UQAC
Université du Québec
à Chicoutimi

Rapport
Mini-Projet de Conception sur la reconnaissance de formes
Automne 2022

Intelligence artificiel et reconnaissance de formes
(6GEI608)

Département des Sciences Appliquées
Module d'ingénierie

Travail d'équipe

RÉDACTEUR

Maxime Simard
SIMM26050001

Date de remise : 16 12 2022

Table des matières

Liste des figures

Introduction

Dans ce mini-projet de conception, je devrai :

- Effectuer une implémentation « fonctionnelle » dans le domaine de l'intelligence artificielle, plus précisément en reconnaissance de formes;
- Se familiariser et expérimenter avec des algorithmes d'apprentissage supervisé dans le but d'effectuer de la reconnaissance de formes;

Ce mini-projet de conception consistera à faire la planification et la conception d'un réseau de neurone permettant de faire la classification de pistes audio selon 10 genres musicales. Ce rapport expliquera en profondeur les différentes méthodes de traitement audio, de prétraitement des données et comparera les performances d'un perceptron multicouche à un CNN (Convolution Neural Network) dans le domaine de l'analyse audio.

Information générale

Un fichier « README.md » est disponible dans le dossier « Scripts », expliquant comment mettre en place et faire fonctionner le projet.

Recherche

Préparation au projet

Reconnaissance audio simple (1 heure)

Pour me préparer au projet, j'ai décidé de tout d'abord m'initier au traitement audio par intelligence artificielle. J'ai tout d'abord suivi le projet d'introduction à la reconnaissance audio simple offert par TensorFlow car il permet à la fois d'en apprendre plus sur le traitement audio, mais aussi sur la librairie TensorFlow, ses différentes classes et méthodes ainsi que son fonctionnement général. C'est dans cette introduction que j'ai entendu parler pour la première fois d'un spectrogramme, un graphique permettant de représenter un signal audio au travers du temps tout en ayant accès à l'information de chaque fréquence. Le projet d'introduction consiste à utiliser une banque de mot sous format « .wav » et à apprendre à un réseau de neurone à détecter quel mot est prononcé. Le tout m'a pris 1 heure à survoler et à analyser.

PyTorch pour traitement audio (3 heures)

Une fois cette introduction à TensorFlow terminé, j'ai voulu me renseigner sur les autres technologies que je pourrais possiblement utiliser pour faire mon projet. J'ai trouvé une librairie permettant de faire des projets d'intelligence artificielle semblable à TensorFlow, mais celle-ci est beaucoup plus populaire du côté recherche dans le domaine. J'ai donc suivi un cours d'introduction à PyTorch en traitement audio de 3 heures offert par Valerio Velardo. Valerio Velardo est une personne réputée dans le domaine de l'intelligence artificielle et se spécialise dans tout ce qui touche le traitement audio. Il a aussi la charge du projet en source ouverte « The Sound of AI ». Ce cours couvrait des sujets semblables au projet d'introduction de TensorFlow, mais comparé à leur méthode de développement, PyTorch encourage ses utilisateurs à créer eux-mêmes leur propre réseau de A à Z, y compris les méthodes de mise à jour des poids, et leur donne tous les outils nécessaires pour le faire. Bien que cette option eût l'air très formatrice et intéressante, j'ai décidé de me concentrer sur TensorFlow puisque cela me permettra de me concentrer plus sur la conception de mon architecture.

Apprentissage profond pour traitement audio avec Python (12 heures)

Je me suis replongé dans la recherche au sujet du traitement audio avec TensorFlow en suivant une autre formation de 12 heures de Valerio Velardo, qui celle-ci se concentrait sur le prétraitement des données, l'importation de ces dernières et l'implémentation d'un réseau de neurone utilisant TensorFlow. C'est dans cette formation que j'ai le plus appris sur le traitement audio en général, mais aussi de son utilité dans le domaine de l'intelligence artificielle et les différentes manières d'interpréter et de traiter un signal à l'aide d'un réseau de neurone. Dans ce cours, j'ai été introduit au concept des spectrogrammes MFCC qui permettent d'augmenter la rapidité d'apprentissage d'un réseau sans réduire sa précision. Même que parfois, cette méthode augmente la précision et réduit les pertes comparées aux performances d'un spectrogramme régulier. À la suite de cette formation, j'ai décidé qu'au cours du projet, je testerai les performances d'un perceptron multicouche comparé aux performances d'un CNN dans le domaine du traitement et de l'analyse audio, plus précisément dans la classification des genres musicales.

Augmentation de données audios (1 heures)

Pour compléter mon bagage en analyse et traitement audio, j'ai décidé de m'informer sur l'augmentation de données audios. Valerio Velardo offre aussi un cours de 1 heure explorant le sujet. Bien que ce sujet soit fort intéressant, j'ai pu constater qu'il serait difficile d'appliquer ces techniques sur mes données audios

puisque le « pitch » et la vitesse de la musique sont des données essentielles à l'indentification du genre musical. Si je modifie ces paramètres pour créer de nouvelles données, je risque d'accidentellement changer le genre de musique de la piste originale. Cependant, j'ai trouvé une augmentation possible qui est de segmenter les différentes pistes en petite section audio, ce qui permet de passer d'une piste à une centaine de segments de pistes par exemple.

Représentation de signal audio

L'ensemble des informations de cette section ont été acquises en lisant les articles suivants : « Learning from Audio: Wave Forms », « Learning from Audio: Fourier Transformations », « Learning from Audio: Spectrograms » et « Learning from Audio: The Mel Scale, Mel Spectrograms, and Mel Frequency Cepstral Coefficients ». Les liens vers ces articles sont disponibles dans la section « Références » du rapport.

Échantillonnage

Avant de pouvoir représenter un signal audio, il faut tout d'abord échantillonner ce signal. Pour ce faire, on doit choisir une fréquence d'échantillonnage (décidée selon nos besoins). Cette fréquence définira combien d'échantillon le signal audio aura par seconde. En d'autres mots, c'est le nombre de mesure prise par seconde. Plus ce nombre est élevée, plus le nombre de fréquences qui seront analysées sera élevé.

Spectre d'amplitude

Un spectre d'amplitude permet de représenter graphique l'amplitude en fonction du temps d'un signal. C'est la méthode la plus simple de visualiser un signal audio. Cette manière a cependant ses limitations, notamment qu'elle ne présente que l'amplitude moyenne de toutes les fréquences. Il est donc impossible d'identifier des tendances existantes dans les fréquences d'un signal. Voici un exemple de spectre d'amplitude :

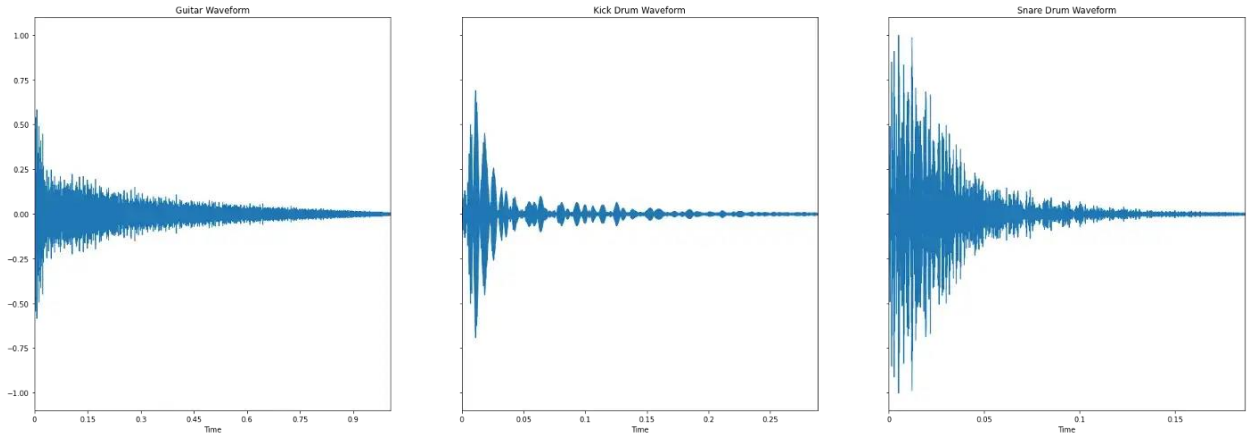

Figure 1 - Exemples de spectre d'amplitude

Dans la figure ci-dessus, on peut bien visualiser à quel moment le bruit est plus fort. On peut aussi voir différents pics qui représente différents échantillons.

Spectre de fréquence

Un spectre de fréquence représente les différentes amplitudes d'un signal audio selon chaque fréquence échantillonnée. Cette présentation nous permet donc de bien évaluer la présence et l'importance des fréquences dans le signal audio. Il peut être obtenu en calculant la transformé de Fourier du signal que l'on veut étudier.

