap aggregating, est une technique d'apprentissage automatique qui vise à **réduire l'instabilité des modèles** en utilisant plusieurs modèles indépendants et en prenant la **décision finale en votant**.
Cela se fait en générant de nouvelles données d'entraînement à partir de l'ensemble de données d'origine en utilisant un échantillonnage aléatoire avec remise (également connu sous le nom de "bootstrapping").
Chaque modèle est entraîné sur une version différente de ces données générées aléatoirement et les prédictions finales sont obtenues en votant.

Niveau de précision

XGBoost a tendance à être plus précis que Random Forest, car il utilise une méthode de "boosting" qui ajoute des modèles les uns après les autres jusqu'à ce qu'une précision satisfaisante soit atteinte. Random Forest, d'autre part, utilise un ensemble de modèles indépendants et choisit la prédiction finale en votant.

Vitesse d'exécution

XGBoost est généralement plus rapide que Random Forest car il utilise une méthode de "boosting" qui ajoute des modèles de manière itérative, ce qui permet d'éviter la nécessité de construire de nouveaux modèles à partir de zéro à chaque itération.

Utilisation de la mémoire

XGBoost peut utiliser plus de mémoire que Random Forest car il utilise une méthode de "boosting" qui ajoute des modèles de manière itérative, ce qui peut entraîner une utilisation accrue de la mémoire.

Pour toutes ces raisons, on a pris **la décision d'utiliser XGboost**.

Preprocessing

Le traitement des données dans ce cas est assez ressemblant à la méthode vue au-dessus. La différence ici est qu'une fois qu'on a traité nos fichiers avec Librosa on les utilisera pour **différentes caractéristiques des sons**. Vous pourrez retrouver les explications de chaque caractéristique utilisée dans cette partie.
Un fois qu'on a bien toute nos données on va pouvoir construire nos ensembles d'entraînement avec comme Target nos notes de son expliqué également dans la partie « Notation des sons ».

Analyse des résultats

Les résultats pour ce modèle sont intéressants, on a une précision d'environ 90%. Mais avec nos autres modèles, on trouve de meilleurs résultats, c'est pour cette raison qu'on ne retiendra pas ce modèle.

Transfer Learning

Recherches amenant à la création du modèle

Le Transfer Learning, ou apprentissage par transfert en français, désigne l'ensemble des méthodes qui permettent de **transférer les connaissances acquises à partir de la résolution de problèmes donnés pour traiter un autre problème**.
Le Transfer Learning est très utile dans notre cas car nous manquons cruellement de données (sons non labelisés). Il va permettre d'appuyer notre modèle grâce à un entraînement sur une autre base de données bien plus conséquente.

Le modèle que nous allons utiliser s'appelle YAMNet. C'est un **réseau neuronal** profond pré-entraîné qui peut prédire des événements audios parmi 521 classes, comme le rire, l'aboiement ou le cri d'un enfant.
Il peut utiliser une **forme d'onde audio comme entrée** et faire des prédictions indépendantes pour chacun des 521 événements audios du corpus AudioSet. (http://g.co/audioset)
En interne, le modèle extrait des "frames" du signal audio et traite les paquets (batch) de ces frames. Cette version du modèle utilise des frames de 0,96 seconde et extrait un frame toutes les 0,48 secondes.
Le modèle reçoit un tenseur 1-D float32 ou un tableau NumPy contenant une forme d'onde de longueur arbitraire, représentée par des échantillons 16 kHz à un seul canal (mono) dans la plage [-1,0, +1,0]. Nous allons ensuite convertir les fichiers WAV dans le format pris en charge.

Le modèle **retourne 3 sorties**, dont les scores des classes, les embeddings (que nous utiliserons pour le Transfer Learning) et le log mel spectrogramme.
Une utilisation spécifique de YAMNet est **l'extraction de caractéristiques** de haut niveau - la sortie d'intégration à 1024 dimensions. Nous allons utiliser les caractéristiques d'entrée du modèle de base (YAMNet) et les introduirons dans notre modèle moins profond composé d'une couche cachée "Dense". Ensuite, nous **entraînerons le réseau sur nos samples audios labelisés** (1000 au moment de la rédaction) pour la classification audio. Il n'y a donc pas besoin d'un grand nombre de données pour train.

Tout d'abord, nous construirons le pipeline de prétraitement des données. Par la suite, nous allons tester le modèle et visualiserons les résultats de la classification audio.

