DataScientest • com

Rapport Technique d'évaluation

Séparation de voix

Promotion DATA SCIENTEST formation continue juin 2021

Participants :

Ephi MAGLARAS, Mickaël RIVIER, Stany GALLIER

Contexte

Dans notre métier d'ingénieur dans le secteur industriel, nous disposons de beaucoup de données :

- numériques générées par des calculs ou des mesures expérimentales
- visuelles obtenues lors du suivi de fabrication de matériaux par exemple

Ces données sont souvent des sources d'informations sous-utilisées et qu'il est possible de mieux exploiter par la data science.

Ce projet nous permet de monter en compétences sur les techniques de data science pour :

- explorer nos données
- exploiter nos données via du machine learning ou deep learning
- prédire le comportement de nos systèmes et matériaux grâce à des modèles d'apprentissage de qualité

Le machine learning/deep learning représente un gain temporel, et donc économique, non négligeable, comme par exemple dans :

- classification des imagettes à la place de l'oeil humain
- conception plus rapide de nos produits (et a fortiori des coûts) grâce à la prédiction
- substitution d'un calcul élément fini coûteux

L'intelligence artificielle est aujourd'hui omniprésente, et en tant qu'ingénieurs dans le secteur industriel, nous avons besoin de connaître les sciences de la donnée afin de mieux concevoir nos produits, plus efficacement. Elle permet également d'extraire de nouvelles informations et/ou modèles à partir des données à disposition.

Objectifs

Le principal objectif de ce projet est de construire un modèle qui permet d'isoler et extraire les voix d'une musique en les séparant de l'accompagnement. La séparation de voix dans des pistes musicales peut avoir de nombreuses applications comme le karaoké, la remasterisation, le remixage mais aussi en télécommunications (débruitage) ou en reconnaissance vocale.

Le niveau d'expertise des membres du groupe sur cette problématique est variable :

- Ephi a une bonne connaissance de l'exploration de données numériques, aucune en traitement du signal et deep learning
- Mickaël a de légères bases mathématiques de traitement du signal, mais une bonne expérience en machine learning et, dans une bien moindre mesure, en deep learning
- Stany a des connaissances en acoustique et traitement du signal, quelques unes en machine learning mais aucune en deep learning

Pour affiner la problématique et les modèles sous-jacents, nous avons d'abord réalisé une étude bibliographique et testé les modèles existants dans la communauté scientifique. Nous n'avons pas eu d'interactions spécifiques avec des spécialistes du domaine.

Notre entreprise (ArianeGroup) mène quelques projets de deep et machine learning ; certains ont abouti à la suite du travail d'étudiants, d'autres sont en cours. Bien que la musique ou l'acoustique ne fassent pas partie du cœur de métier de notre entreprise, le projet de séparation de voix nous permet d'acquérir une bonne méthodologie applicable dans n'importe quel domaine scientifique et des connaissances solides notamment sur les CNN.

Data

Cadre

La base de données choisie pour débuter ce projet "séparation de voix" est la base MUSDB18 disponible librement [1]. Cette base contient 150 pistes de différents genres musicaux pour une durée approximative de 10 h. Chaque piste est également accompagnée de sa décomposition en percussions ("drums"), basse ("bass"), voix ("vocals") et le reste ("others"), qui inclut notamment tous les autres instruments. La somme "drums", "bass" et "others" constitue l'accompagnement (c'est-à-dire toute la musique une fois la voix séparée), voir Fig.1.

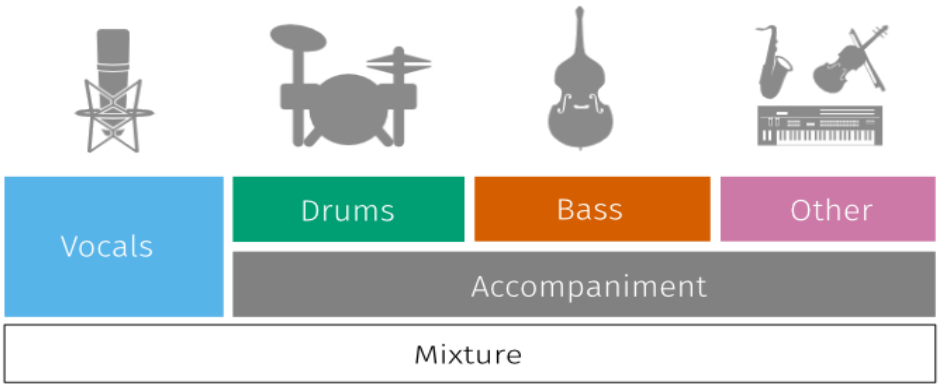

Figure 1. *Une chanson ("mixture") est décomposée en sa partie voix ("vocals") et accompagnement ("drums", "bass" et "others"). Ces quatre parties distinctes (appelées "stems") ainsi que leur somme ("mixture") sont disponibles dans la base MUSDB18.*

Les signaux sont stéréophoniques, encodés avec une fréquence de 44.1 kHz et compressés au format mp4, ce qui permet d'avoir une base plus légère (4 Go au lieu de 22 Go pour le format wav).

La base d'apprentissage comporte 100 chansons et la base de test 50 chansons (ratio train/test=67/33). La Fig.2 trace un histogramme des genres musicaux présents dans la base complète. Les différents genres sont représentés de manière peu uniforme avec une surreprésentation des styles pop/rock et rock au détriment de certains styles (jazz, country et reggae notamment).

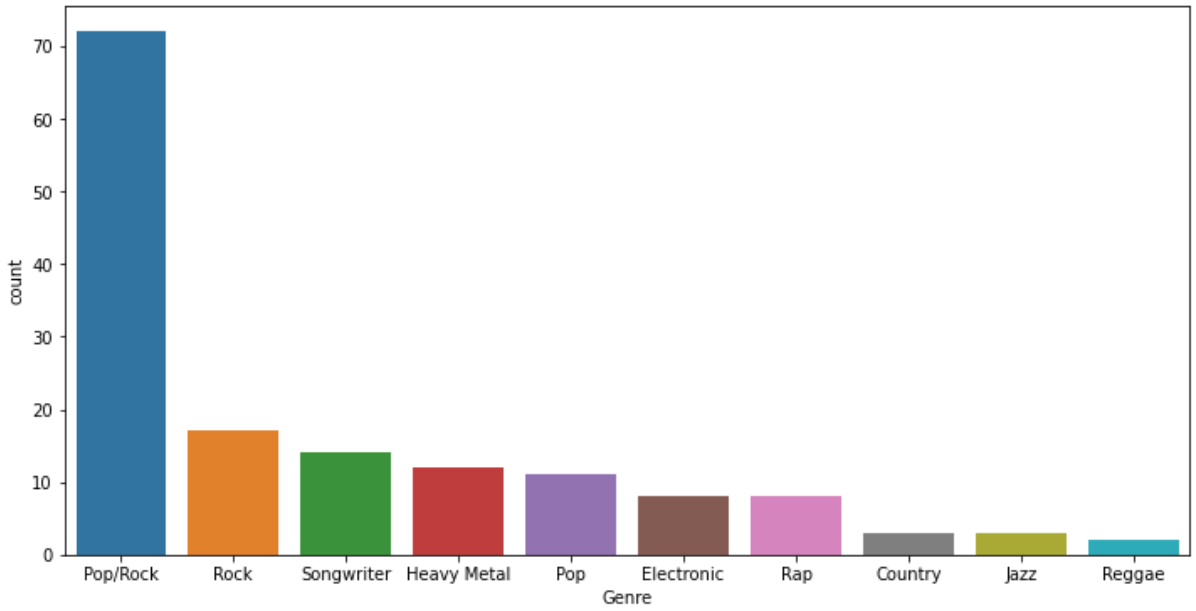

Figure 2. *Répartition par genres musicaux de la base MUSDB18*

On peut vérifier que cette répartition stylistique est similaire dans les sets d'entraînement et de test, pour maximiser la généralisabilité du modèle construit sur les données d'entraînement. Cette visualisation est donnée en Figure 3, avec le set d'entraînement représenté en bleu et celui de test en orange. Il semble que les données de test soient très majoritairement des musiques Pop/Rock, ou le ratio train/test atteint 50/50. On voit dans cette figure que tous les styles sont représentés dans la base d'entraînement, ce qui est un point positif. Cependant, cette omniprésence du style Pop/Rock dans la base de test fait que les métriques finales donneront principalement la capacité de l'algorithme à séparer la voix de l'accompagnement dans un contexte Pop/Rock.

