UNIVERSITÉ DU QUÉBEC À MONTRÉAL

DÉTECTION ET CLASSIFICATION DES NOTES D'UNE PISTE AUDIO MUSICALE

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
QUENTIN VIGNAUD

JUIN 2020

REMERCIEMENTS

Comme pour toute chose accomplie en cette période, il y aurait bien trop de personnes à remercier, de tous nos aïeuls au plus sombre inconnu de nos contemporains. Mais entre ces extrêmes se trouvent certains qui le méritent plus que d'autres. Mes parents, grands-parents, et sœurs pour leur soutien depuis le Vieux Continent. Mes professeurs, actuels comme passés, pour leur vocation. Et mes très chers camarades et amis de tous bords.

Plus spécialement, je tiens à remercier pour cette maîtrise le professeur Bouguessa pour m'avoir accepté dans son équipe, et pour toutes ses invitations à de bons repas ; le professeur Privat pour ses cafés devenus rituels ; et mes collègues avec qui j'ai pu partager moultes idées plus ou moins judicieuses. Merci également à Pierre Pellegrin, qui a travaillé sur ces recherches lors de son stage à l'UQAM. Enfin, merci à Florian, Mélanie, Lâm, Oswald, Jehan, Ellen, Julien, Morgane, Aimeric, Éloïse, Victor, et Simon, et tous les autres, pour ces années passées au travers des hivers et étés de Montréal.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

RÉSUMÉ

Nous présentons dans ce mémoire des contributions aux domaines du MIR, *Musical Information Retrieval*, et à l'AMT, *Automatic Music Transcription*. Une échelle de classification des travaux d'AMT est proposée, afin de définir quels sont les champs d'application de chaque recherche. Cette échelle classe en fonction des contextes monophonique et polyphonique, ainsi que mono-instrumental et multi-instrumental.

Un outil est proposé, nommé Mélodium, permettant d'appliquer des traitements de signal à des données audio et musicales. Il gère également les données MIDI, *Musical Instrument Digital Interface*, ainsi que les fontes sonores. Cet outil est couplé à un langage de script permettant d'automatiser les traitements et de faciliter l'écriture d'expérimentations. Ce langage est axé sur l'application de traitements et la transmission des données entre-eux, en ôtant à l'utilisateur la responsabilité de manipulation de mémoire et d'ordonnancement.

En plus de diverses expérimentations réalisées avec Mélodium, nous présentons des modèles à base d'auto-encodeurs éparses et de réseaux de neurones à propagation avant, capables de retranscrire de la musique dans des contextes monophoniques et polyphoniques. Ces modèles exploitent notamment les descripteurs HPCP, *Harmonic Pitch Class Profiles*, MFCC, *Mel Frequency Cepstral Coefficients*, et GFCC, *Gammatone Frequency Cepstral Coefficients*, pour effectuer la transcription. Sur la base de ces expérimentations, nous proposons une méthode permettant d'étendre à tout intrument la transcription monophonique automatique, en détaillant les étapes et l'architecture nécessaire pour y parvenir.

Mots-clés : MIR, AMT, monophonique, polyphonique, MIDI, fonte sonore,
Mélodium, auto-encodeur, HPCP, MFCC, GFCC, signal

INTRODUCTION

0.1 Origine

Depuis que l'électronique et l'informatique ont rencontré le son et la musique au cours du siècle dernier, énormément de progrès communs à ces deux mondes ont été faits. Les systèmes techniques d'enregistrement et de restitution de l'onde sonore se sont sans cesse améliorés et continuent encore de le faire actuellement.

Dans un passé plus récent, depuis les années 1980 [1], a été développé un système, à la fois technique, normatif, et d'encodage, permettant de disposer de la donnée musicale et non plus uniquement de l'onde sonore. Il s'agit du système MIDI, pour *Musical Instrument Digital Interface* (Moog, 1986). Ce système permet de représenter numériquement les notes ainsi que d'autres caractéristiques musicales sur une plage de temps définie. Initialement utilisé par les synthétiseurs électroniques, le MIDI est exploité de nos jours par tous les logiciels de composition et de rendu musical qui se sont développés depuis sa création. Il est présentement le standard de fait pour véhiculer l'information musicale.

Nous disposons donc d'un moyen d'encoder la musique, et en parallèle de multiples techniques pour enregistrer le signal sonore, parmi lesquelles nous pouvons citer les formats PCM, Flac, ou MP3 et Vorbis/Opus [2]. Or, notre problématique com-

---

1. La première norme du système MIDI fut publiée en 1983.

2. PCM : *Pulse Code Modulation*, couramment utilisé dans le format Wave, *Waveform Audio File Format* ; Flac : *Free Lossless Audio Codec* ; MP3 : *MPEG-1/2 Audio Layer III* ; *Opus Interactive Audio Codec* ; la différence entre PCM & Flac ainsi que MP3 & Vorbis/Opus étant que les premiers conservent le signal tel qu'il est enregistré « sans perte », là où les seconds effectuent une compression excluant certaines composantes sonores.

mence ici : la génération d'un signal sonore à partir de données musicales est une tâche courante. Qu'il s'agisse d'un musicien jouant sa partition, d'un synthétiseur générant un signal, d'un logiciel faisant un rendu, nous disposons de multiples techniques abouties. Bien que ne pouvant être qualifiées de simples, elles sont éprouvées et ne représentent plus de difficultés. Mais il n'en va pas de même pour la tâche inverse.