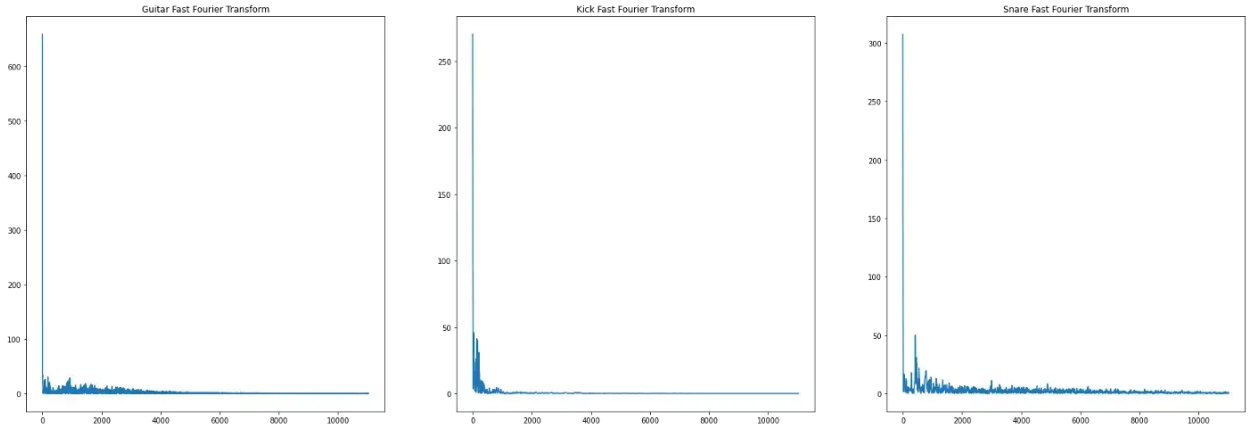

Figure 2 - Exemples de spectre de fréquence

Comme on peut le voir dans les exemples de la figure 2, les instruments de musique se trouvent généralement dans les plus basses fréquences. Cela rend donc inutile d'avoir une fréquence d'échantillonnage trop élevée.

Spectrogramme

Un spectrogramme combine l'information du spectre d'amplitude et celle du spectre de fréquence, ce qui résulte en un graphique 3d mettant en relation l'évolution de l'amplitude de chaque fréquence au fil du temps. Pour obtenir un spectrogramme, on doit utiliser un STFT (Short Time Fourier Transfer), qui permet de calculer la transformé de Fourier sur des courtes périodes tout au long du signal. Voici le résultat de cette opération :

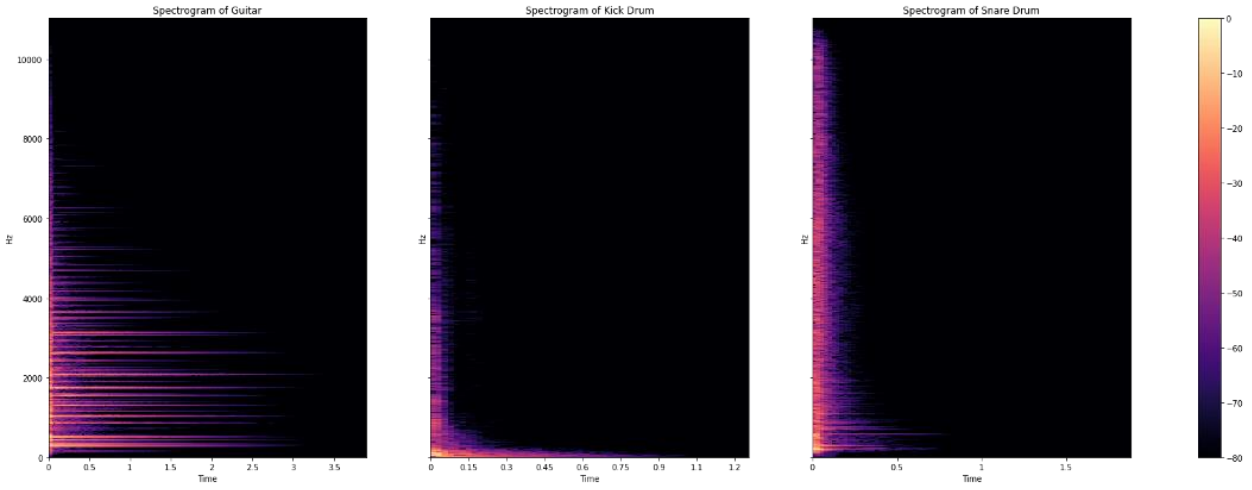

Figure 3 - Exemples de spectrogramme

La troisième dimension (l'amplitude) du graphique est présentée sous forme de teinte de couleur, soit noir étant un silence et blanc étant bruyant.

Spectrogramme MFCC

Un spectrogramme MFCC est une variation du spectrogramme présenté précédemment. La différence principale est qu'au lieu d'avoir 8000 fréquences à analyser, on se contente simplement de diviser ces 8000 fréquences en N groupes, et puis d'analyser ces groupes. En général, on utilise entre 13 et 20 groupes dépendant de la précision qu'on souhaite. On effectue finalement DCT (Discrete Cosine Transformation)

sur l'amplitude logarithmique du signal audio. Voici un exemple d'une conversion d'un spectrogramme en spectrogramme MFCC :