Preprocessing

YAMNet fournit des scores de classe pour chaque sample (c'est-à-dire 521 scores par sample).
Afin de déterminer les prédictions du sample audio, les scores peuvent être agrégés par classe à travers les images (par exemple, en utilisant l'agrégation moyenne ou maximale). On utilise la fonction np.mean(axis=0). Enfin, pour trouver la classe la mieux notée au niveau du sample audio, nous prenons le maximum des 521 scores agrégés.

Nous commençons par créer nos classes: écoutable / non écoutable. De plus, nous importons le csv contenant les informations relatives à chaque fichier audio: nom du fichier / médiane (note) / audible ou non (booléen).
Nous implémentons une colonne “fold” qui va assigner un nombre entre 1 et 10 inclus à chacun des samples.

Lors de l'extraction des embeddings des données WAV, nous obtenons un tableau de forme (N, 1024) où N est le nombre de "frames" que YAMNet a trouvé (un pour chaque 0,48 secondes d'audio).
Notre modèle utilisera chaque **frame comme une entrée**. Par conséquent, nous devons créer une nouvelle colonne qui contient un frame par ligne. Nous devons également développer les labels et la colonne de `*fold*` pour refléter correctement ces nouvelles lignes.
La colonne de `*fold*` conserve les valeurs originales. Nous ne pouvons pas mélanger les frames car, lorsque nous effectuons les diviser, nous risquerions d'avoir des parties du même fichier audio sur différentes séparations, ce qui rendrait nos étapes de validation et de test moins efficaces.

Passons maintenant à la division de notre dataset. Nous utiliserons la colonne `*fold*` pour **diviser le jeu de données** en jeux de train, de validation et de test. On peut ensuite drop la colonne `*fold*` car elle ne va plus nous servir pour l’entrainement.

Le modèle

Maintenant que nous avons pre-process nos data nous pouvons passer à la **construction du modèle**.
Ce sera un simple **modèle séquentiel** avec une couche cachée et une autre couche cachée avec **2 sorties** pour reconnaître si les sons sont audibles ou non.

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_34 (Dense)            (None, 512)               524800

 dense_35 (Dense)            (None, 2)                 1026

=================================================================
Total params: 525,826
Trainable params: 525,826
Non-trainable params: 0
_________________________________________________________________
```

```
my_model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                 optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
                 metrics=['accuracy'])

callback = tf.keras.callbacks.EarlyStopping(monitor='loss',
                                            patience=3,
                                            restore_best_weights=True)
```

Pour ce qui est de la compilation, ce sera la *sparse categorical cross-entropy* que nous allons utiliser, ainsi que *l'optimizer* **Adam** et l'*accuracy* en tant que mesure. Nous avons aussi choisi d'implémenter un *EarlyStopping* callback afin de prévenir d'un quelconque comportement d'overfitting.

Analyse des résultats

```
Epoch 50/50
32/32 [==============================] - 0s 13ms/step - loss: 0.2324 - accuracy: 0.9149 - val_loss: 0.1575 - val_accuracy: 0.9379
```

Nous obtenons de bon résultats (entre 90 et 95% d'*accuracy*) pour le train et le val set.

Dans le but d'avoir des **résultats plus fiables** nous avons comparés les résultats que nous avons obtenus avec différentes « *random seed* ». Cela va nous permettre d'observer des résultats obtenus avec une **différentes répartitions** de nos éléments entre les **échantillons de d'entrainement et de test.** L'idée est de réduire ce caractère un peu aléatoire en ayant un résultat sur un plus grand nombre d'éléments.

```
seed : 14 ||  loss: 0.2448 - accuracy: 0.9225 - val_loss: 0.3430 - val_accuracy: 0.8867
seed : 15 ||  loss: 0.2531 - accuracy: 0.9215 - val_loss: 0.3128 - val_accuracy: 0.9007
seed : 16 ||  loss: 0.2253 - accuracy: 0.9298 - val_loss: 0.5294 - val_accuracy: 0.8267
seed : 17 ||  loss: 0.2402 - accuracy: 0.9224 - val_loss: 0.4097 - val_accuracy: 0.8727
seed : 18 ||  loss: 0.2551 - accuracy: 0.9208 - val_loss: 0.2813 - val_accuracy: 0.9091
seed : 19 ||  loss: 0.2442 - accuracy: 0.9229 - val_loss: 0.2252 - val_accuracy: 0.9269
seed : 20 ||  loss: 0.2579 - accuracy: 0.9173 - val_loss: 0.2356 - val_accuracy: 0.9234
```

Figure 12: exemple des loss et des accuracies que l'on peut obtenir selon les seeds

En moyenne, sur une vingtaine de seed, nous obtenons :

	loss	accuracy	val loss	val accuracy
Moyenne :	0,25257	0,920635	0,2645	0,913685
Médiane :	0,25365	0,9208	0,2342	0,92125

Figure 13: médiane et moyenne des différentes valeurs pour 20 seeds