0.2 Objectif

Notre problématique est la suivante : nous voulons établir une méthode permettant de retranscrire automatiquement la mélodie présente dans un signal audio (un enregistrement sonore), en des données musicales (un fichier MIDI, et par extension des partitions).

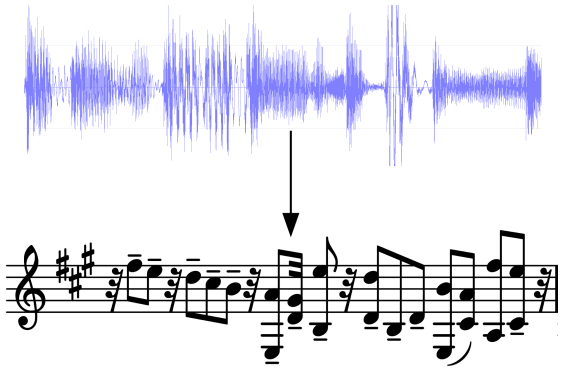


Figure 0.1: Illustration de l'objectif recherché.

Le processus de génération d'une combinaison de signaux n'est en l'état pas réversible, c'est-à-dire qu'à partir d'une piste audio, on ne peut pas retrouver de manière triviale la partition qui a permis de générer la musique. La retranscription musicale implique entre autres les problèmes de déconstruction de signaux, et de séparation des sources ; qui ne sont pas des tâches solvables de façon satisfaisante de manière automatique. La meilleure solution consiste encore à effectuer

le travail manuellement.

Aucun outil, qu'il soit commercial ou expérimental, n'est capable à ce jour de restituer depuis un fichier sonore plat des partitions convenables pour les instruments originels, et encore moins en disposant d'un signal sonore en direct. Le moyen le plus commun actuellement pour obtenir, en direct, une retranscription musicale est d'utiliser des instrument disposant d'une interface MIDI. Ceci afin d'avoir numériquement l'information musicale au même moment que le son est produit naturellement. Cette technique est lourde car elle requiert des instruments spécifiquement équipés, et est dans les faits relativement peu usitée lorsqu'elle n'est pas absolument nécessaire. Elle ne permet pas non plus de retranscription a posteriori.

0.3 Perspectives

Les implications de la possibilité de retranscrire automatiquement la musique sont multiples. En premier lieu, l'industrie musicale disposerait d'un moyen plus large pour analyser et relever des plagiats ou des utilisations non autorisées d'œuvres. Les plateformes et les instituts disposant de larges banques d'enregistrement pourraient en effectuer des classifications sur la base du contenu musical.

Pour ce qui est du domaine de la musicologie, ceci permettrait des retranscriptions et des analyses massives pouvant avoir un intérêt pour la recherche. Toujours dans le domaine de la recherche, nous avons déjà des méthodes expérimentales pour faire de la séparation de signal lorsque la partition de la musique est connue. Une telle extension permettrait de l'appliquer à toute musique, et de disposer alors de pistes audio distinctes pour chacun des instruments la composant.

D'un point de vue conservatoire, retranscrire une musique la pérennise, ainsi des instituts d'archivage auraient intérêt à appliquer ce traitement à leurs enregistrements. Également, avec des outils suffisamment souples, le monde de l'art pourrait

exploiter cette technologie afin de créer des scénographies, spectacles, ou œuvres diverses ayant une intégration plus poussée avec la musique, incluant l'improvisation et autorisant plus de liberté de jeu.

Enfin, au delà de l'application musicale, une technique de séparation et de classification d'éléments de signaux audio peut être utilisée dans de multiples autres contextes. Parmi ceux-ci nous pouvons citer en exemple la détection de défaillances mécaniques dans un cadre industriel, d'évènements sonores dans un cadre de sûreté, ou outre le domaine sonore, d'éléments dans des signaux électromagnétiques.

0.4 Contributions

Dans nos travaux, nous apportons les contributions suivantes :

- une échelle de classement de la transcription automatique de la musique ;
- un outil et langage de script pour effectuer des analyses sonores ;
- une méthode de création de modèles de transcription automatique.

0.5 Structure

Dans ce mémoire, au Chapitre 1, nous présentons tout d'abord une définition plus spécifique de notre problématique, et proposons une échelle de progression. Nous effectuons par la suite, au Chapitre 2, une revue des travaux connexes à nos recherches. Nous expliquons alors au Chapitre 3 le besoin d'un outil et le présentons, ainsi qu'au Chapitre 4 le langage associé. Viennent ensuite, dans le Chapitre 5, nos études empiriques où nous détaillons diverses situations possibles, à la suite desquelles nous exposons et développons au Chapitre 6 nos analyses des résultats obtenus. Enfin, nous proposons une méthode générique de création de modèles de classification audio musicale au Chapitre 7.