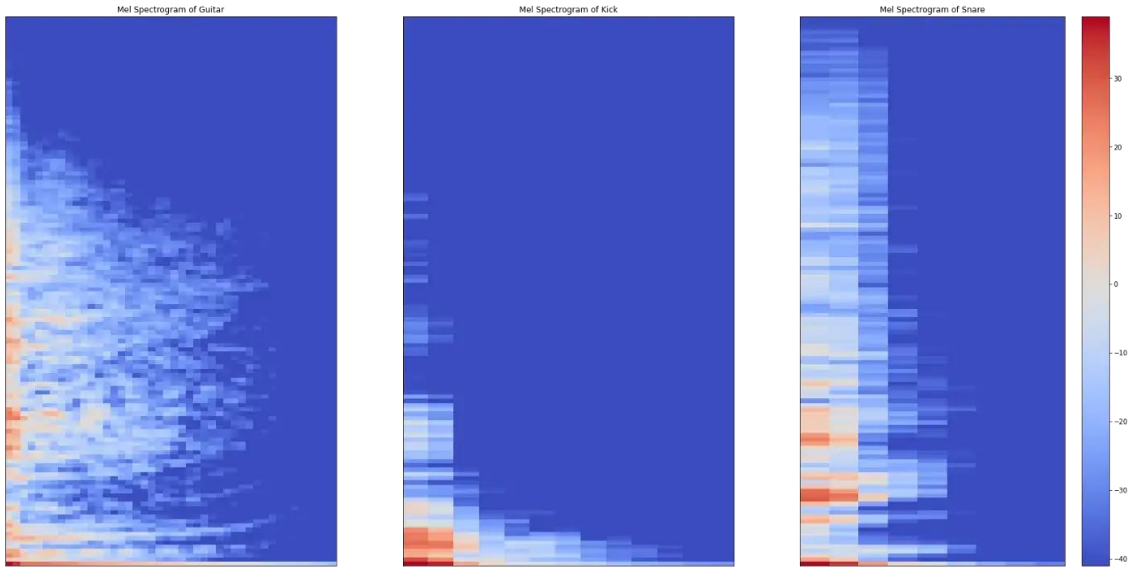

Figure 4 – Exemples de spectrogramme à convertir en MFCC

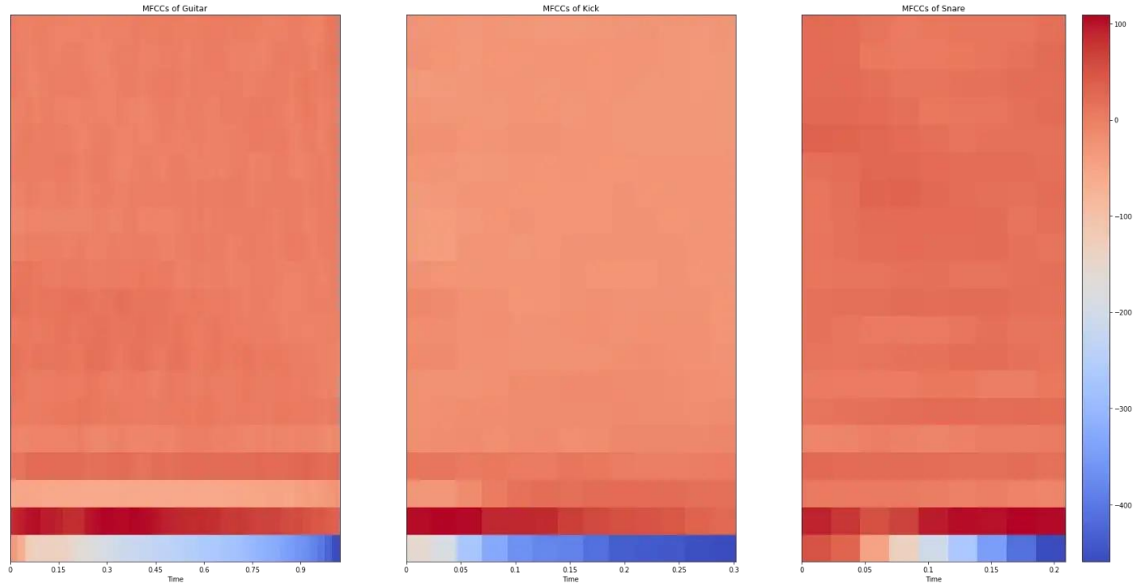

Figure 5 - Exemples de spectrogramme MFCC convertis

Source de données audios

Il est essentiel de trouver une source de données audios permettant de télécharger des pistes de tout genre. J'ai premièrement pensé à Spotify, un site de streaming réputé ayant un API très complet. Cependant, ce service ne permet pas au public de télécharger des pistes. Pour cette raison, je me suis redirigé vers un service grandement utilisé pour partager de la musique : YouTube. Il existe plusieurs librairies permettant de télécharger les vidéos qui se trouvent sur ce site web. De plus, on peut facilement faire des recherches par titre et trouver l'url d'une vidéo.

Genres musicaux

Afin de sélectionner les genres de musiques pour lesquels j'entrainerai le réseau de neurone, j'ai premièrement cherché une liste de genre musical. Le site web « https://www.musicgenreslist.com/ » permet de se faire une bonne idée de l'ensemble des genres et sous-genres musicaux existant. J'ai ensuite considéré prendre seulement les 10 genres les plus écoutés dans le monde selon une étude de 2018 conduite par IFPI (International Federation of the Phonographic Industry). Voici les résultats de cette étude :

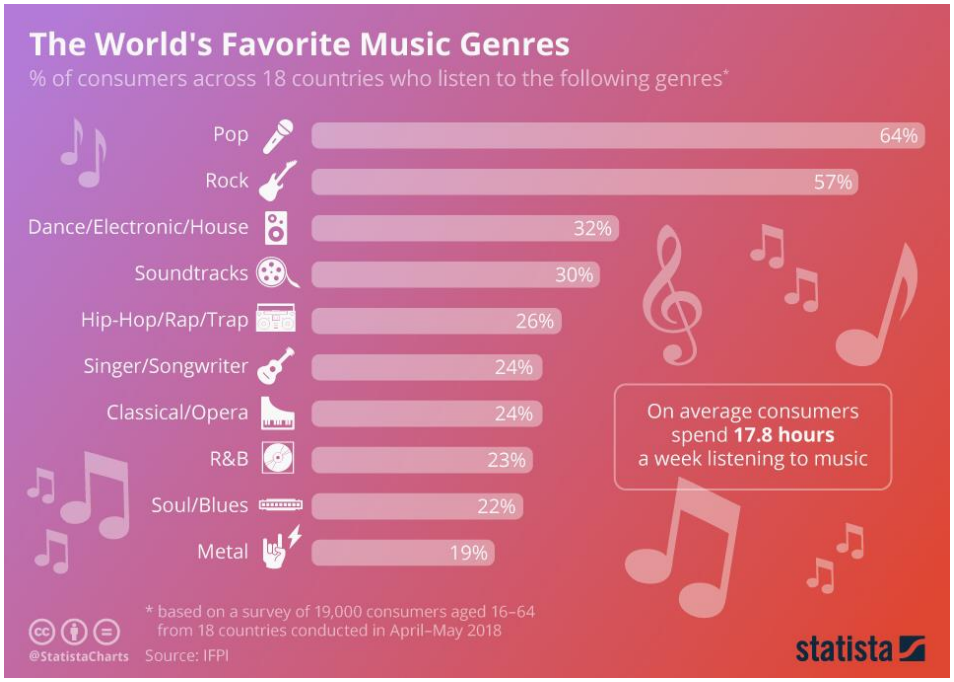

Figure 6 - Les genres de musique les plus populaire

J'ai finalement décidé de prendre un mix de certains des genres les plus écoutés, mais aussi des genres de musique un peu plus obscure. Cela me permettra de voir comment l'intelligence artificielle réagira à un mix de genres globaux et de sous-genres qui sont très proches de ces genres globaux. Voici la liste finale des genres utilisés :

- Rock
- Alternatif
- Classique
- Midwest Emo
- Hip-Hop / Rap
- Cloud Rap
- Pop
- Hyperpop
- Electronique
- Shoegaze

Développement et implémentation

Outil de téléchargement YouTube

Pour avoir une grande quantité de données audio, il serait pratique d'avoir un outil permettant le télécharger une grande quantité de fichier audio rapidement et les classés selon leur genre. Pour ce faire, j'ai trouvé deux librairies : PyTube qui permet le téléchargement de vidéo YouTube et YTMusicAPI qui permet la recherche de vidéo par titre et auteur. L'aspect de classification peut être résolu en mettant en place un format de fichier Excel ayant des colonnes pour le genre musical, l'url de la vidéo et le titre de la piste.

	A	B	C
1	genre	url	title
2	Rock	https://music.youtube.com/watch?v=7TDeBi34(	Smells like teen spirit
3	Rock	https://music.youtube.com/watch?v=hpSrLjc5S	Wonderwall
4	Rock	https://music.youtube.com/watch?v=yGxHDmH	Zombie
5	Rock	https://music.youtube.com/watch?v=X791lzOwt	Stairway to heaven
6	Rock	https://music.youtube.com/watch?v=A__cH65V	Californication
7	Rock	https://music.youtube.com/watch?v=liRMxUT8/	Somebody told me
8	Rock	https://music.youtube.com/watch?v=f1ldyrhOrG	Come as You Are
9	Rock	https://music.youtube.com/watch?v=wSenWOy	Plug in Baby
10	Rock	https://music.youtube.com/watch?v=AxuTd9rwf	Everlong
11	Rock	https://music.youtube.com/watch?v=bY3vXr7fm	It's My Life
12	Alternatif	https://music.youtube.com/watch?v=YdRjNeu1	Reptilia
13	Alternatif	https://music.youtube.com/watch?v=nbCOAPR	Karma Police
14	Alternatif	https://music.youtube.com/watch?v=6FnbPTao	You Only Live Once
15	Alternatif	https://music.youtube.com/watch?v=VredAgNS	Teddy Picker
16	Alternatif	https://music.youtube.com/watch?v=h_JR9klw3	Take Me Out
17	Alternatif	https://music.youtube.com/watch?v=phDGqlvw/	Last Nite
18	Alternatif	https://music.youtube.com/watch?v=CvjRIYpXS	Jigsaw Falling Into Place
19	Alternatif	https://music.youtube.com/watch?v=H8bNHRV	From the Ritz to the Rubble
20	Alternatif	https://music.youtube.com/watch?v=9RfVp-GhI	Creep
21	Alternatif	https://music.youtube.com/watch?v=hO4-QAjk	12:51
22	Classic	https://music.youtube.com/watch?v=P4159FQ3	Clair de Lune
23	Classic	https://music.youtube.com/watch?v=VKGRssiq	Rêverie
24	Classic	https://music.youtube.com/watch?v=9_C6CTs0	Moonlight Sonata
25	Classic	https://music.youtube.com/watch?v=3cqsIqYFH	Bolero

Figure 7 - 25 premières lignes du fichier Excel

À l'aide de la librairie Pandas, on peut facile extraire cette information ligne par ligne, effectuer une recherche YouTube si l'url est manquante, télécharger la vidéo par url et la déposer dans le dossier approprié au genre audio. Si la vidéo existe déjà dans le dossier du genre, on passe à la prochaine ligne. Cela permet d'accélérer le traitement du fichier et éviter des tâches inutiles.

J'ai aussi considéré ajouter une recherche Spotify pour associer une piste avec un genre musical un peu plus officiellement à l'aide de la librairie Tekore, mais l'API Spotify ne retourne seulement des genres musicaux pour les artistes et non les titres. Puisque le genre de chaque piste sera décidé par moi, il y aura une possibilité d'erreur et de biais qui pourrait porter certains genres à être mal représentés.