```
6/6 [==============================] - 0s 15ms/step - loss: 0.1555 - accuracy: 0.9314
Loss:  0.15547522902488708
Accuracy:  0.9314285516738892
```

On peut ensuite vérifier qu'il n'y pas d'overfitting en utilisant la méthode evaluate sur le test set. Le résultat est concluant.
Notre modèle fonctionne lorsque nous lui fournissons les **embeddings en entrée**.
Dans un scénario du monde réel, nous voudrions utiliser des données audios comme entrée directe.
Pour ce faire, nous allons **combiner YAMNet avec notre modèle afin de créer un seul modèle** que nous pourrions exporter pour d'autres applications.

Pour faciliter l'utilisation du résultat du modèle, la dernière couche sera une opération *reduce_mean*. Lorsque nous utilisons ce modèle pour le traitement, nous avons besoin du **nom de la couche finale**. Si nous n'en définissons pas, TensorFlow en définira automatiquement un incrémentiel qui rendra les tests difficiles, car il changera à chaque fois que nous entraînerons le modèle.
Lorsque nous utilisons une opération TensorFlow brute, nous ne pouvons pas lui attribuer de nom. Pour résoudre ce problème, nous allons créer une couche personnalisée qui appliquera *reduce_mean* et l'appeler " **classifier** ".

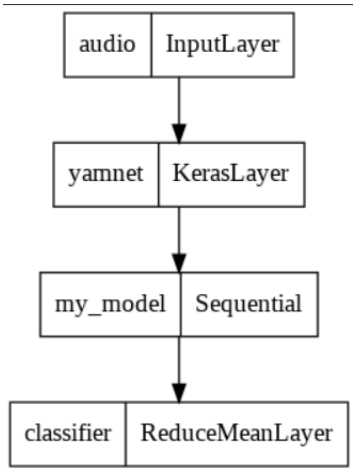
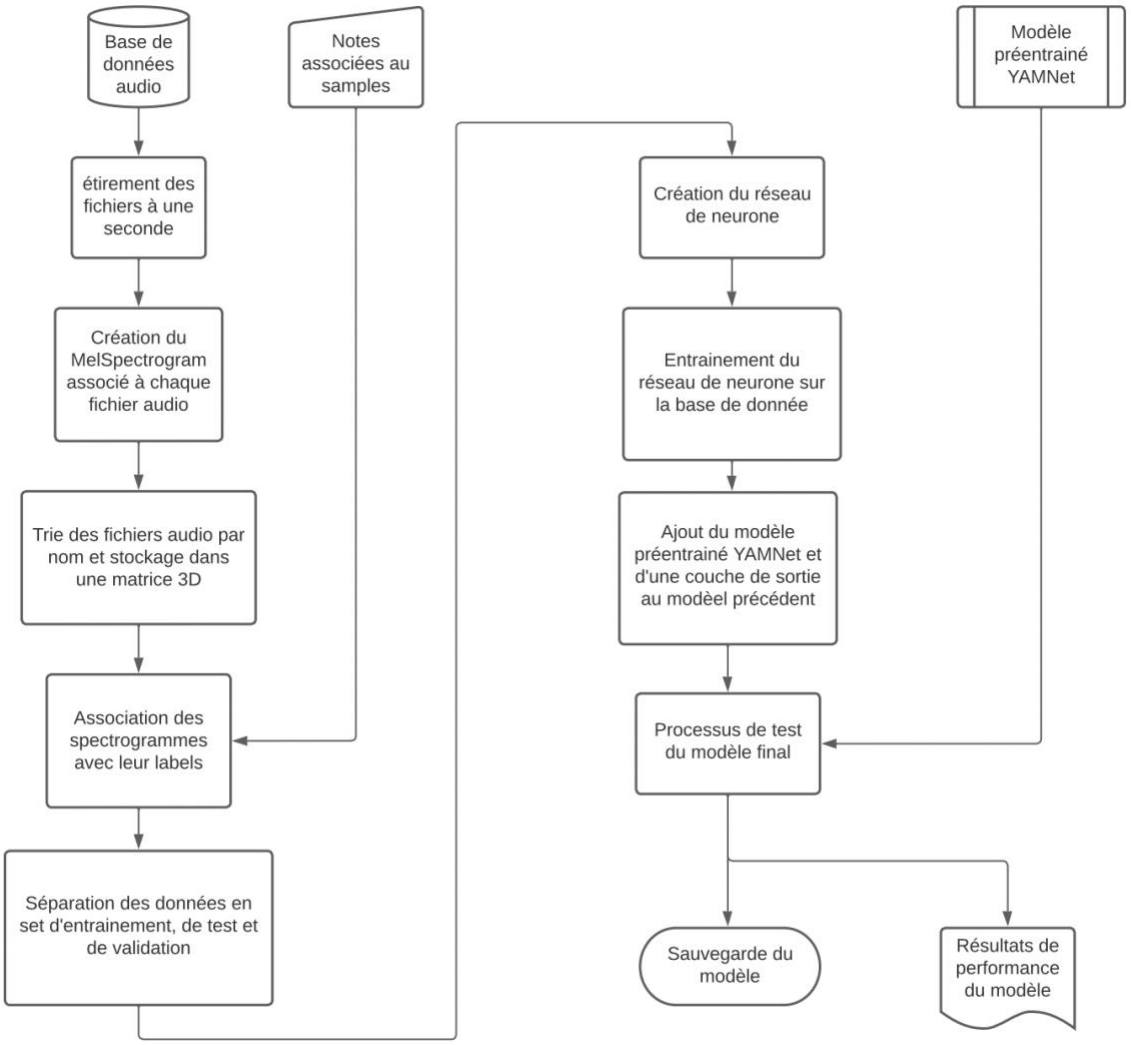


Figure 14: Représentation du modèle

Après ceci, nous pouvons aller plus loin en comparant les scores de YAMNet et de notre modèle (crée à partir de YAMNet) sur nos samples audio.
Le résultat donnera ainsi **une sortie de "catégorie"** (parole, sirène, animal…) et **une sortie binaire (audible ou non).**
Le mix de ces 2 modèles est une piste intéressante pour la suite de ce projet.

Organigramme de programmation de l'entrainement du transfert learning



Comparaison des modèles

Model	Accuracy	Val_Accuracy	Loss	Val_Loss	Epochs	Seed	
CNN	0.8562	0.923	0.3533	0.242	?	15	Moyenne
	0.849	0.926	0.363	0.237	?	15	Médiane