CHAPITRE I

DÉFINITION ET ÉCHELLE DE TRANSCRIPTION AUTOMATIQUE DE LA MUSIQUE

Le domaine de la restitution d'informations musicales s'appuie en grande partie sur la musicologie, qui est la science d'étude de la musique. Cette science porte aussi bien sur l'aspect culturel que physique, et même psychoacoustique [3]. La restitution d'informations musicales est une tâche non triviale, mêlant diverses problématiques. Tout d'abord, l'identification des fondamentales, harmoniques de premier rang, ou notes, présentes à un instant d'un signal. Ensuite, l'identification des timbres, ou instruments, liés à ces harmoniques. De plus, la distinction des différentes sources, les différents instruments, combinées en un seul signal. Ces trois problématiques d'ensemble constituent la tâche à accomplir pour pouvoir restituer les informations de n'importe quelle œuvre musicale. Mais vouloir les résoudre d'un seul tenant n'est ni raisonnable, ni encore envisageable.

Nous proposons donc une échelle, à laquelle peuvent se rapporter les différents travaux existants tout comme ceux présentés ici. Chacun des niveaux de cette échelle constitue une progression dans les capacités offertes par la transcription automatique de la musique. Nous détaillons chacun d'eux dans une section dédiée.

---

3. La psychoacoustique est la science décrivant la perception auditive humaine du son.

1	Monophonique mono-instrumental
2	Polyphonique mono-instrumental
3	Monophonique multi-instrumental
4	Polyphonique multi-instrumental indistinct
5	Polyphonique multi-instrumental distinct

Tableau 1.1: Échelle de transcription automatique de la musique.

1.1 Monophonique mono-instrumental

Premier niveau de la transcription automatique, la détection monophonique mono-instrumentale porte sur la restitution de l'harmonique de premier rang, ou note, jouée par un instrument. Ceci dans un contexte où l'on sait que seul cet instrument joue, et qu'au plus une note est produite à un instant donné.

Bien que semblant basique cet exercice est déjà complexe, car il requiert une connaissance du profil harmonique, ou timbre, de l'instrument examiné. Trois approches principales sont possibles :

- Générale, où un modèle agnostique vis-à-vis de l'instrument tenterait de classer au mieux des instants sonores dans la gamme des notes possibles ; cela au risque de perdre en précision selon les spécificités de l'instrument réellement à l'origine du son.
- Par instrument, où un modèle est conçu et entraîné spécifiquement pour un type d'instrument ; la difficulté étant alors d'effectuer une bonne généralisation à tous les instruments du même type.
- Par famille, où un modèle est conçu et entraîné pour répondre aux caractéristiques sonores d'une famille instrumentale partageant des timbres similaires, tels les violons, altos, violoncelles et contrebasses ; la détermination finale pouvant alors se faire par connaissance de la tessiture [4].

4. La tessiture est l'étendue des notes pouvant être jouées par un instrument.

Chacune de ces approches comporte des bénéfices et sacrifices. À défaut d'offrir un modèle universel, déterminer une architecture générique produisant des modèles spécialisés lors de l'apprentissage d'un type d'instrument, ou famille, est un compromis raisonnable qui constitue une grande avancée en soi.

1.2 Polyphonique mono-instrumental

La détection polyphonique mono-instrumentale consiste à restituer les différentes harmoniques fondamentales jouées par un unique instrument à un instant donné. Une telle détection permet de transcrire des accords et les mélodies usuelles des instruments polyphoniques, comme les cordes.

Les trois approches possibles sont les mêmes que pour le niveau monophonique mono-instrumental. La difficulté supplémentaire que constitue ce niveau de détection est particulière : il faut parvenir à distinguer de multiples fréquences harmoniques entre elles, et ce même en cas de superposition. L'exemple type est un jeu de la même note sur différentes octaves. L'émission d'une note la_3 produisant une fréquence de 440 Hz, ainsi que ses harmoniques à 880 Hz, 1 760 Hz, etc., et d'une note la_4 simultanée, de fréquence 880 Hz, devient très difficile à distinguer.

1.3 Monophonique multi-instrumental

Le niveau de détection monophonique multi-instrumental est l'augmentation naturelle de complexité survenant après les cas mono-instrumentaux. Il consiste en la détection d'une unique note à un instant donné, et l'attribution de cette note à un ou plusieurs instruments.

Bien que proposant un problème théoriquement intéressant, ce niveau de détection ne propose pas d'utilité pratique vis-à-vis de l'immense majorité des cultures

musicales. Il est en effet très rare qu'une œuvre multi-instrumentale ne fasse appel qu'à un seul instrument à la fois, ou que l'ensemble de ses instruments ne jouent toujours qu'une unique note à un instant donné. On peut constater que les problématiques concrètes se rapportent aux niveaux de cette échelle soit inférieurs, soit supérieurs.