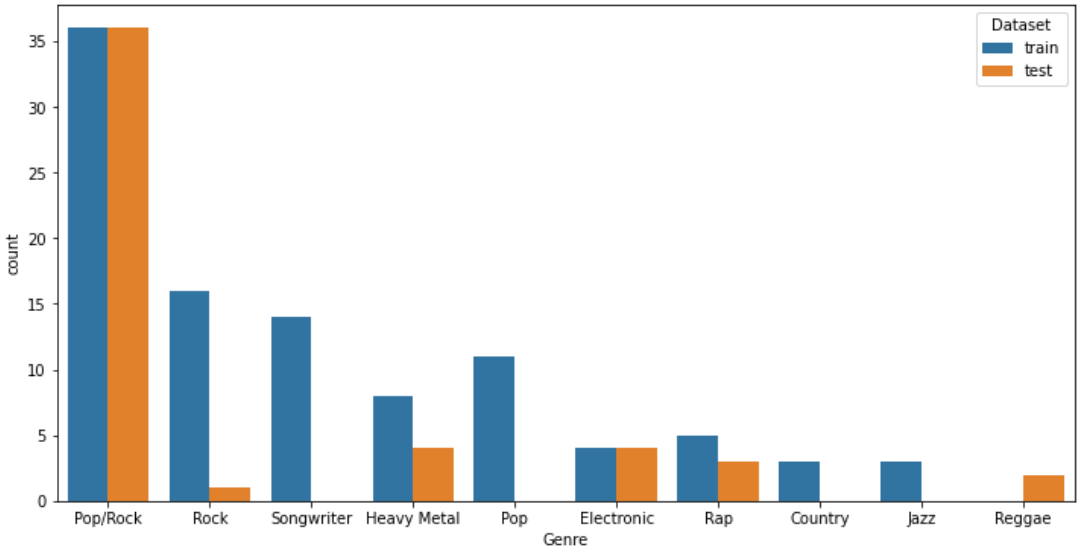

Figure 3. *Comparaison de la répartition des genres musicaux selon la base*

Il est aussi possible de visualiser la durée de ces différentes musiques. Un histogramme représentant la distribution de cette durée dans notre base complète, ainsi qu'une estimation KDE, est donné en Figure 4. On peut remarquer que les musiques ont une durée très variable, allant de seulement 12.8 secondes à 10 minutes et 28.3 secondes. La durée moyenne est de 3

minutes et 55.5 secondes, assez proche de la médiane (4 minutes) du fait de la forme relativement simple de la distribution. On remarque finalement le nombre très faible de musiques durant entre 1 et 2 minutes. On en déduit que les musiques présentent sont soit très courtes (moins d'une minute), soit d'une durée d'au moins 2 minutes.

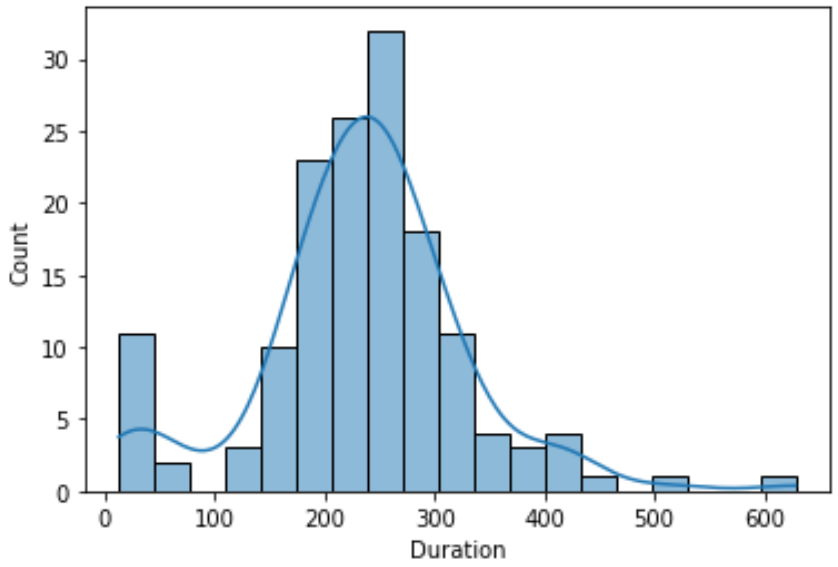

Figure 4. *Histogramme de la durée des musiques, en secondes*

A nouveau, on peut comparer les différences entre la base d'entraînement et la base de test. La Figure 5 superpose les histogrammes de la durée des musiques associées à ces deux bases. On peut remarquer que ces distributions semblent assez similaires, avec une différence de 20 secondes entre les moyennes et seulement 10 secondes entre les médianes. Cependant, il est important de noter que les cas les plus extrêmes (musiques très courtes ou très longues) sont tous inclus dans la base d'entraînement, évitant ainsi le risque de rater une musique "outlier", dont la durée étonnante pourrait aussi être signe d'un style peu banal.

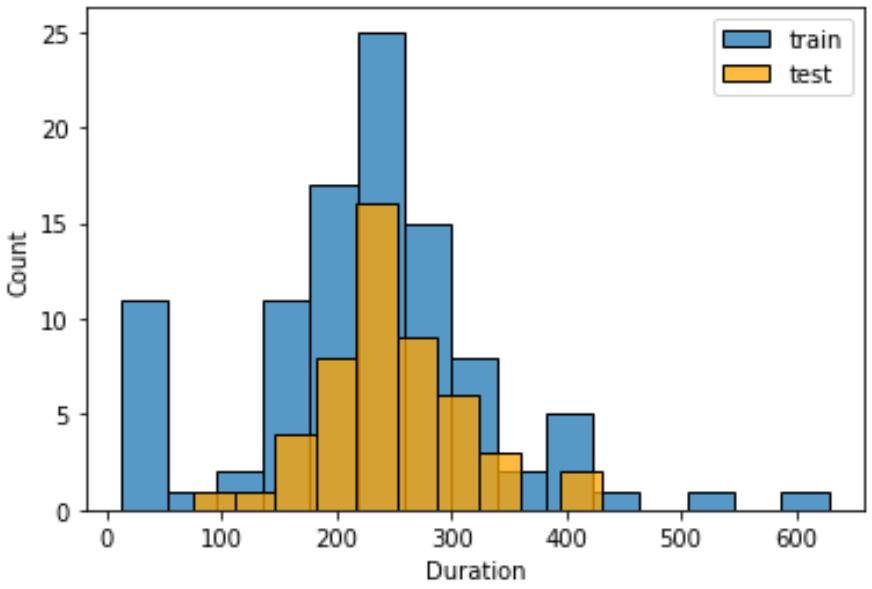

Figure 5. *Histogramme de la durée des musiques en fonction de la base*

Pertinence

Cette base est notamment utilisée lors des évaluations d'algorithmes de séparation de voix (comme le *Signal Separation Evaluation Campaign,* SiSEC). Les données semblent propres, sans valeurs manquantes. Seules quelques corrections ont dû être apportées dans le dataframe des musiques où quelques titres ne correspondaient pas entre le fichier .csv et les musiques.