Transfer Learning	0.92	0.914	0.252	0.264	?	20	Moyenne
	0.92	0.921	0.253	0.234	?	20	Médiane
XGBoost	0.828						Moyenne
	0.831						Médiane

Pour conclure sur les différents modèles, nous voyons que les deux modèles les plus performant sont le **CNN** et le **Transfer Learning** car ils proposent tous les deux une *val_accuracy* d'environ 92 %.

Cependant, lorsqu'on regarde la précision sur la base d'entraînement, on observe que le Transfer Learning surpasse le CNN de quasiment 7 %. Ceci est indicatif que le modèle CNN souffre d'un problème d'*underfitting*, il a du mal à trouver des corrélations intéressantes sur la base d'entraînement. Ainsi, bien qu'ayant le même score, le **Transfer Learning** sera probablement **plus fiable** étant donné un échantillon audio aléatoire.

Le modèle XGBoost est aussi une piste qui pourrait être approfondie dans le futur cependant les premiers résultats obtenus avec ce modèle n'ont pas été assez convaincant pour que nous y dédiions une partie de notre temps.

Pour conclure le **Transfer Learning** est le meilleur modèle ayant été créé dans ce projet, mais tous les modèles ici sont possiblement améliorable, une résolution du problème d'*underfitting* du CNN pourrait résulter en une amélioration considérable de ses résultats.

Sources

2201.02490.pdf (arxiv.org)

Audio Feature Extraction (devopedia.org)

https://larevueia.fr/machine-learning-pour-la-classification-automatique-de-musiques-avec-python/

https://datascientest.com/transfer-learning

https://github.com/tensorflow/models/tree/master/research/audioset/yamnet

Étudier la musique sous toutes ses formes : la démultiplication des approches scientifiques en musicologie (openedition.org)

index.pdf (ircam.fr)

https://towardsdatascience.com/how-to-start-implementing-machine-learning-to-music-4bd2edccce1f

https://sigsep.github.io/datasets/musdb.html

https://github.com/deezer/spleeter