1.4 Polyphonique multi-instrumental indistinct

Le niveau de détection polyphonique multi-instrumental indistinct consiste à détecter l'ensemble des harmoniques fondamentales jouées possiblement simultanément par de multiples instruments. Instruments qui ne sont à ce niveau pas distingués.

La problématique consiste ici à découvrir l'ensemble des notes sans confusion, celles-ci provenant de différents profils harmoniques, ou timbres, qui doivent donc être discriminés sans pour autant être classifiés. Un modèle de ce niveau permet de traiter la quasi totalité des œuvres musicales de toutes cultures et genres, en ce sens que leur mélodie est restituée, peu importe les instruments employés.

1.5 Polyphonique multi-instrumental distinct

Pour finir, le niveau de détection polyphonique multi-instrumental distinct correspond à la détection des harmoniques fondamentales jouées possiblement simultanément par différents instruments, incluant l'identification et l'attribution à un timbre.

Il s'agit ici de découvrir l'ensemble des notes et de savoir quels instruments les jouent. Un tel modèle est l'objectif ultime de la transcription automatique de la musique. Il constitue la possibilité de traiter toute œuvre musicale et d'en recons-

tituer les partitions la décrivant intégralement pour chaque instrument employé en son sein. Un tel modèle est possiblement un ensemble de modèles de niveau polyphonique mono-instrumental couplés à des méthodes de filtrage leur permettant d'ignorer les timbres autres que celui analysé. Cependant cette approche pose le problème de décomposition de signal dans laquelle une perte d'information survient, nécessitant de faire appel à des techniques de reconstruction de signal. Ces méthodes de filtrage et techniques de reconstruction sont encore à établir.

CHAPITRE II

REVUE DU DOMAINE DE LA RESTITUTION DES INFORMATIONS MUSICALES

Ce travail fait partie d'un ensemble portant sur la recherche d'informations musicales, MIR, pour *Musical Information Retrieval*. Ce domaine recoupe plusieurs disciplines parmi lesquelles la musicologie, la psychoacoustique, l'analyse de signal, l'apprentissage automatique, et plus généralement l'informatique. Le regard porté sur ce domaine se fait ici essentiellement depuis le point de vue des recherches en apprentissage machine.

2.1 Représentation des données

Une des premières caractéristique de ce domaine est la représentation des données à adopter en tant que vecteur d'entrée. Là où l'analyse de données visuelles peut se faire par des matrices de pixels brutes, l'analyse de signal audio requiert un choix, ou une composition de choix, entre les multiples représentations possibles. Les plus courantes sont le spectrogramme (pouvant être linéaire ou logarithmique), des résultats de filtrage du signal, ou des descripteurs sonores spécifiques. L'usage direct de la forme d'onde (couramment appelé « signal brut ») est moins populaire, et quasi systématiquement accompagné d'une autre représentation (Sigtia *et al.*, 2016).

S'agissant d'un signal continu, il est nécessaire d'effectuer un découpage en trames de celui-ci. Ce découpage est toujours opéré sur la forme d'onde lorsque des analyses ayant pour résultat des valeurs uniques sont opérées, ces valeurs résultantes qualifiant alors l'intégralité de la trame.

La fréquence d'échantillonnage de la forme d'onde, la largeur de trame, ainsi que le saut de décalage entre trames sont des facteurs primordiaux dans l'apprentissage et les résultats obtenus. Un enregistrement audio standard sur *Compact Disc* est échantillonné à 44 100 Hz, les enregistrements professionnels à 48 000 Hz, et la téléphonie utilise 7 000 Hz. Lors de l'application d'algorithmes d'apprentissage automatique, il est courant de réduire la fréquence d'échantillonnage à 16 000 Hz.

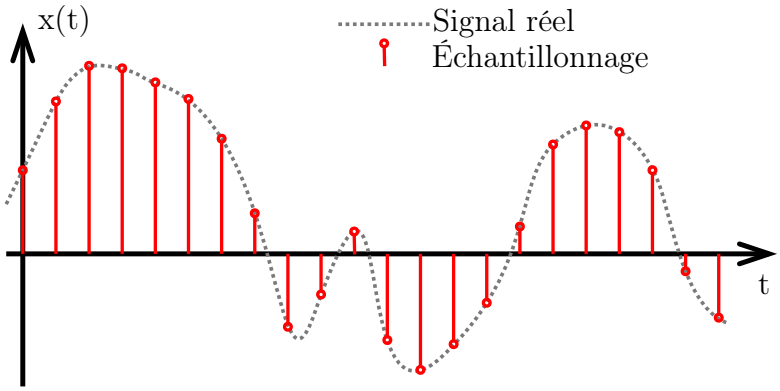


Figure 2.1: Échantillonnage d'un signal [5].

2.2 Descripteurs et transformations

2.2.1 Spectre

Le spectre sonore est la représentation de l'onde sonore une fois celle-ci traitée par la transformée de Fourier. Extrêmement utilisé dans le domaine de l'analyse audio,

5. Image adaptée de `https://commons.wikimedia.org/wiki/File:Sampled_signal.svg`, sous licence Creative Commons By-Sa 3.0.

le spectre sonore ou spectrogramme est couramment exploité comme descripteur dans les travaux du MIR. La décomposition de fréquences qu'il effectue le rend très utile pour extraire les fréquences harmoniques.