Un premier prétraitement a consisté à ré-échantillonner toutes les chansons à une fréquence plus faible (8 kHz voire 4 kHz pour certains tests) que la fréquence d'échantillonnage initiale (44 kHz). Cela permet de limiter fortement la taille des données et le temps calcul. Cette fréquence est choisie de manière à pouvoir encoder les fréquences maximales de la voix tout en minimisant les pertes d'information. Pour les mêmes raisons de compression des données, les signaux stéréo sont passés en mono pour gagner un facteur deux en taille.

Nous verrons par la suite que la méthode choisie nécessite de quitter l'espace temporel pour travailler dans l'espace spectral. L'autre traitement des données brutes consiste donc en une transformée de Fourier glissante (STFT) sur les signaux temporels initiaux pour obtenir un spectrogramme et ce, grâce à la libraire `nussl [2]` elle-même basée sur `librosa [3]`. La base MUSDB contient les chansons ("mélange" ou "mixture") mais aussi les voix et accompagnements séparément : la Figure 6 présente ainsi un extrait de spectrogramme obtenu pour la chanson ("mélange"), mais aussi pour la voix seule ou l'accompagnement seul. L'idée est alors, à partir du spectrogramme du mélange (en haut) de déduire le spectrogramme de la voix (au centre). Notons que dans le cadre de notre projet, nous nous sommes essentiellement concentrés sur la voix et pas sur l'extraction de l'accompagnement, faute de temps. La figure montre clairement que la voix possède une signature spectrale spécifique, avec la présence d'une fréquence fondamentale (vers 200 Hz sur cet exemple), accompagnée d'harmoniques. Cette structure diffère assez de l'accompagnement (riche en fréquences et sans harmoniques visibles). L'idée générale de l'approche consiste ainsi à optimiser un masque permettant d'extraire les informations pertinentes pour passer du spectrogramme du mélange à celui de la voix seule.

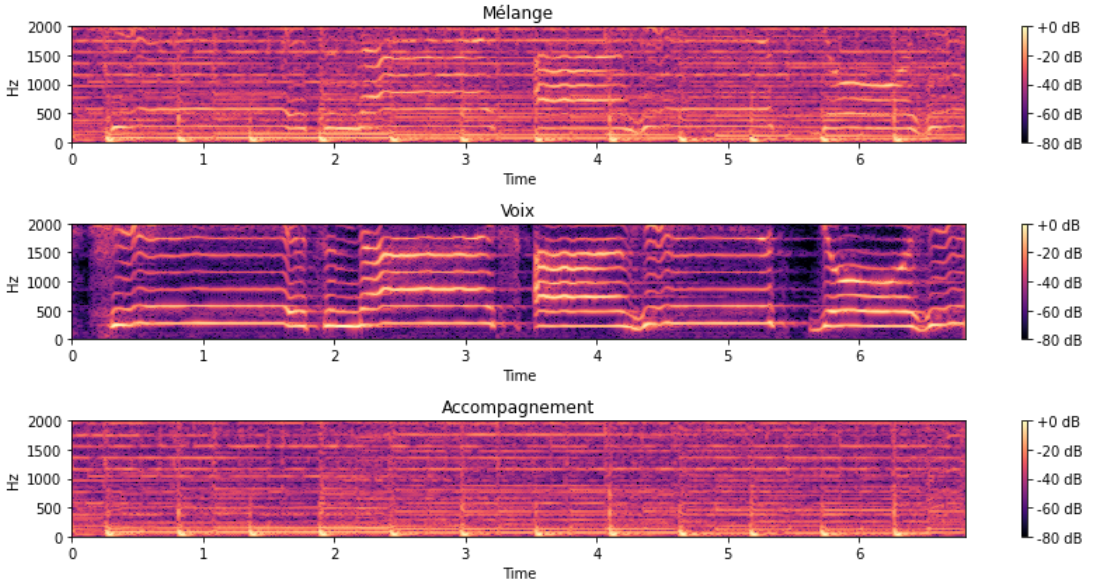

Figure 6. *Exemple de spectrogramme : mélange, voix et accompagnement d'une même chanson*

Les spectrogrammes seront conservés en échelle fréquentielle. Il existe une autre échelle, dite échelle Mel [5], qui est une transformation logarithmique afin de mieux rendre compte de la perception humaine. Des visualisations en échelle Mel ont été réalisées mais l'apprentissage s'est fait uniquement en fréquence, faute de temps.

Au vu du modèle qui sera choisi par la suite, les données d'entrée sont des portions de spectrogramme et s'apparentent donc à des images, comme pour un CNN classique. Les variables explicatives sont donc les "pixels" de ces images, donc la norme du spectre à un temps et une fréquence donnés. La variable cible est le spectrogramme de la voix seule, extraite du spectrogramme de la chanson ("mixture"). Une spécificité est donc que la sortie, comme l'entrée, est sous la forme de matrice 2x2 (image).

Une analyse rapide des données d'entrée a été effectuée en visualisant les spectrogrammes pour un certain nombre de chansons. Il existe une grande diversité de données, notamment du fait que certains morceaux contiennent peu de chant ou encore plusieurs voix superposées (chœurs).

Projet

Classification du problème

Dans la mesure où l'on recherche un spectrogramme (et donc des valeurs), il s'agit d'un problème de régression supervisée. Pour autant, il s'apparente à des problématiques d'analyse d'image et de reconnaissance faciale puisqu'il se base sur une analyse d'images (des spectrogrammes). Pour autant, l'analyse d'image est souvent un problème de classification à la différence de notre cas.

Une première évaluation très qualitative des performances consiste à écouter le résultat (voix seule reconstruite) par rapport à la vérité terrain (voix seule, issue de la base). De la même manière, une comparaison visuelle des spectrogrammes (vrais et reconstruits) fournit déjà de premiers éléments. L'écoute se fait grâce à la classe `Audio` dans `IPython.display` [6]. Une barre de lecture est alors disponible pour écouter l'extrait.

De manière plus quantitative, nous utilisons une métrique, assez classique dans le domaine, appelée SI-SDR (Scale-Invariant Source-to-Distortion Ratio [12]), définie par $SDR=10\log(|S_{true}|^2/|S_{pred}-S_{true}|^2)$ avec S le signal temporel. L'aspect "Scale-Invariant" est assuré par une normalisation (moyenne nulle) des données. Le calcul est réalisé via la librairie `BSS`. Sur la base test MUSDB (50 chansons), le SI-SDR prédit pour la voix s'échelonne entre 3,25 et 9 dB selon les modèles [13] mais avec de fortes disparités selon les chansons.

Un travail important a été de considérer et d'évaluer des modèles disponibles de la littérature [7-10], notamment des modèles réputés pour leur performance comme Spleeter ou Demucs. Ceci permettra de situer notre approche par rapport à l'état de l'art.

Choix du modèle & Optimisation

Choix du modèle

Le modèle s'appuie sur un réseau complètement convolutionnel de type U-Net, initialement développé pour la segmentation d'images dans le domaine biomédical. Notre réseau s'inspire fortement des travaux de Jansson *et al.* [14] qui proposent une application du U-Net à la séparation de voix en travaillant sur le spectrogramme.
Le modèle initialement codé est tiré de cette référence et sa structure est schématisée sur la Fig. 7. Le réseau se compose d'une partie contractante et une partie expansive, ce qui lui confère une architecture en forme de U. La partie contractante est un réseau de convolution typique qui consiste en une application répétée de convolutions (*kernel* 5x5, *strides* 2x2), chacune suivie d'une unité linéaire rectifiée de type *LeakyReLU*(0.2) avec *BatchNormalization*. Pendant la contraction, les informations spatiales sont réduites tandis que les informations sur les caractéristiques sont augmentées : une image de spectrogramme de taille 512 x 128 est ainsi réduite à une taille 8x2 mais sur 512 filtres La voie expansive combine une séquence de déconvolutions (*Deconv2D*) avec concaténations des informations issues de la voie contractante. Dans cette voie, l'activation est un *ReLU* avec dropout à 50 % pour les trois premières couches. Dans la couche finale, une activation sigmoïdale permet d'obtenir un masque M, soit une image 512 x 128 (même taille que l'originale) avec des valeurs comprises entre 0 et 1. La sortie finale Y_{pred} du réseau est la multiplication de ce masque M avec l'image d'entrée X, soit $Y_{pred}=M \cdot X$ où $\cdot$ désigne le produit d'Hadamard (produit terme à terme). Le spectrogramme prédit Y_{pred} est donc censé encoder la voix seule.