Prétraitement

Conversion des fichiers

Après avoir expérimenté avec la librairie Librosa permettant de lire et de faire du traitement sur des fichiers audios, il est beaucoup plus facile de travailler avec des fichiers « .wav ». Les fichiers télécharger sur YouTube sont des fichiers « .mp4 », et il serait donc pertinent de les convertir avant de faire du traitement

audio sur ces derniers pour réduire la durée lors de l'extraction des données. Pour convertir un fichier, on utilise la fonction suivante de FFMPEG en console :

```
ffmpeg -i fichier.mp4 -ac 1 -ar 16000 -f wav fichier.wav
```

Cette commande convertit le fichier en « .wav » mais permet aussi de passer d'un fichier audio stéréo à mono, et de changer la fréquence d'échantillonnage à 16000Hz. L'ensemble de ces changements augmenterons grandement l'efficacité et la rapidité du module de traitement de données. On peut finalement appeler cette commande à l'aide de la fonction « system() » de la librairie os de Python.

Troncature

Afin d'éliminer des sections inutiles souvent retrouver dans les débuts et fins des audios YouTube, on utilise la fonction « trim() » du module « effects » de Librosa, qui permet de retirer tout ce qui est en-dessous de 60dB au début et à la fin du signal audio. Une fois cette troncature effectuée, on peut passer à la segmentation de l'audio.

Segmentation

La segmentation audio permet d'augmenter le nombre de données disponible pour l'entrainement. Pour se faire, on détermine la durée souhaitée de chaque segment (2.5 secondes dans ce projet). Ensuite, on multiplie cette durée par la fréquence d'échantillonnage de l'audio (16kHz dans ce projet). On obtient donc le nombre d'échantillon dans cette durée. Maintenant, à l'aide de ce nombre, on peut commencer de l'échantillon 0 et trouver la fin du segment en y ajoutant le nombre d'échantillon par segment. On peut finalement couper le segment de la façon suivante :

```
segment = trimmed_signal[start:finish]
```

Pour créer d'autres segments, on peut simplement définir l'échantillon de début du nouveau segment à l'échantillon de fin du dernier segment, et itérer sur la piste tronquée au complet.

Extraction de données

Génération des MFCC

Dans le but d'avoir des données représentant les fréquences et leur amplitude d'une manière compacte, il est nécessaire de transformer les spectres audios d'amplitude en graphique MFCC. Il faut tout d'abord décider le nombre de groupe MFCC dans le graphique. Plus ce nombre est élevé, plus le graphique représentera les fréquences avec précision. Cependant, le temps de traitement et l'espace mémoire nécessaire seront plus élevés. Il faut aussi définir le nombre de segment compris dans la FFT ainsi que la distance de saut entre chaque FFT. Ces deux paramètres sont la différence entre une FFT (Fast Fourier Transform) et une STFT (Short Time Fourier Transform). On peut finalement donner l'ensemble de ces paramètres à la fonction « mfcc() » du module feature de Librosa. Avant d'ajouter les MFCC calculés à nos données, on vérifie que le nombre de MFCC correspond bien au nombre de MFCC prévu par segment, ce qui assure donc que l'ensemble de nos données sont du même format (essentiel pour l'entrée du réseau de neurone).

Normalisation des données

Il est important d'assurer que l'ensemble de données utilisé pour entrainer le réseau de neurone ait une répartition équitable de chaque genre musical. Puisqu'il est très difficile de prévoir rapidement combien de segments seront créés par une piste, il a été beaucoup plus facile d'assurer une répartition égale en implémentant un simple algorithme. Premièrement, on trouve le genre avec le plus petit nombre de segment MFCC. Une fois trouvé, on choisit aléatoirement dans chaque genre ce nombre de segment. Cela permet d'éliminer l'inégalité créer par des pistes de classique qui ont parfois 250 segments et des pistes de pop qui n'ont que 75 segments. La sélection aléatoire des segments permet d'assurer que l'on couvre tout de même, le plus possible, toutes les pistes qui avaient été mises dans la liste de données audios au lieu de simplement prendre les premières. À l'aide de cette fonctionnalité, on évite une source d'erreur majeure.

Application

Prédiction

Une fois le modèle entrainé, il serait intéressant de mettre en place une application permettant de donner une url d'une vidéo YouTube et de prédire le genre musical de l'audio qu'elle contient. Pour ce faire, il faut avoir une fonction permettant la prédiction d'une piste complète au lieu de segments de 2.5 secondes. La

solution est de diviser cette piste en segment, puis prédire le genre de chaque segment. À l'aide de la sortie, on peut estimer, et donc prédire, le genre musical de la piste complète. J'ai conçu deux manières qui utilisées et retournées à l'utilisateur afin de pouvoir évaluer laquelle obtient les meilleures prédictions. Les voici :

Méthode de la mode

La méthode de la mode commence par trouver la valeur maximale des prédictions de chaque segment. Une fois une liste des valeurs maximales obtenue, on peut trouver la mode de cette liste, c'est-à-dire la valeur qui est présente le plus souvent au travers de tous les segments. Puisqu'une piste a, en moyenne, une centaine de segment, cette méthode est assez efficace.

Méthode de la somme

La méthode de la somme consiste à faire la sommation de l'ensemble des prédictions de chaque segment. Cela veut donc dire qu'on obtient une liste contenant la proportion totale de chaque genre de musique au travers de la piste analysée. Une fois cette liste obtenue, on peut simplement trouver la valeur la plus élevée qui s'y trouve. L'index de cette valeur sera le genre prédit. Un point important de cette méthode est qu'elle conserve les proportions des genres de chaque segment. Cela veut donc dire qu'une différence de 0.01 entre la prédiction en première place et en deuxième place n'élimine pas pour autant son importance dans l'équation.

Interface Web

Une simple interface web a été conçu avec Flask permettant de facilement envoyer une url à l'application pour son traitement. Une fois la réponse reçue de la part du serveur, on nous redirige vers une page affichant le contenu de la réponse sous format JSON. Depuis cette page, on peut voir le genre musical prédit par la méthode de la mode et la méthode de la somme. Cette interface est disponible à l'adresse « localhost » sur le port 5000.

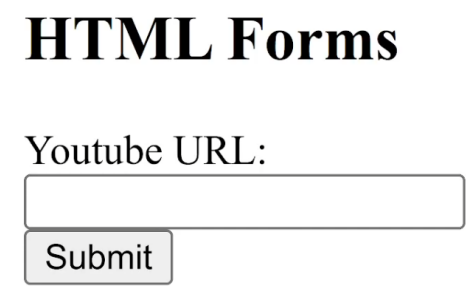

Figure 8 - Interface web de l'applcation

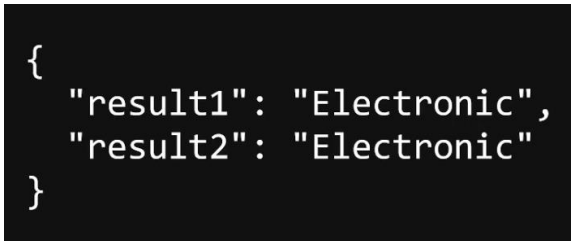

Figure 9 - Exemple de résultat pour la piste « Knights » de Crystal Castles

Traitement

Pour traiter les requêtes de prédiction, on commence tout d'abord par extraire de la requête l'url de la vidéo YouTube. Ensuite on télécharge la vidéo en question à l'aide de l'outil de téléchargement mentionné précédemment. Une fois la vidéo téléchargée et convertie en « .wav », on l'envoie au module d'extraction de données qui nous retourne le fichier audio en tant que liste de segment sous forme de spectrogramme MFCC. On donne finalement cette liste à notre fonction de prédiction avec le modèle à utiliser. Une fois la prédiction terminée, on retourne les résultats à l'interface web.