2.2.2 MFCC

Les coefficients cepstraux en fréquence de Mel, ou MFCC pour *Mel Frequency Cepstral Coefficients*, sont des descripteurs psychoacoustiques ; ils permettent une représentation du son sur la base de la perception auditive humaine (Davis et Mermelstein, 1980; Moore et Glasberg, 1983). Originellement développé dans le cadre de la reconnaissance et transcription vocale ; il existe de multiples implémentations du MFCC (Ganchev *et al.*, 2005).

2.2.3 GFCC

Les coefficients cepstraux en fréquences gammatones, ou GFCC pour *Gammatone-Frequency Cepstral Coefficients*, sont substantiellement équivalents aux MFCC, mais utilisent une banque de filtres gammatones calqués sur une échelle de largeurs de bandes rectangulaires (Shao *et al.*, 2009).

Un filtre gammatone est un filtre linéaire décrit comme une réponse impulsionnelle étant le produit d'une distribution gamma et d'une tonalité sinusoïdale (Patterson *et al.*, 1987). Tout comme le MFCC, GFCC est utilisé dans les travaux de reconnaissance et transcription vocale (Shao *et al.*, 2009).

2.2.4 HPCP

Les profils de classes de hauteur harmonique, HPCP pour *Harmonic Pitch Class Profiles*, sont des descripteurs conçus dans le cadre du MIR pour effectuer de la

détection d'accords musicaux (Fujishima, 1999). Ce descripteur se révèle particulièrement intéressant pour nos travaux puisqu'il consiste en un prétraitement extrayant les combinaisons harmoniques présentes dans un signal (Gómez, 2006).

2.3 Classification du genre musical

Les premières applications fonctionnelles de l'apprentissage machine ont été obtenues pour la classification du genre musical d'une piste audio. Tout d'abord, l'usage de modèles de Markov cachés ayant en vecteur d'entrée les coefficients cepstraux d'une piste audio permet de distinguer les enregistrements de voix, de musique, de rires, ainsi que nul de ces trois cas (Kimber *et al.*, 1997). D'autres méthodes plus avancées, faisant également usage de modèles de Markov cachés, permettent de détecter en temps réel les variations dans des films ou diffusions télévisées, les catégorisant en musique, chanson, dialogue avec fond musical, parole simple et environnement sonore quelconque sans harmonie (Zhang et Kuo, 2001). La distinction au sein d'un signal musical des séquences comprenant du chant et des séquences comprenant de la musique uniquement instrumentale est aussi réalisable (Berenzweig et Ellis, 2001).

L'extraction de spectre d'un signal audio permet une classification efficace du genre musical au moyen de modèles de mélange gaussien ou de k plus proches voisins (Tzanetakis et Cook, 2002). Selon le genre à classer, l'exactitude du meilleur modèle varie entre 61% et 88%. Il faut cependant garder à l'esprit que les genres musicaux sont subjectifs, et que la classification par des humains dans les travaux de Tzanetakis et Cook n'excédait guère les 70% d'exactitude.

2.4 Détection des instruments

Une autre application de l'apprentissage dans ce domaine, fortement corrélée avec la classification du genre, est la détection des instruments présents au sein d'une piste ou d'une section de signal. La présence ou l'absence d'un type d'instrument étant une information importante dans la détermination du genre, les premiers travaux dans le domaine en sont issus. À la différence du genre qui est une sonorité globale commune à différents signaux, la détection d'instruments est sensiblement plus complexe. Un instrument particulier a de multiples caractéristiques qui lui sont propres, et qui peuvent même dépendre d'un enregistrement précis, dû aux matériaux, à la forme, aux procédés de fabrication, au vieillissement, ainsi qu'au matériel utilisé pour l'enregistrement, microphone, pièce, environnement et technique de jeu. Tout ceci sans même parler d'effets sonores ajoutés ultérieurement. Il est évident, connaissant cela, de devoir disposer d'une grande variété dans les jeux de données, ce qui augmente d'autant le temps nécessaire pour effectuer l'apprentissage.

Dans leurs travaux, Barbedo et Tzanetakis font usage du spectre des signaux pour identifier les harmoniques présentes et relever les inharmonicités. Ils travaillent alors avec un algorithme d'élection de leur conception se basant sur les fréquences harmoniques, puis des analyses discriminantes linéaires pour extraire les instruments présents dans la section de signal étudiée. Une fois le signal intégralement analysé de cette manière, ils considèrent que si un instrument est détecté dans plus de 5% du signal, celui-ci est effectivement présent (Barbedo et Tzanetakis, 2011).

La méthode qu'ils utilisent s'avère efficace si suffisamment de données d'apprentissage sont disponibles. Tout d'abord, le taux de reconnaissance sur la banque de données d'apprentissage s'élève à 78%, ce qui est déjà bien meilleur que l'humain.