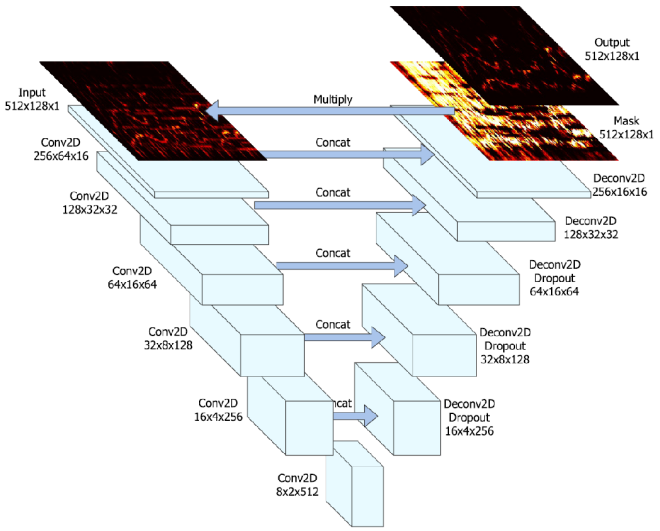

Figure 7 *: Structure du réseau U-Net codé [14]*

L'entraînement se fait en minimisant une perte calculée entre l'entrée X (spectrogramme du mélange) et le spectrogramme de la voix Y_{true} en norme L_1 : on minimise donc $|M \cdot X - Y_{true}|$. Un optimiseur ADAM est choisi ainsi que des batchs de taille modeste (par ex., un batch de 32 s'est révélé un bon compromis) ; on intègre également un *early stopping* et une réduction de learning rate sur plateau en *callbacks*. L'apprentissage est poursuivi jusqu'à stagnation de la perte de validation (*early stopping*) : on note alors dans ce

cas que la perte d'apprentissage pourrait encore descendre. Il s'agit encore une fois d'un signe d'overfitting lié à un nombre de données pas assez important.

Nous avions initialement essayé d'optimiser directement sur le spectrogramme sans l'aide d'un masque, soit une perte $L=|X - Y_{true}|$ mais le réseau tend généralement à stagner assez vite avec une sortie souvent proche de 0.

Toute la phase de calcul est réalisée sous Google Colab (version gratuite). Dans la mesure du possible, l'accès aux GPU est privilégié (gain d'un facteur 10 à 20 en temps calcul). Cet accès est toutefois quelquefois aléatoire dans la version gratuite...

La sortie du réseau est un spectrogramme, en norme donc. Pour remonter au signal temporel (pour une écoute), il est nécessaire de remultiplier cette norme par une phase (en l'occurrence la phase du mélange) puis de refaire une transformée de Fourier inverse.

Data generation

Une des problématiques est la taille de la base qu'il est impossible de charger entièrement en RAM sur la version gratuite de Google Colab. Pour contourner ce problème, un générateur python est utilisé pour les datasets d'entraînement et de validation. Ce générateur est grandement personnalisable, que ce soit au niveau des paramètres du traitement audio ou du réseau de neurone. Nous nous laissons notamment la possibilité de fixer le batch size et de randomiser les batchs.

Un problème de taille subsiste alors : la récupération des données depuis la base musdb nécessite la lecture de l'audio et la construction du spectrogramme, ce qui représente un temps de calcul significatif. En intervenant à tous les batchs de toutes les epochs, il s'agit alors de la cause principale de la lenteur d'entraînement du réseau. Pour pallier ce problème, nous proposons la possibilité de mettre les spectrogrammes construits en cache sur le disque après leur première lecture, de sorte que dès la deuxième epoch, la donnée en cache soit lue (quasi instantanément) au lieu d'être reconstruite depuis l'audio. Dans les faits, cela permet de passer de 30 minutes à une vingtaine de secondes par epoch (sur GPU) à partir de la deuxième.

Enfin, nous donnons la possibilité de paralléliser les tâches de construction et de lecture de ces spectrogrammes. Cependant, les gains ne sont pas très importants sur ce volet. Un multithreading a tout d'abord été testé pour la tâche de lecture des données en cache (qui est IO-bound), avec un speed-up à peine supérieur à un et des risques de surcharger la RAM en chargeant les spectrogrammes de plusieurs musiques simultanément sur la même unité de calcul. Nous avons alors envisagé d'utiliser du multiprocessing pour la tâche de construction des spectrogrammes depuis les données audios (CPU-bound). Cependant, la version gratuite de Colab ne permet d'utiliser que deux unités de calcul, avec un temps de communication non-négligeable, et un risque persistant de surcharger la RAM, partagée entre ces CPU. Finalement, la parallélisation ne s'est pas avérée payante et nous l'utilisons en pratique avec l'argument `max_workers` à 1 dans les notebooks finaux. Ces développements ont été réalisés grâce à la librairie `concurrent.futures`.

Optimisations diverses

Dans le papier initial de Jansson, les signaux sont sous-échantillonnés à 8 kHz et le spectrogramme est calculé sur une taille de fenêtre de 1024 (donnant donc lieu à 512 bins en fréquence). Les extraits (patchs) de spectrogramme sont de taille 128, qui correspondent environ à 11 s de temps. Les premières tentatives ont montré un comportement correct du réseau dans l'extraction de voix mais avec des temps d'apprentissage longs et des résultats moyens, avec souvent un overfitting.

Nous avons pu proposer un certain nombre d'optimisations dont :

1. Accélérer l'apprentissage en ajustant les paramètres de la STFT
2. Augmenter le nombre de données d'entraînement par data augmentation
3. Ajuster la complexité du réseau à la quantité de données pour limiter l'overfitting

- *Paramètres de la STFT*

  Les premiers résultats, ainsi que l'analyse des spectrogrammes, montrent que la voix reste dans des fréquences inférieures à 1~2 kHz. Nous avons ainsi choisi de revoir la fréquence d'échantillonnage à 4 kHz, ce qui permet d'encoder des signaux jusqu'à 2 kHz (critère de Nyquist) en étant plus léger en termes de données. Nous pouvons ainsi choisir une fenêtre de 512 (*hop_length*=384) et ainsi aboutir à des images 256 x 64 avec la même résolution temporelle et fréquentielle.

- *Data augmentation*

  La base d'apprentissage MUSDB utilisée contient 100 chansons (environ 7 h de musique, soit approximativement 2300 images de spectrogrammes avec les paramètres initiaux). Avec le même réseau, Jansson *et al.* utilisent plus de 20,000 chansons, soit 200 fois plus de données que nous. Il n'est pas étonnant que les performances diffèrent. Ce manque de données conduit à un sur-apprentissage avec le modèle initial : il est donc nécessaire d'agir sur le nombre de données mais aussi sur le réseau. Pour la data augmentation, nous proposons une approche simple qui s'apparente à celle utilisée classiquement pour les images. Au lieu de découper le spectrogramme complet concaténé en un nombre disjoint de spectrogrammes, nous tirons aléatoirement des spectrogrammes de taille voulue au sein de notre base. Il n'y a pas de nouvelles informations mais les « images » de spectrogramme sont aléatoirement décalées, de manière analogue au « shift » d'un *ImageGenerator*.