Expérimentations et ajustements

Répartition des données

Pour la durée totale de ce projet, le même ensemble de données sera utilisé. Voici sa répartition avant et après normalisation :

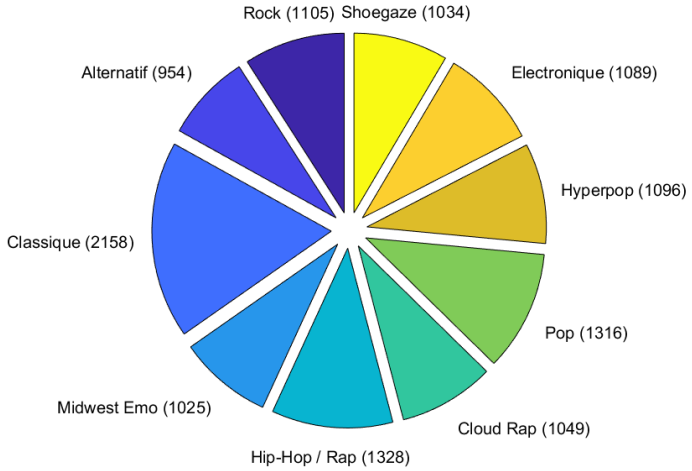

Figure 10 - Répartition des données avant normalisation

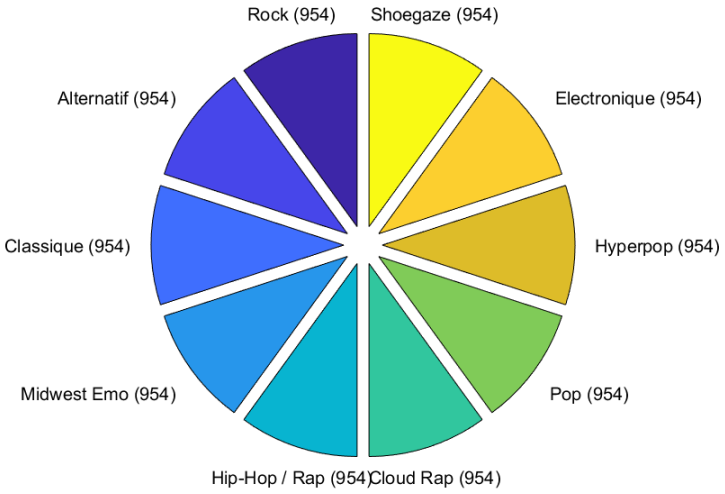

Figure 11 - Répartition des données après normalisation

De plus, on sépare ces données en trois parties :

- Données d'entrainement (70%)
- Données de validation (10%)
- Données de test (20%)

Paramètres d'entrainement

Les entrainements de réseaux effectués dans cette section ont tous été fait avec un « learning rate » de 0.0001 avec l'optimiseur « ADAM ».

Perceptron multicouche

Conceptions initiales

Dans la conception initiale du perceptron multicouche, il n'y avait qu'une couche d'entrée avec deux couches pleinement connectées de 512 et 256 neurones avec un fonction d'activation ReLU (Rectified Linear Unit), puis finalement une couche de sortie de 10 neurones (pour les 10 classes à identifier) avec une fonction d'activation « softmax » (ou fonction exponentielle normalisée). Voici le code pour construire cette architecture initiale avec TensorFlow :

```
# build network architecture
model = keras.Sequential([

    # input layer
    keras.layers.Flatten(input_shape=(input_shape[1], input_shape[2])),

    # 1st dense layer
    keras.layers.Dense(512, activation="relu"),

    # 2nd dense layer
    keras.layers.Dense(256, activation="relu"),

    # output layer
    keras.layers.Dense(10, activation="softmax")
])
```

Figure 12 - Architecture initiale du perceptron multicouche

Le premier changement qui a été fait est d'ajouter une couche supplémentaire de 64 neurones. Cette couche utilise aussi la fonction d'activation ReLU.

Le second changement a été d'ajouter des couches de « dropout » de 30% à chaque couche utilisant la fonction ReLU, ainsi que de leur ajouter un « kernel regularizer ».

Après un entrainement de 100 epochs avec des « batchs » de 32, le réseau de neurones a atteint les performances suivantes :

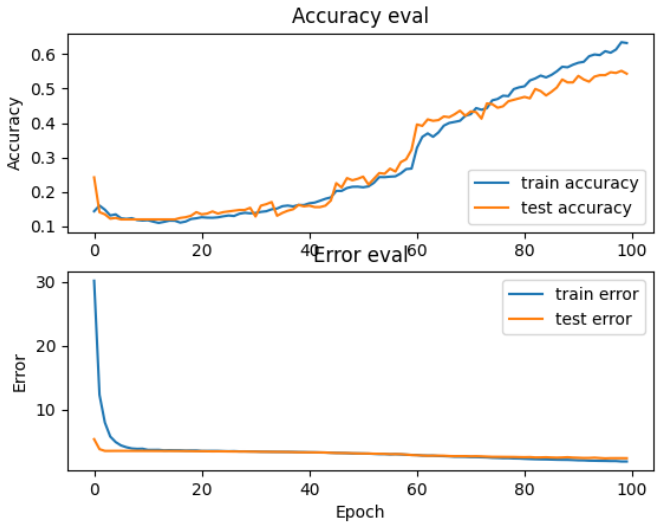

Figure 13 - Évolution de l'itération #1 du MLP

```
Validating model...
30/30 [==============================] - 0s 1ms/step - loss: 2.3035 - accuracy: 0.5462

Validation loss: 2.3035075664520264
Validation accuracy: 0.5462273955345154
```

Figure 14 - Performances de l'itération #1 du MLP

Avec un modèle de base, on réussit déjà à obtenir une précision de 54.6% et une perte de 2.3035. Ces performances sont loin d'être satisfaisantes, mais elles sont bonnes pour une toute première itération.

Augmentation du nombre de MFCC

Avant cette section, le nombre de MFCC utilisé était de 13, comme recommandé dans plusieurs articles concernant les MFCC et le traitement audio. J'étais curieux de voir les augmentations en performance si on doublait le nombre de MFCC, et donc doublait le nombre de données accessibles à l'intelligence artificielle.

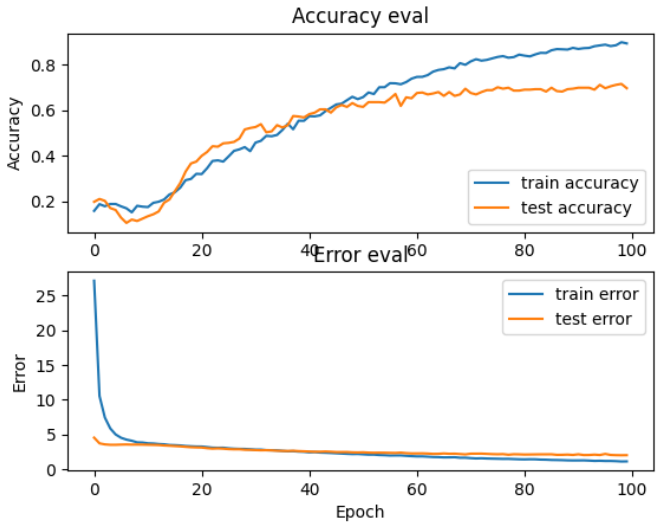

Figure 15 - Évolution de l'itération #2 du MLP

```
Validating model...
30/30 [==============================] - 0s 2ms/step - loss: 2.2224 - accuracy: 0.7003

Validation loss: 2.222438335418701
Validation accuracy: 0.7003188133239746
```

Figure 16 - Performances de l'itération #2 du MLP

La durée de l'entrainement est passée de 95 secondes à 158 secondes. C'est une augmentation de 166% de la durée d'entrainement. Ce ralentissement du processus d'entrainement n'est cependant pas si décevant car le nouveau modèle atteint maintenant une précision de 70.0% et une perte de 2.22. La perte n'a pas descendu beaucoup mais la précision a grandement augmenté grâce à ce changement. Cette modification sera alors retenue.

Durée de segment réduit

La durée initiale utilisée pour couper les segments était de 5 secondes. Je me suis demandé si avoir des segments plus petits permettrait de mieux caractériser les genres musicaux qu'on tente de classifier. Pour cette raison, j'ai décidé de couper le temps des segments en deux. De plus, cela doublerait le nombre de données disponibles à l'entrainement.

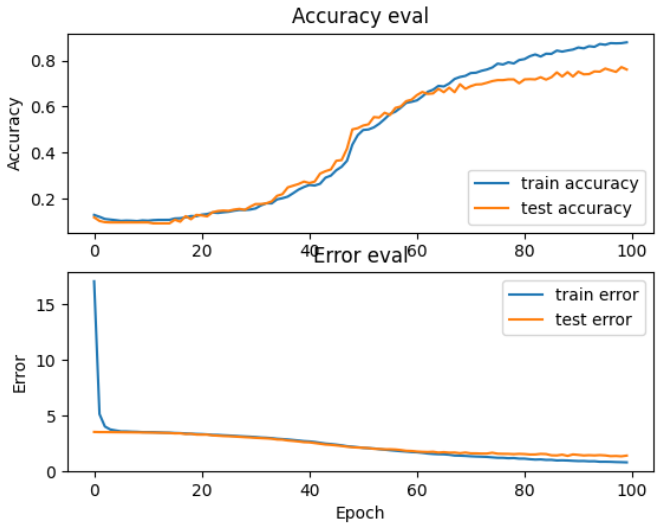

Figure 17 - Évolution de l'itération #3 du MLP

```
Validating model...
60/60 [==============================] - 0s 7ms/step - loss: 1.5020 - accuracy: 0.7480

Validation loss: 1.5019890069961548
Validation accuracy: 0.7480148077011108
```

Figure 18 - Performances de l'itération #3 du MLP

Tout comme dans l'itération précédente, la précision à augmenter. De plus, la perte est passée de 2.22 à 1.50. Ce changement est extrêmement grand et me porte à croire qu'il est mieux d'utiliser des segments de plus court pour la suite du projet. On peut aussi voir que l'allure du graphique de précision et de perte est beaucoup plus portée à évoluer puisque les pentes sont encore en train d'augmenter tandis que dans l'itération précédente nous avions commencé à stagner.