Des travaux sur des étudiants au conservatoire montrent que leur taux moyen de reconnaissance est de 55% (Srinivasan *et al.*, 2002). Ensuite, leur solution dispose d'une bonne capacité de généralisation, l'apprentissage effectué ayant pu détecter les instrument dans une autre banque d'enregistrements avec 76% de réussite.

2.5 Transcription automatique de la musique

À l'inverse de la classification du genre ainsi que la détection des instruments, la transcription automatique de la musique, ou AMT, pour *Automatic Music Transcription*, est une tâche non triviale qui ne bénéficie à ce jour d'aucune solution satisfaisante (Kelz et Widmer, 2017).

L'objectif de la transcription polyphonique est de déterminer les notes jouées par différents instruments au sein d'un signal audio sans avoir accès à la partition d'origine. Il s'agit justement de reconstituer cette partition par l'écoute, comme le feraient des humains. Les humains compétents dans ce domaine abordent le problème en donnant une importance à ce qu'ils s'attendent à entendre, en plus de ce qui est simplement présent lors de l'écoute. L'humain effectue donc une tâche d'anticipation.

En usant de RTRBM, pour *Recurrent Temporal Restricted Boltzmann Machine*, et de réseaux de neurones récurrents couplés à des RBM, *Restricted Boltzmann Machine*, Boulanger-Lewandoski, Bengio et Vincent présentent un moyen de reconnaître et anticiper des séquences musicales usuelles. Ils ne s'appuient pas sur le signal sonore mais sur des partitions déjà connues (Boulanger-Lewandowski *et al.*, 2012). L'intérêt de ces travaux réside dans le tri ou l'orientation musicale qui sont à rechercher lors de la transcription automatique d'une musique.

Des travaux de Benetos, Ewert et Weyde proposent un modèle d'extraction basé sur l'analyse probabiliste de composants latents, ou PLCA, pour *Probabilistic La-*

tent Component Analysis, permettant d'effectuer une transcription polyphonique automatique de la musique occidentale (Benetos *et al.*, 2014). Ce modèle permet également d'extraire les percussions présentes. Cependant, les résultats sont limités, la performance de détection des notes par instrument étant inférieure à 50% lorsqu'appliqué à des échantillons de la base de données MAPS (détaillée en Section 5.1.2).

Dans leurs travaux, Khlif et Sethu présentent un algorithme basé sur les factorisations non-négatives de matrices (Khlif et Sethu, 2015). Ils utilisent en prétraitement une transformation à Q constant sur le spectre audio d'une mélodie polyphonique de piano. Leur algorithme IMRNMF, pour *Iterative Multi Range Non-Negative Matrix Factorization*, permet ainsi d'obtenir des performances significativement meilleures qu'un NMF simple, avec une exactitude de 52% contre 38% pour le NMF et un score F1 de 69% contre 55% pour le NMF. La limitation d'application de ces travaux est qu'il faut déterminer empiriquement les paramètres de leurs algorithmes, ce qui empêche une réutilisation automatique sur d'autres instruments que le piano, qui est le seul sujet de leur étude.

Au moyen d'une architecture combinant un réseau de neurones récurrent et un NADE, *Neural Autoregressive Density Estimator*, il est possible d'établir un modèle de langage musical basé sur de multiples partitions (Sigtia *et al.*, 2015). Ce modèle permet ensuite d'effectuer une transcription polyphonique du piano lorsqu'il est appliqué par un réseau de neurones récurrent ou un réseau de neurones profond, combinés ou non. Cette méthode donne à son meilleur une exactitude de 53% et un score F1 de 69% sur 200 pistes provenant de la base MAPS (détaillée en Section 5.1.2).

CHAPITRE III

PROPOSITION D'UN NOUVEL OUTIL : MÉLODIUM

3.1 Nécessité d'un outil

Un problème se pose rapidement lorsque l'on souhaite effectuer de l'analyse sonore de manière modulaire tel que requis pour ces travaux : il n'existe pas d'outil permettant efficacement l'application de traitements, la gestion de données audio, et l'intégration d'algorithmes d'apprentissage machine. Certains outils permettent d'effectuer des tâches bien spécifiques, comme les utilitaires Sox et FFMPEG pour le traitement de fichiers audio, Audacity [6] pour les analyses, notamment spectrales, ou encore divers programmes dont l'utilisation pour ces recherches serait un détournement inconfortable de leur conception. On peut noter l'existence d'un projet nommé Marsyas [7], dont l'objectif de permettre l'analyse de signal est similaire aux besoins de nos travaux. Cependant son développement semble interrompu et ses dépendances sont aujourd'hui obsolètes et difficiles à reconstituer.

Quelque soit la qualité, l'avancement du développement, la continuité et l'actualisation de différents outils mentionnés, tous souffrent dans une mesure plus ou moins grande des inconvénients suivants :

---

6. Disponible à l'adresse `https://www.audacityteam.org/`.

7. Disponible à l'adresse `https://github.com/marsyas/marsyas`.

— développement *ad hoc* ;
— fragmentation technologique ;
— exécution peu ou pas optimisée ;
— automatisation restreinte ou impossible.