- *Ajustement du réseau*

  Des signes de sur-apprentissage peuvent apparaître avec une saturation de la perte de validation après quelques époques. Dans nos premiers essais, la base était de l'ordre de 100 chansons et le réseau possédait 9,8 millions de paramètres. Pour le même réseau, Jansson et al. utilisent plus de 20,000 chansons. Nous avons ainsi tenté de trouver un compromis entre nombre de données et gestion du sur-apprentissage. A réseau fixé, on note clairement une diminution de la perte de validation lorsque le nombre de données augmente, comme l'illustre la Fig. 8. Différents réseaux ont été testés en jouant sur le noyau de convolution (passage en 3x3 à la place du 5x5 sur les plus basses couches), le nombre de filtres, ou la structure même avec un U moins profond. Le nombre de paramètres a été varié entre 0,2 et 9,8 millions. Les structures avec le moins de paramètres montrent quant à elles des signes de sous-apprentissage lorsque le nombre d'images est important.

Un bon compromis est de conserver le U-Net initial avec un noyau 5x5 seulement sur la première convolution (et 3x3 pour les autres) et avec 256 filtres en bas de U (contre 512 initialement), ce qui conduit à un modèle à 2,3 millions de paramètres. Nous avons également “joué” avec certains autres paramètres comme le niveau de dropout, sans effets notables.

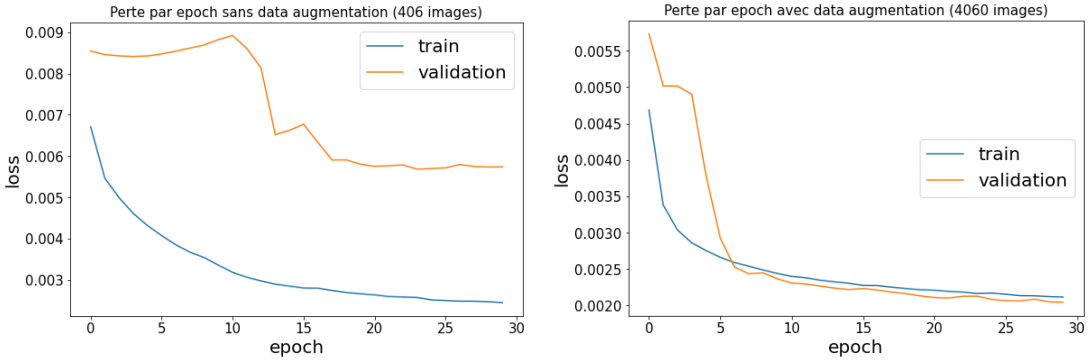

Fig. 8 : Perte d'entraînement (bleu) et de validation (orange) sans data augmentation (406 images, gauche) et avec (4060 images, droite). Le nombre de données diminue clairement le sur-apprentissage.

- *Choix de la loss*
  Enfin, quelques essais ont été effectués concernant le choix de la fonction de perte. Les loss classiques (L_1 ,L_1 relatif, L_2) ont été testées, et les meilleurs résultats sont généralement obtenus avec la perte de type L_1 (mean absolute error), qui était celle conseillée dans l’article de Jansson.
  Une fonction de perte personnalisée a aussi été testée, avec de bons résultats, notamment en l’absence d’augmentation de données pour éviter de tomber des des minima locaux renvoyant des spectrogrammes entièrement vides. Cette fonction de perte vise ainsi à pénaliser les spectrogramme trop vides (et trop pleins), et se définit comme suit : $L = \sum_{i,j}\left|Y_{ij} - \widehat{Y_{ij}}\right| + \left|\sum_{i,j} Y_{ij} - \sum_{i,j}\widehat{Y_{ij}}\right|$, avec Y le spectrogramme 2D réel et $\widehat{Y}$ la prédiction. Cette loss correspond donc à la somme des écarts plus l’écart des sommes, ce qui pénalise particulièrement un spectrogramme vide.

Versions du U-Net

Le temps a manqué pour terminer une version finale incluant à la fois la *data generation* et la *data augmentation.* Notre modèle existe ainsi en deux versions.

La première tire profit de la *data generation :* l’apprentissage se fait sur l’intégralité de la base mais elle n’inclut pas d’augmentation de données. En l’état, elle a comme désavantage insidieux qu’un batch d'entraînement ne contient des données que de quelques musiques différentes, au lieu d’avoir des tronçons répartis sur l’entièreté de la base de donnée. Cela tend en effet à fortement bruiter l’estimation du gradient à chaque batch, rendant la descente de gradient beaucoup plus erratique, avec une loss de validation faisant parfois de grands sauts dans la mauvaise direction, même avec un faible learning rate. Dans ce contexte, un callback `EarlyStopping` avec restauration des meilleur poids était crucial pour récupérer des résultats relativement performants. Cette version fait aussi usage de la fonction de perte personnalisée permettant de réduire le risque de se piéger dans un minimum local.

Dans la seconde version, l'absence de générateur empêche de considérer l'intégralité de la base en conservant l'échantillonnage à 8 kHz. Il est alors proposé de charger l'ensemble des données en RAM en passant à 4 kHz, et de mettre en œuvre l'augmentation de données. Cette augmentation, couplée à une estimation moins biaisée des gradients apporte une descente beaucoup plus monotone, comme on peut le voir en Figure 8. La fonction de perte classique (L_1) est ici suffisante pour obtenir une bonne convergence. Les résultats de ces modèles semblent très légèrement supérieurs à ceux de la première version, bien qu'une certaine perte de qualité audio se fasse entendre à cause du sous-échantillonnage.

Résultats

Parmi les nombreux tests effectués, les meilleurs réseaux (par ex., en termes de perte de validation) sont stockés. Ils sont par la suite appliqués à l'ensemble de la base de test (50 chansons) pour une prédiction de la performance (métrique SI-SDR). Les résultats sont présentés sous forme de boxen-plot. Comme spécifié plus haut, un travail a également consisté à évaluer cette métrique sur la base de test sur un grand nombre de méthodes de la littérature, des plus simplistes (filtrage) aux plus avancées.

Plus précisément, les méthodes auxquelles nous nous comparerons sont divisées en trois catégories :

- Les méthodes de référence benchmark (plancher et plafond). La méthode *High-Low Pass* est une méthode extrêmement naïve, qui classifie toutes les basses fréquences (<1 kHz) en accompagnement et toutes les hautes fréquences en voix. Cela devrait constituer un score plancher en dessous duquel une méthode sera considérée comme très mauvaise. Les méthodes *Ideal Binary Mask* et *Ideal Ratio Mask* sont des méthodes oracles (connaissant la vérité) avec une classification binaire ou continue de chaque "pixel" du spectrogramme concernant l'appartenance à la voix ou l'accompagnement.
- Les méthodes primitives correspondent aux approches mathématiques de séparation de sources audio, qui étaient à l'état de l'art avant l'arrivée du Deep Learning (jusqu'à 2012 environ). Le *TimbreClustering* réalise un clustering des Mel Frequency Cepstral Coefficents (MFCC) des spectrogrammes factorisés par matrices non négatives (NMF). La méthode *FT2D* calcule la transformée de Fourier 2D du spectrogramme et classifie les pics (patterns qui se répètent) à l'accompagnement et le reste à la voix. La méthode *HPSS* (harmonic/percussive source separation) implémente la méthode proposée de [15]. Enfin, les méthodes *Repet* et *RepetSim*, pour REpeating Pattern Extraction Technique, recherche des notions de périodicité ou de similarité, respectivement, pour extraire l'accompagnement. Elles ont été proposées dans les travaux [16] et [17].
- Les méthodes par Deep Learning, beaucoup plus générales et capables de détecter des patterns très complexes, qui n'ont pas à être explicitement décrits par un humain. On retrouve des variantes de *Demucs* [8], développé par Facebook, vainqueur du dernier concours de séparation de sources proposé par Sony. Il s'agit d'une méthode hybride, travaillant simultanément sur le spectrogramme et le temporel audio, basée sur une architecture UNet. La méthode *Spleeter* [7] développée par Deezer, consiste en un réseau UNet plus simple, entraîné sur une base différente de musdb pendant environ une semaine sur GPU. Enfin, *Open-Unmix*, développé dans le cadre académique, notamment par Inria, suit une

architecture très différente. Celle-ci est basée sur un réseau récurrent LSTM bidirectionnel avec une couche de réduction de dimension permettant un embedding plus performant que le spectrogramme classique obtenu par STFT.