Taux d'échantillonnage plus élevée

Après avoir effectué quelque recherche, j'ai trouvé que des compagnies qui se spécialise dans le domaine de musique utilise généralement une fréquence d'échantillonnage de 22050Hz pour analyser les signaux audios afin de voir la différence que cela pourrait faire. Mon hypothèse est que ce changement n'aura presque aucun impact.

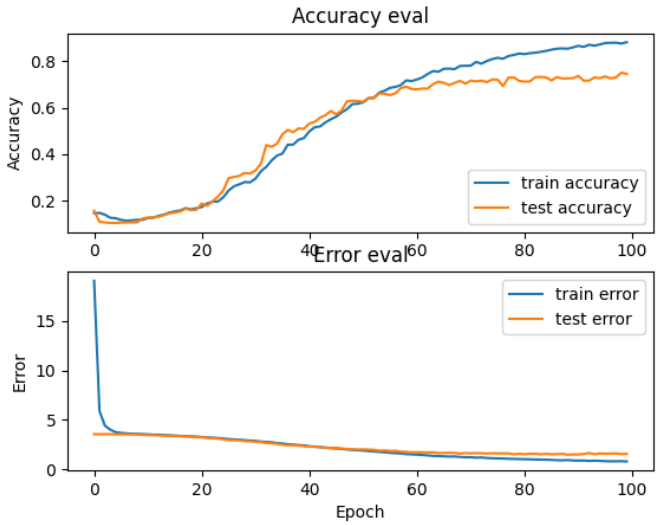

Figure 19 - Évolution de l'itération #4 du MLP

```
Validating model...
58/58 [==============================] - 0s 1ms/step - loss: 1.6898 - accuracy: 0.7276

Validation loss: 1.689765214920044
Validation accuracy: 0.7276159524917603
```

Figure 20 - Performances de l'itération #4 du MLP

Les performances à 22050Hz ont été décevantes. Bien que l'on eût maintenant un plus grand nombre de données à évaluer, le modèle a performé avec une précision 5% moins élevée que la dernière itération. De plus, la courbe d'apprentissage à l'air de stagner plus tôt. On rejettera donc ce changement.

Augmentation des « batchs »

Afin d'augmenter la précision de notre méthode de rétropropagation, j'ai décidé d'augmenter la taille des « batchs » à 64. En conséquence, j'ai aussi dû augmenter le nombre d'epochs à 200.

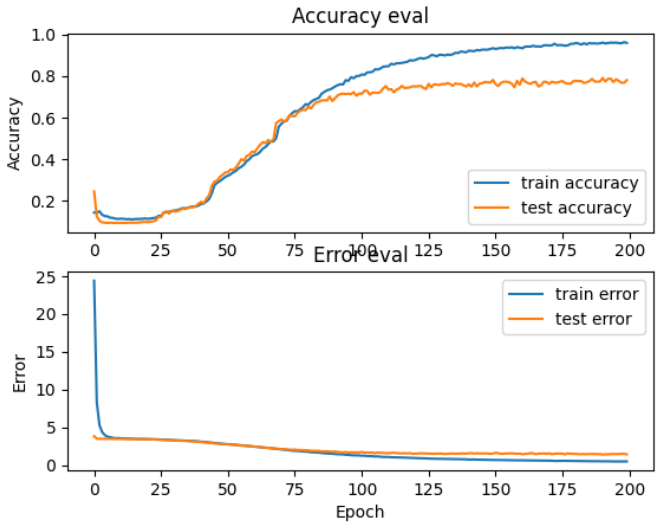

Figure 21 - Évolution de l'itération #5 du MLP

```
Validating model...
60/60 [==============================] - 0s 2ms/step - loss: 1.6472 - accuracy: 0.7628

Validation loss: 1.647163987159729
Validation accuracy: 0.7628374695777893
```

Figure 22 - Performances de l'itération #5 du MLP

Cette itération a une précision un peu plus haute que celle de l'itération #3, mais on peut voir que la perte est cependant un peu plus haute. Nous allons tout de même conserver cette itération puisque je crois qu'avec un peu plus de méthodes évitant « l'overfitting », ou plus de données, ce changement sera bénéfique au projet.

Ajout d'une couche pleinement connectée

Pour la dernière itération, j'ai essayé d'ajouter une couche de 1024 neurones au réseau afin de voir si cela aiderait à augmenter les performances et à traiter l'information. J'ai décider de garder le même modèle de couche que les trois autres déjà présentes dans le model.

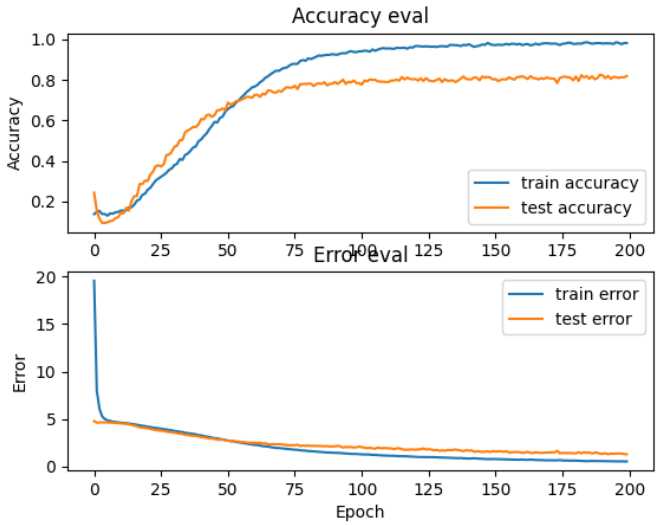

Figure 23 - Évolution de l'itération #6 du MLP

```
Validating model...
60/60 [==============================] - 0s 3ms/step - loss: 1.4763 - accuracy: 0.7909

Validation loss: 1.4762897491455078
Validation accuracy: 0.790894627571106
```

Figure 24 - Performances de l'itération #6 du MLP

À l'aide de la nouvelle couche, on peut voir que le réseau a obtenu les meilleurs résultats jusqu'à présent. En effet, la précision du réseau est encore plus élevée, à une valeur de 79.1%, et la perte est plus basse que celle de l'itération #4 (l'itération qui avait eu la meilleure perte jusqu'à présent), à une valeur de 1.4763. Le temps d'entrainement à cependant augmenté, passant de 168 secondes à 357 secondes. Ce temps d'entrainement est encore très acceptable pour le cadre du projet. Cette itération sera donc la dernière en ce qui est de la conception du perceptron multicouche.

Architecture finale

L'architecture finale du perceptron multicouche est basée sur l'itération #6 qui a une précision de 79.1%, une perte de 1.48 et un temps d'entrainement de 6 minutes. Voici le code pour construire cette architecture avec TensorFlow :

```
# build network architecture
model = keras.Sequential([

    # input layer
    keras.layers.Flatten(input_shape=(input_shape[1], input_shape[2])),

    # 1st dense layer
    keras.layers.Dense(1024, activation="relu",
                       kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # 2nd dense layer
    keras.layers.Dense(512, activation="relu",
                       kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # 3rd dense layer
    keras.layers.Dense(256, activation="relu",
                       kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # 4th dense layer
    keras.layers.Dense(64, activation="relu",
                       kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # output layer
    keras.layers.Dense(10, activation="softmax")
])
```

Figure 25 - Architecture finale du perceptron multicouche

CNN

Conceptions initiales

Contrairement au réseau du perceptron multicouche, l'architecture du CNN a commencé en étant déjà avancé puisque j'ai pu apprendre des manipulations et des essais précédents. Son architecture initiale consiste d'une couche de 3 couches de convolution utilisant 32 kernels de 3x3, une activation « ReLU », un « max pooling » de 3x3 avec des bonds de 2x2 avec un « padding » assurant que la sortie ait la même taille que l'entrée, puis une normalisation de « batch ». Après les couches de convolution, on aplatit notre sortie afin d'obtenir une liste des résultats, puis on utilise une couche de 64 neurones avec une activation « ReLU » et un « dropout » de 30%. Finalement, on a une couche de sortie de 10 neurones. Voici le code pour construire cette architecture initiale avec TensorFlow :

```python
# build network topology
model = keras.Sequential()

# 1st conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 2nd conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 3rd conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# flatten output and feed it into dense layer
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))

# output layer
model.add(keras.layers.Dense(10, activation='softmax'))
```