Pour satisfaire aux besoins de nos recherches, un outil de pré-traitement, analyse, apprentissage et restitution des données a donc été développé, dans le cadre de ce mémoire.

Notre outil est baptisé « Mélodium », néologisme formé à partir de
— « mélodie », tiré du latin *melodia*, lui-même issu du grec ancien μελῳδία, *melôidía* (« chant ») composé de
  — μέλος, *mélos* (« arrangement musical ») ;
  — et ᾠδή, *ôidê* (« chant ») ;
— et « medium », intermédiaire.

Un mélodium est donc un intermédiaire entre les mélodies, entre les arrangements musicaux.

Mélodium dispose de sources et d'une documentation disponibles à l'adresse `https://gitlab.com/qvignaud/Melodium`. L'ensemble est prévu pour être compilé de manière conventionnelle sur une machine classique, un fichier de référence étant fourni à la racine du dépôt pour les informations techniques.

3.2 Tâches spécifiques

Afin d'effectuer tous les traitements nécessaires aux expérimentations, bon nombre de tâches sont nécessaires. Ces tâches sont pour certaines orientées vers la recherche en restitution d'informations musicales, et d'autres plus générales. Toujours est-il que l'assemblage à effectuer entre elles est spécifique à ce domaine, ce qui peut complexifier l'usage de certains outils existants non prévus à cet effet.

Parmi ces tâches nous pouvons citer :

- la modélisation des fichiers MIDI ;
- la conversion des multiples formats audio ;
- la décomposition de fontes sonores ;
- l'intégration de banques d'enregistrements sonores ;
- l'analyse et le traitement algorithmique du signal sonore ;
- la persistance des données extraites et l'échange interopérable (CSV, images) ;
- l'interfaçage avec des modèles d'apprentissage machine ;
- l'apprentissage et l'exploitation des modèles d'apprentissage machine ;
- la restitution et compilation des résultats d'apprentissage en données musicales ;
- l'interprétation et l'évaluation des résultats.

3.3 Composants techniques

Le programme est écrit en C++ et repose sur la bibliothèque Qt [8] pour disposer d'un environnement de développement et de la plupart des fonctionnalités conventionnelles n'étant pas propre aux besoins spécifiques de ces recherches. Une des premières nécessités techniques, manquant à la quasi-totalité des outils évoqués, est la parallélisation des tâches. Mélodium repose massivement en interne sur les patrons de conception fabriques et commandes, couplés à un groupe de fils. Cela requiert une conception de classes conteneures susceptibles d'êtres accédées par de multiples fils d'exécution en simultané, notamment dans le cadre de flux entre algorithmes. Ceci est implémenté dans Mélodium essentiellement au moyen de mutexes et sémaphores, en plus d'un système d'ordonnancement des commandes voué à éviter des accès concurrents et mises en attente de fils pour accéder aux

8. Documentation à l'adresse `https://doc.qt.io/`.

données.

Pour l'analyse de signaux, Mélodium dispose d'un ensemble d'interfaces permettant l'exploitation transparente de la majorité des algorithmes implémentés dans la bibliothèque Essentia (Bogdanov *et al.*, 2013), au nombre de 173 au moment de cette rédaction. Seuls quelques algorithmes spécifiques d'Essentia ne disposent pas présentement de possibilité d'exploitation au travers de Mélodium, mais il s'agit d'algorithmes relatifs à l'import et export de données, rendus de facto inutiles au sein de ce programme. L'interfaçage avec Essentia est conçu pour prendre en charge d'éventuels nouveaux algorithmes pouvant être ajoutés ultérieurement, sans recompilation de Mélodium.

Pour l'exploitation des fichiers MIDI, il est fait usage de QMidi [9]. Il s'agit d'un développement connexe visant à fournir un support du MIDI, fichiers comme flux, intégré à l'environnement Qt. Si une conversion de format audio est nécessaire, Mélodium s'appuie sur la bibliothèque FFMPEG [10], communément répandue au sein des systèmes de famille Unix. Cela permet une prise en charge aisée d'un grand nombre de codecs sans effort de développement majeur, et un coût d'exploitation mineur au sein du projet, se concentrant exclusivement autour de PCM et Flac comme formats de travail. Là encore, la prise en charge de nouveaux formats audio par FFMPEG ne requiert aucune modification au sein de Mélodium, qui prendra dès lors ces nouveaux formats en compte de façon transparente.

Concernant l'exploitation d'algorithmes d'apprentissage, la bibliothèque Mlpack (Curtin *et al.*, 2018) est utilisée. Différents types de traitements sont effectués par son biais, à savoir :

— l'instanciation de modèles ;

9. Dépôt Git à l'adresse `https://gitlab.com/qvignaud/QMidi`.

10. Site du projet à l'adresse `https://ffmpeg.org/`.

— l'apprentissage ;

— la classification ;

— l'évaluation ;

— la sauvegarde ;

— et l'exploitation productive de ces modèles.