Notre objectif serait ici d'arriver à obtenir a minima des scores supérieurs aux méthodes primitives, qui étaient à l'état de l'art il y a une dizaine d'années. Nous doutons fortement de pouvoir atteindre les résultats de l'état de l'art actuel, bien que plusieurs des méthodes de la littérature (notamment *Spleeter*) aient des approches proches de la nôtre. Les différences résiduelles seront principalement dues à la faible quantité de données d'entraînement et au manque de temps pour peaufiner le choix de notre architecture et de nos paramètres d'optimisation.

Les scores SI-SDR de toutes ces méthodes sur les 50 musiques de la base de test sont tracés ci-dessous sous forme de boxen-plot en Figure 9, avec 7 des réseaux que nous avons construits. Les différentes méthodes sont classées par score moyen croissant. Nous retrouvons sans surprise *High-Low Pass* en dernière position et les *Ideal Mask* en tête. Viennent ensuite les *Demucs EXTRA*, entraînés sur de grosses bases des données supplémentaires, puis les *Demucs* classiques, entraînés sur musdb uniquement. Suivent ensuite *Spleeter* et *Open-Unmix*, prouvant que les approches récentes par Deep Learning dominent effectivement le domaine de la séparation de sources audios.

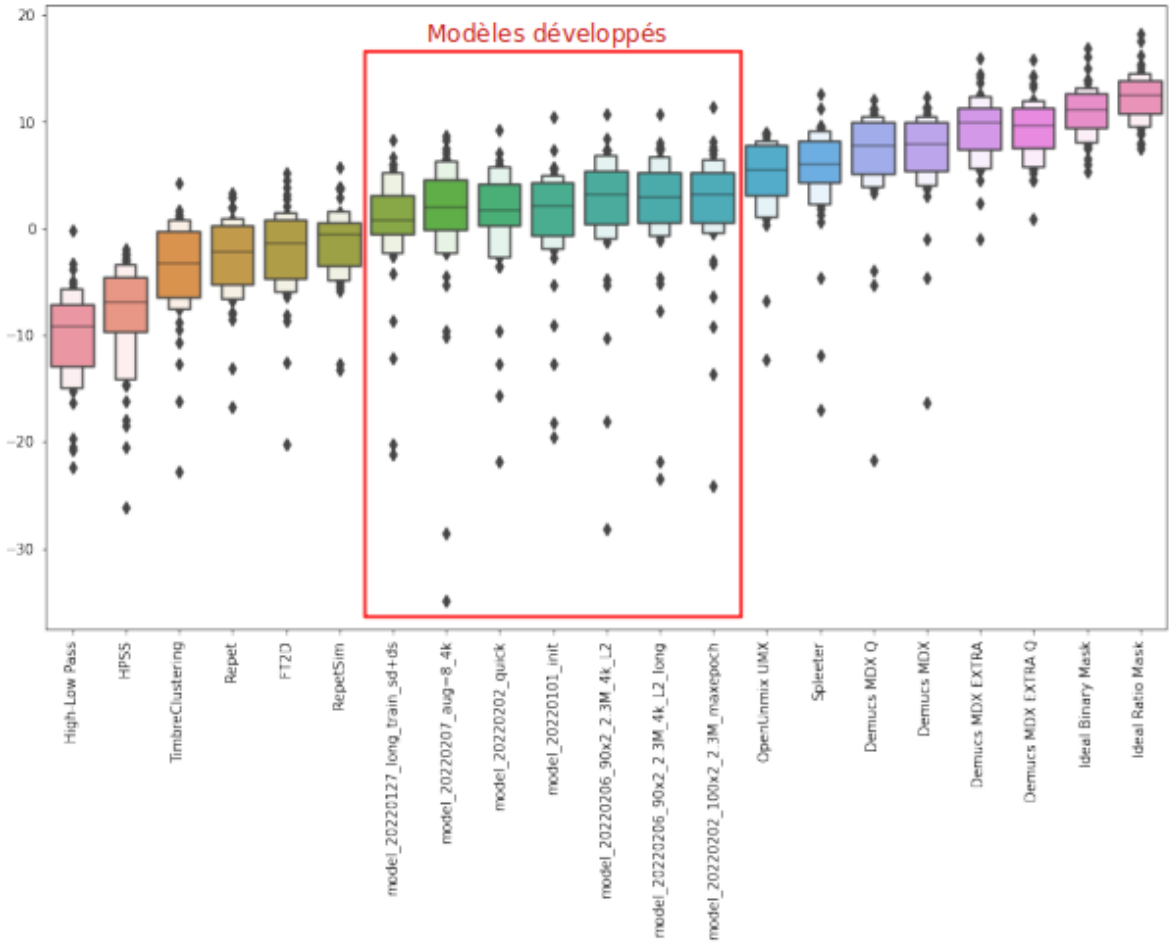

Figure 9 *: Scores SI-SDR de toutes les méthodes testées, classés par moyenne croissante*

Les sept méthodes suivantes en termes de performance sont nos UNet, avec une tendance très légèrement favorable pour les réseaux construits sans générateurs et avec augmentation de données (*100x2_2.3M_maxepoch, 90x2_2.3M_4k_L2_long, 90x2_2.3M_4k_L2, aug=8_4k*) face aux réseaux avec générateur et sans augmentation de données (*init, quick, long_train_sd+ds*). Ces sept méthodes ne diffèrent au final que par des variations de paramètres (nombre d'epochs, type de loss, paramètres de data augmentation). Finalement, les cinq méthodes primitives présentent les scores les plus faibles.

Il peut être plus simple de représenter ces scores sur une échelle normalisée, comme en Figure 10. Cela permet de positionner chaque méthode de 0 (pire score) à 1 (meilleur score) pour chaque musique, et de tracer les boxen-plot regroupant les 50 musiques de test. On remarque ici que l'*Ideal Ratio Mask* donne le meilleur score de manière parfaitement consistante. Les méthodes *Demucs* sont en moyenne légèrement au-dessus de 0.8, *Spleeter* et *Open-Unmix* autour de 0.7, nos UNet à quasiment 0.6 et les méthodes primitives à 0.4 et en-dessous.