Figure 26 - Architecture initiale du CNN

Voici le résultat de l'entrainement avec cette architecture :

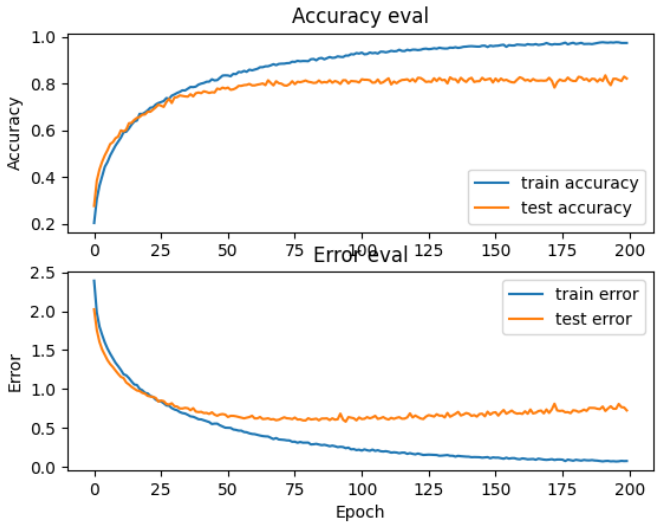

Figure 27 - Évolution de l'itération #1 du CNN

```
Validating model...
30/30 [==============================] - 0s 7ms/step - loss: 0.7204 - accuracy: 0.8147

Validation loss: 0.7203702330589294
Validation accuracy: 0.8147167563438416
```

Figure 28 - Performances de l'itération #1 du CNN

Comme on peut le voir avec l'architecture initiale, le CNN est déjà beaucoup plus efficace que la dernière itération du perceptron multicouche. En effet, le CNN atteint une précision de 81.5% (2.4% plus élevée) et une perte de 0.72 (la moitié de la meilleure perte atteinte).

Ajout d'une condition de fin

Un problème qui est présent autant dans le CNN que dans les nombreuses itérations du MLP est « l'overfitting ». On continue souvent à entrainer notre modèle même s'il ne fait plus de progrès depuis de nombreuses epochs. Pour régler ce problème, j'ai décider d'ajouter une fonction de rappel (callback) qui permet de vérifier s'il y a réellement eu une amélioration de la perte dans les 10 dernières epochs. Cette méthode s'appelle un « early stop ».

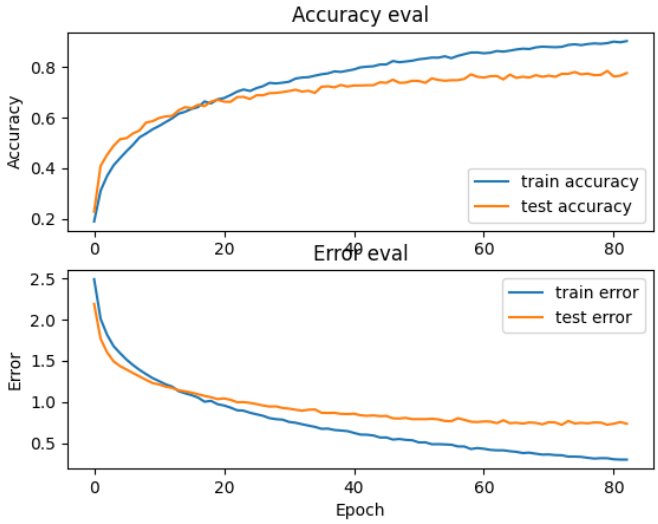

Figure 29 - Évolution de l'itération #2 du CNN

```
Validating model...
30/30 [==============================] - 0s 7ms/step - loss: 0.6831 - accuracy: 0.7946

Validation loss: 0.6831164956092834
Validation accuracy: 0.7946003079414368
```

Figure 30 - Performances de l'itération #2 du CNN

Le graphique nous indique que la fonction a bel et bien détecté un arrêt de progrès après 80 epochs. Cela réduit donc grandement le temps d'entrainement. La précision du modèle a cependant réduite à 79.5%, ce qui est toujours très acceptable, mais l'erreur elle a aussi réduite à 0.68, ce qui est selon moi plus pertinent. Cela assure donc que le modèle reste plus général au lieu de se perfectionner seulement sur les données sur lesquels il s'entraine.

Modification de la 3ème couche de convolution

J'ai décider d'expérimenter en modifiant la dernière couche de convolution. Pour se faire, j'ai changé la taille du kernel à 2x2 de même que la taille du « max pooling ».

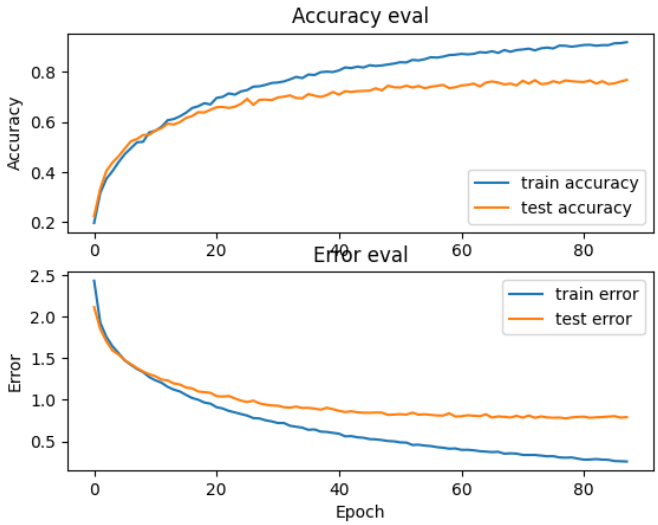

Figure 31 - Évolution de l'itération #3 du CNN

```
Validating model...
30/30 [==============================] - 0s 7ms/step - loss: 0.7039 - accuracy: 0.7840

Validation loss: 0.7038832306861877
Validation accuracy: 0.7840127348899841
```

Figure 32 - Performances de l'itération #3 du CNN

D'après les résultats, la précision et la perte ont tous les deux empiré à la suite de ce changement. Pour cette raison, cette itération sera rejetée.

Ajout d'un « kernel regularizer » à la couche pleinement connectée

Comme on peut le voir dans l'itération #2, la courbe d'apprentissage tend encore à stagner rapidement. Pour tenter d'augmenter le nombre d'epoch qui seront utiles à l'apprentissage du réseau, j'ai décidé d'ajouter une « kernel regularizer », un changement qui avait été bénéfique au MLP.

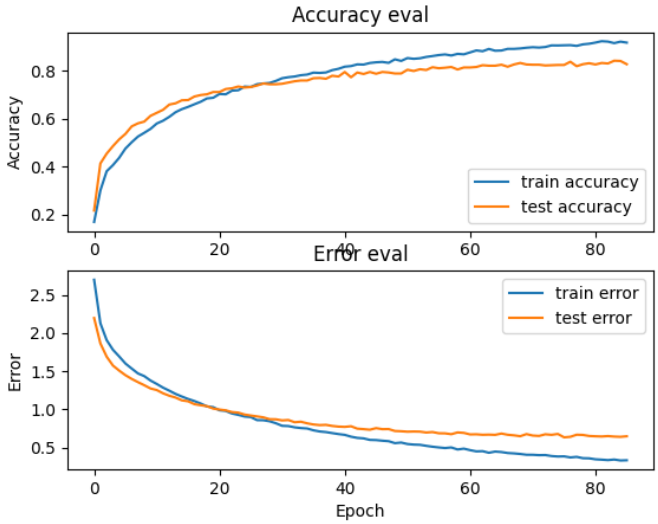

Figure 33 - Évolution de l'itération #4 du CNN

```
Validating model...
30/30 [==============================] - 0s 7ms/step - loss: 0.6965 - accuracy: 0.8237

Validation loss: 0.6964833736419678
Validation accuracy: 0.8237162232398987
```

Figure 34 - Performances de l'itération #4 du CNN

La courbe de précision est beaucoup plus intéressante que celle de l'itération #2 puisqu'on voit qu'elle est toujours en croissance. De plus, cette architecture a obtenu la précision la plus élevée jusqu'à présent avec une valeur de 82.4%. La perte est cependant un peu moins bonne que celle de l'itération #2 mais le changement est négligeable pour le gain de performance.

Ajout d'une deuxième couche pleinement connectée

Après avoir effectué quelques recherches, j'ai pu constater que plusieurs implémentations de CNN utilise plus qu'une couche de neurones après les couches de convolutions. J'ai donc décidé d'ajouter une couche de 128 neurones afin d'observer son impact sur les résultats.