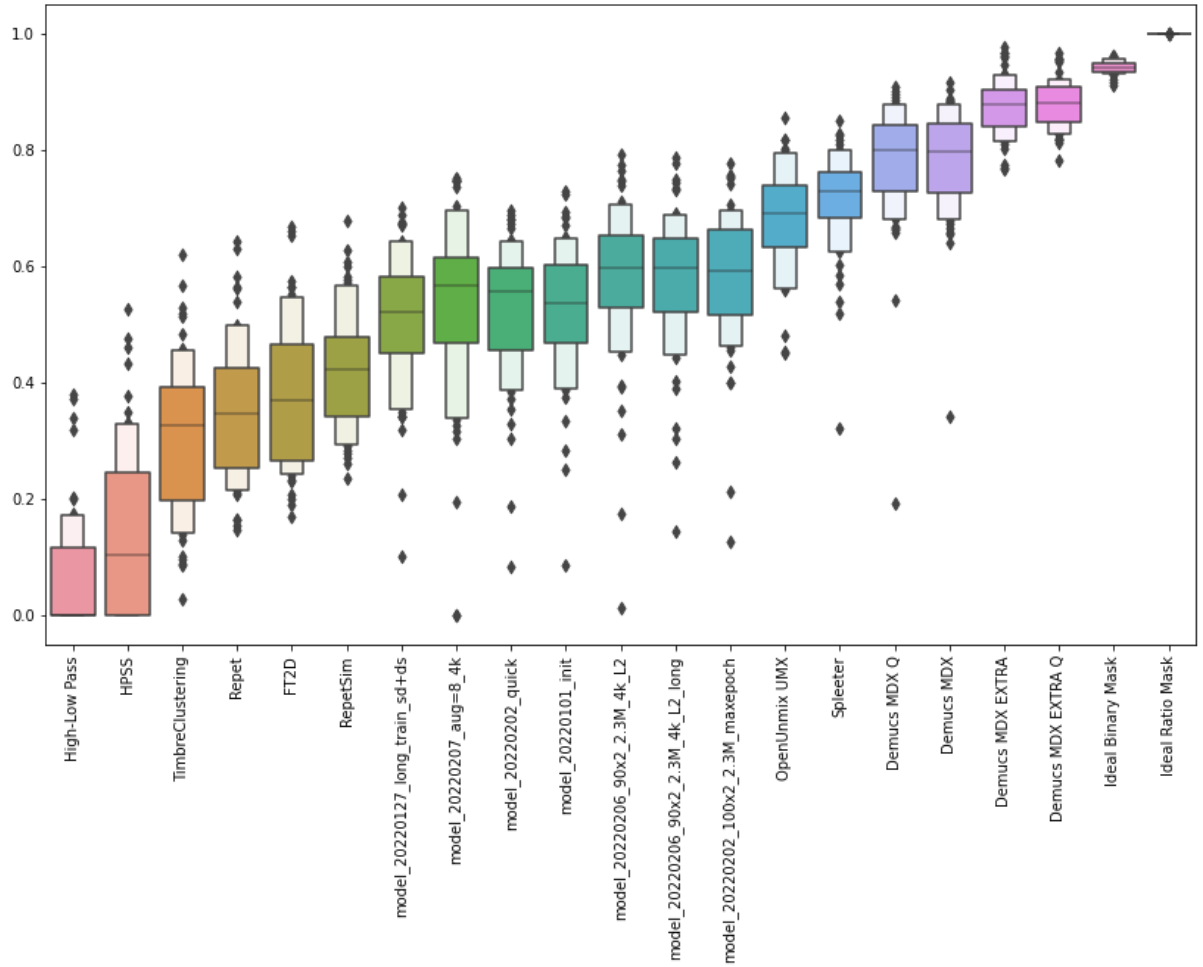

Figure 10 *: Scores SI-SDR normalisés, classés par moyenne croissante*

Pour finir l'analyse des résultats, nous pouvons visualiser un exemple d'extraction de voix par nos modèles. Les Figures 11, 12 et 13 représentent respectivement les spectrogrammes de la musique complète (mélange voix et accompagnement), de la voix seule réelle et de la voix extraite par un de nos modèles UNet. Bien que la prédiction ne soit pas parfaite, on voit que la majorité de l'accompagnement a été éliminé et que les zones de forte énergie (en jaune) du spectrogramme de voix sont assez bien captées.

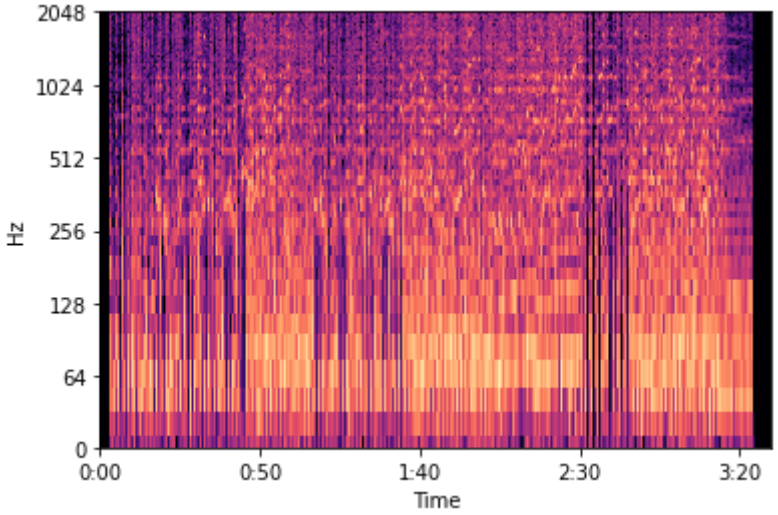

Figure 11 *: Spectrogramme du mix (voix et accompagnement) d'une musique de test*

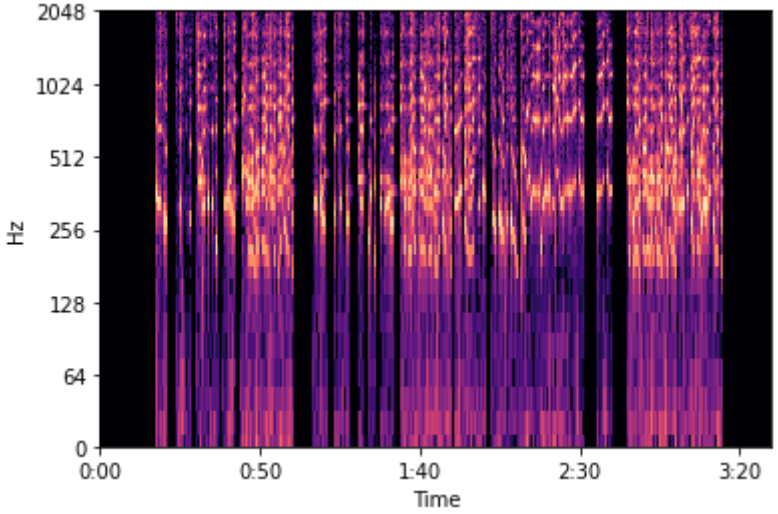

Figure 12 *: Spectrogramme de la voix seule réelle*

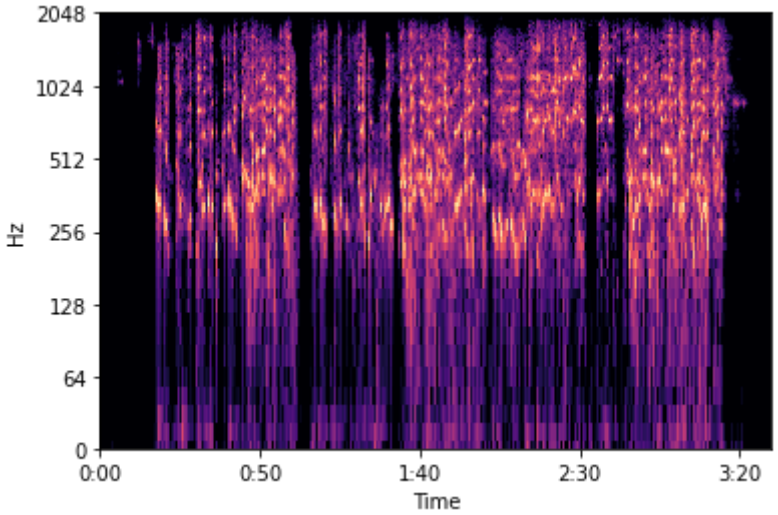

Figure 13 *: Spectrogramme de la voix extraite, prédit par l'un de nos UNets*

Discussion

Faute de temps, le seul modèle construit est le U-net présenté, même s'il existe de nombreuses autres approches décrites dans la littérature, notamment dans l'espace temporel. Ces autres approches, disponibles dans la littérature, ont toutefois été utilisées pour se comparer à nos résultats.