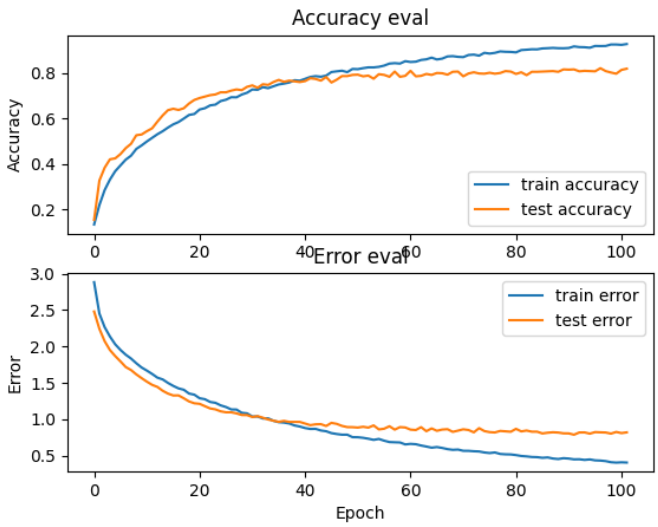

Figure 35 - Évolution de l'itération #5 du CNN

```
Validating model...
30/30 [==============================] - 0s 8ms/step - loss: 0.8859 - accuracy: 0.7946

Validation loss: 0.8859197497367859
Validation accuracy: 0.7946003079414368
```

Figure 36 - Performances de l'itération #5 du CNN

Encore une fois, ce changement n'a pas été bénéfique à l'apprentissage du réseau. On atteint une précision similaire aux itérations précédentes mais la perte est beaucoup plus élevée. Il est donc impossible d'ajouter ce changement au réseau.

Augmentation du nombre de neurones dans la couche pleinement connectée

Puisque l'ajout d'une nouvelle couche n'a pas aidé, j'ai décidé de tenter de grossir la couche déjà présente. Cette couche passe donc de 64 neurones à 256 neurones.

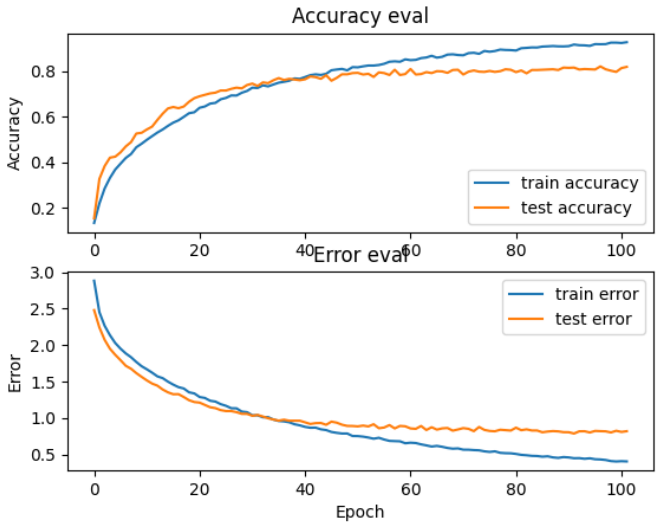

Figure 37 - Évolution de l'itération #6

```
Validating model...
30/30 [==============================] - 0s 7ms/step - loss: 0.7403 - accuracy: 0.8269

Validation loss: 0.7403151392936707
Validation accuracy: 0.8268925547599792
```

Figure 38 - Performances de l'itération #6

Cette fois-ci, la précision est un petit peu plus élevé (de 0.3%) que la meilleure itération, mais encore une fois la perte est aussi beaucoup plus élevée. Pour cette raison, cette itération est aussi rejetée.

Architecture finale

L'architecture finale du CNN est basée sur l'itération #4 qui a une précision de 82.3%, une perte de 0.69 et un temps d'entrainement de 4 minutes. Voici le code pour construire cette architecture avec TensorFlow :

```
# build network topology
model = keras.Sequential()

# 1st conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 2nd conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# 3rd conv layer
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

# flatten output and feed it into dense layer
model.add(keras.layers.Flatten())

# 1st dense layer
model.add(keras.layers.Dense(64, activation='relu',
                             kernel_regularizer=keras.regularizers.l2(0.001)))
model.add(keras.layers.Dropout(0.3))

# output layer
model.add(keras.layers.Dense(10, activation='softmax'))
```

Figure 39 - Architecture finale du CNN

Conclusion

En conclusion, le mini-projet de conception a été une réussite. J'ai réussi à mettre en place une implémentation fonctionnelle qui permet de reconnaître à quel genre de musique appartient une musique tout en me familiarisant à divers algorithmes en lien avec l'apprentissage supervisé. J'ai aussi eu l'occasion d'en apprendre plus sur le traitement audio, les manipulations possibles, mais aussi de mettre en pratique les algorithmes en question de traitement de signal. Ce projet m'a permis de connecter à la fois la théorie vue dans le cours « Intelligence artificielle et reconnaissance des formes », mais aussi celle vu dans le cours « Théorie et technique de la transmission de données ».

Après avoir expérimenté avec un perceptron multicouche et un CNN pour faire de la classification audio, je peux conclure que les CNN sont beaucoup plus adaptés, du moins avec des capacités de calcul limitées, pour faire de l'analyse de signal audio. Les principes de convolution se jumèle très bien avec la représentation graphique des données audios, plus précisément des spectrogrammes et des MFCC.

Une architecture que j'explorais pour faire du traitement audio serait les réseaux LSTM (Long short-term memory) puisque ces réseaux permettent de garder un concept de progression au fil de l'analyse des données. Puisqu'un signal audio est généralement progressif au fil du temps, c'est-à-dire que le contexte est pertinent pour sa compréhension, ce type de réseau permettrait donc sûrement de faire des liens entre l'amplitude des fréquences au fil du temps comparé à ce qu'un MLP ou un CNN fait actuellement.

Références

D. Johnson. « Back Propagation in Neural Network: Machine Learning Algorithm », https://www.guru99.com/backpropogation-neural-network.html (consulté le 5 décembre 2022)

F. Richter. « The World's Favorite Music Genres », https://www.statista.com/chart/15763/most-popular-music-genres-worldwide/ (consulté le 5 décembre 2022)

G. Simmon. « Music Genres List », https://www.musicgenreslist.com/ (consulté le 5 décembre 2022)

J. Bronwlee. « A Gentle Introduction to Dropout for Regularizing Deep Neural Networks », https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/ (consulté le 5 décembre 2022)

MathWorks. « Help Center », https://www.mathworks.com/help/ (consulté le 5 décembre 2022)

MathWorks. « Pie chart », https://www.mathworks.com/help/matlab/ref/pie.html (consulté le 5 décembre 2022)

mlearnere. « Learning from Audio: Fourier Transformations », https://towardsdatascience.com/learning-from-audio-fourier-transformations-f000124675ee (consulté le 5 décembre 2022)

mlearnere. « Learning from Audio: Spectrograms », https://towardsdatascience.com/learning-from-audio-spectrograms-37df29dba98c (consulté le 5 décembre 2022)

mlearnere. « Learning from Audio: The Mel Scale, Mel Spectrograms, and Mel Frequency Cepstral Coefficients », https://towardsdatascience.com/learning-from-audio-the-mel-scale-mel-spectrograms-and-mel-frequency-cepstral-coefficients-f5752b6324a8 (consulté le 5 décembre 2022)

TensorFlow. « API Documentation », https://www.tensorflow.org/api_docs (consulté le 5 décembre 2022)

TensorFlow. « Reconnaissance audio simple : Reconnaître les mots-clés », https://www.tensorflow.org/tutorials/audio/simple_audio (consulté le 5 décembre 2022)

V. Velardo. « Audio Data Augmentation », https://youtube.com/playlist?list=PL-wATfeyAMNoR4aqS-Fv0GRmS6bx5RtTW (consulté le 5 décembre 2022)

V. Velardo. « Deep Learning (for Audio) with Python », https://youtube.com/playlist?list=PL-wATfeyAMNrtbkCNsLcpoAyBBRJZVlnf (consulté le 5 décembre 2022)

V. Velardo. « PyTorch for Audio + Music Processing », https://youtube.com/playlist?list=PL-wATfeyAMNoirN4idjev6aRu8ISZYVWm (consulté le 5 décembre 2022)

Wikipedia. « Confusion matrix », https://en.wikipedia.org/wiki/Confusion_matrix (consulté le 5 décembre 2022)

Wikipedia. « Backpropagation », https://en.wikipedia.org/wiki/Backpropagation (consulté le 5 décembre 2022)