Une analyse du SI-SDR sur la base test montre une très grande variabilité selon les chansons. Le modèle s'est par exemple avéré médiocre sur les chansons contenant très peu de voix (visibles à travers les outliers à SDR fortement négatifs sur la Fig.9). Une tentative d'amélioration des performances du modèle a consisté à modifier le réseau pour prédire simultanément deux informations : le spectrogramme voix (comme auparavant) mais aussi la présence de voix (un réseau de neurones d'une seule couche est alors "branché" sur l'espace latent, en fond de U) pour en déduire une probabilité d'occurrence de la voix. Le spectrogramme voix prédit par la première branche du réseau est alors multiplié par la probabilité d'occurrence de la voix. Ainsi, en absence de voix, le modèle prédit un spectrogramme de voix nul par construction. Malheureusement, les résultats se sont trouvés dégradés notamment parce que la précision de prédiction du caractère voix/absence de voix est assez faible (de l'ordre de 75 % sur la base de test). Ici encore, il est supposé qu'un plus grand nombre de données aurait pu améliorer les résultats.

En conclusion, nous avons pu au travers de ce projet mettre en œuvre un réseau CNN en incluant certaines optimisations (data augmentation et data generation). Les ajustements sur les paramètres du CNN n'ont pas permis d'améliorer les résultats, principalement limités par la petite taille de la base. Nos résultats ont pu être comparés à l'existant.

Description des travaux réalisés

Répartition de l'effort sur la durée et dans l'équipe

1. Ouverture d'un dossier de travail commun sur le drive → Mickaël
2. Mise en place de l'accès à ce dossier depuis chaque drive personnel → tous
3. Récupération et stockage des données sur le drive→ Mickaël
4. Exploration du csv et correction pour cohérence avec les données→ Mickaël et Ephi
5. Lecture bibliographique→ tous
6. Mise en place des bibliothèques de traitement du signal et de calcul de métriques → Mickaël
7. Visualisation des données et traitement du signal→ tous
8. Rédaction du premier rapport dataviz→ tous
9. Création du github (mise en place, architecture, intégration des notebooks)→ Ephi
10. Intégration des modèles existant de séparation (pour benchmark)→ Mickaël
11. Développement de modèles ML/DL pour la prédiction de l'existence de voix → Stany
12. Développement du U-Net→ Stany
13. Travaux sur la data (data generator, data augmentation)→ Mickaël et Stany
14. Tests, utilisations et optimisations du U-Net→ Mickaël et Stany

15. Applications du U-Net, calcul de métriques et comparaison aux modèles existants→ Mickaël
16. Rédaction du rapport→ tous

Bibliographie

1. MUSDB18 database. https://sigsep.github.io/datasets/musdb.html
2. Nussl library. https://github.com/nussl/nussl
3. Librosa library. https://librosa.org/
4. Musdb library. https://github.com/sigsep/sigsep-mus-db
5. Getting to know the Mel spectrogram, *Towards Data Science* https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0
6. IPython Audio display. https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html
7. Spleeter library. https://github.com/deezer/spleeter
8. Demucs library. https://github.com/facebookresearch/demucs
9. Open-Unmix library. https://github.com/sigsep/open-unmix-pytorch
10. Conv-TasNet library. https://github.com/naplab/Conv-TasNet
11. MUSDB18 benchmark. https://paperswithcode.com/sota/music-source-separation-on-musdb18?p=open-unmix-a-reference-implementation-for
12. Emmanuel Vincent, Rémi Gribonval, and Cédric Févotte. Performance measurement in blind audio source separation. *IEEE transactions on audio, speech, and language processing*, 14(4):1462–1469, 2006.
13. https://paperswithcode.com/sota/music-source-separation-on-musdb18
14. Jansson, A., Humphrey, E., Montecchio, N., Bittner, R., Kumar, A., & Weyde, T. (2017). Singing voice separation with deep u-net convolutional networks. 18th International Society for Music Information Retrieval Conference
15. Fitzgerald, Derry. “Harmonic/percussive separation using median filtering.”
16. Rafii, Zafar, and Bryan Pardo.“Repeating pattern extraction technique (REPET): A simple method for music/voice separation.” IEEE transactions on audio, speech, and language processing 21.1 (2012): 73-84.
17. Zafar Rafii and Bryan Pardo.“Music/Voice Separation using the Similarity Matrix,” 13th International Society on Music Information Retrieval, Porto, Portugal, October 8-12, 2012.

Difficultés rencontrées lors du projet

Les principaux verrous techniques ont été les suivants :

- Choix d’un modèle pertinent (mais suffisamment simple à mettre en oeuvre) parmi le nombre important d’approches existantes
- Gestion d’une base de données importante : problématique de RAM, récupération et stockage des bases, etc.
- Faire fonctionner et optimiser un CNN à l'architecture complexe

Certaines autres difficultés ont pu retarder le développement de ce projet.
La faible volumétrie des données pour l'apprentissage des modèles (100 chansons) ne nous a pas permis d'obtenir des modèles de grande qualité contrairement à ceux, disponibles, que nous avons pu tester : ces derniers ont été entraînés sur des dizaines milliers de musiques. Cette base est toutefois suffisamment grande pour générer des problématiques de stockage mais aussi de saturation de la RAM. Un temps non négligeable a été consacré à contourner le problème notamment avec un DataGenerator.

Le temps de calcul a également ralenti et limité nos tests. Le chargement de la base ainsi que son traitement (spectrogrammes) prend de l'ordre de 20 à 30 minutes. Les temps d'apprentissage sont assez longs, typiquement plusieurs heures sur CPU. Ce temps peut être réduit à quelques dizaines de minutes avec des GPU. Mais l'accès aux GPU sur Google Colab s'est avéré quelquefois aléatoire (déconnection en cours de calcul, blocage à l'accès des GPU pendant une journée si trop d'utilisation,...). Une déconnexion inopinée de Colab nécessite de reprendre la procédure à zéro, notamment la demi-heure de chargement et de pré-traitement.

D'autre part, le projet débute très tôt dans la formation alors que la plupart des modules de machine learning n'ont pas encore été abordés, et ceux de deep learning pas du tout puisqu'ils arrivent à la fin du projet. Nous avons d'ailleurs, dans l'attente de la formation deep learning, mis en œuvre des approches de ML plus classiques (régression logistique, SVM, …) pour prédire l'existence ou non de voix en fonction du temps pour une chanson donnée.

D'autre part, le projet est axé sur l'audio et il a fallu un temps préliminaire d'acquisition des concepts et des bibliothèques spécifiques (traitement du signal, écoute audio, …).

Bilan & Suite du projet

L'objectif principal du projet a été atteint, à savoir la mise en place d'une approche de deep-learning pour la séparation de voix (réseau convolutionnel). Les résultats sont encourageants mais limités par la base d'apprentissage relativement petite malgré les optimisations proposées.
Le temps a par contre manqué pour la prédiction de l'accompagnement ou le développement d'approches différentes.

La poursuite du projet pourrait se faire par l'amélioration des performances en utilisant plus de données (notamment via la récupération d'autres bases). Nous avons proposé une première approche intégrant également la prédiction de la voix (information intégrée dans la fonction de perte), là encore limitée probablement par la taille de la base.

Annexes

Description des fichiers de code

Le dépôt GitHub dédié est accessible au lien ci-dessous :

https://github.com/MickaRiv/ProjetDatascientest-VoiceSeparator