À des fins d'optimisation des données comme des calculs, les bibliothèques OpenMP [11], Armadillo (Sanderson et Curtin, 2016), et Ensmallen (Bhardwaj *et al.*, 2018; Sanderson et Curtin, 2019), sont exploitées.

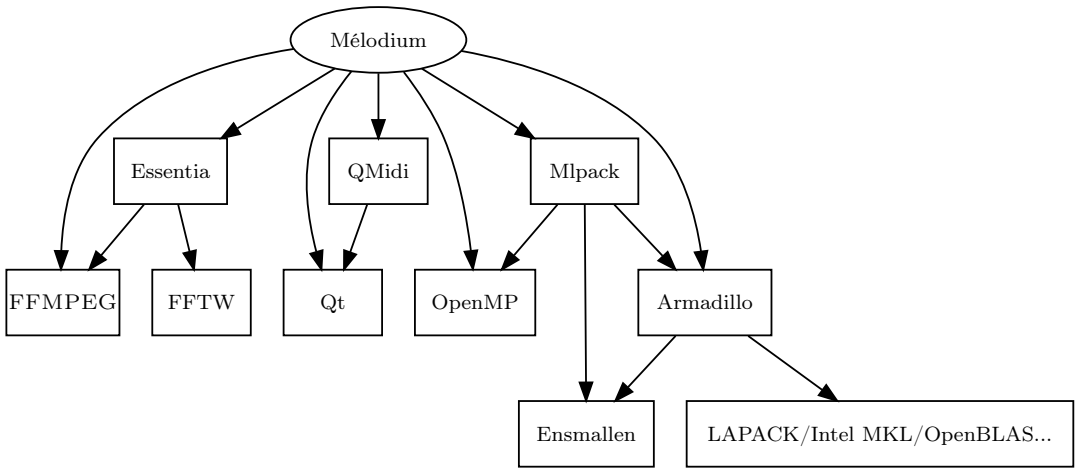


Figure 3.1: Graphe de dépendance de Mélodium.

Enfin, pour l'interaction du programme et la visualisation des données, aussi bien en terminal que graphiquement, les fonctionnalités d'interface utilisateur de Qt sont utilisées. Raison d'être initiale de cette bibliothèque, elles permettent d'effectuer tous les rendus visuels, sous différentes formes possibles, dont il y aurait besoin.

---

11. Page du projet `https://www.openmp.org/`.

3.4 Architecture

Mélodium s'architecture autour de différents types d'éléments : les pistes, les traitements, les exécutants, les modèles, les préparateurs, et les initialisateurs.

3.4.1 Pistes

Les pistes sont des conteneurs descriptifs des données présentes dans un enregistrement. Les données peuvent soit êtres globales, qualifier l'ensemble du signal ; soit simples, qualifier par une unique valeur, un instant ou trame du signal ; soit détaillées, associant alors de multiples valeurs à un instant ou trame du signal. Les qualificateurs simples peuvent être perçus comme des tableaux unidimensionnels, et les détaillés comme des tableaux bidimentionnels.

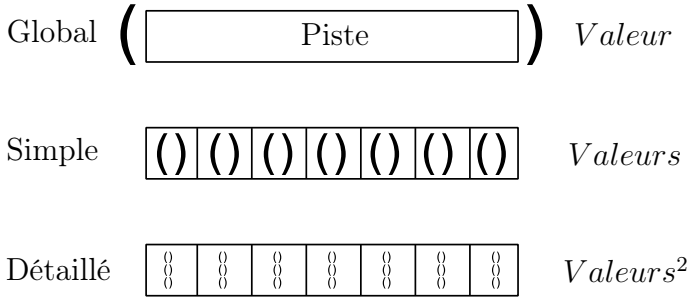


Figure 3.2: Types de données présentes dans une piste.

Les pistes sont conçues pour gérer des accès concurrents dans un contexte d'exécution multifil, et pour maintenir la cohérence des données par rapport au signal étudié (notamment en termes de longueur et nombre de trames).

3.4.2 Traitements

Les traitements sont les descripteurs et paramétreurs des opérations qui vont être appliquées sur des données. Ils jouent différents rôles parmi lesquels :

— convertisseur ;
— rendu audio ;
— analyste de signal ;
— alimenteur de modèle ;
— entraîneur de modèle ;
— prédicteur de modèle ;
— enregistreur de modèle ;
— chargeur de modèle ;
— exporteur de données.

Un traitement peut être indépendant de toute piste, être lié à une unique piste, ou bien opérer sur de multiples pistes. La liste des traitements existants est disponible en Annexe A. Un traitement configure et supervise l'exécutant qui lui est lié unitairement.

3.4.3 Exécutants

Les exécutants sont des éléments qui effectuent les opérations. Ils le font chacun dans un fil d'exécution qui leur est alloué, avec la possibilité d'en instancier de nouveaux à leur propre charge (au travers d'OpenMP notamment). En cas d'erreur lors de l'exécution, ils interceptent toute exception ou échec et le signalent à leur traitement sans faire échouer le programme dans son ensemble, dans le respect des principes résilience aux erreurs et de sortie grâcieuse.

3.4.4 Modèles

Les modèles sont des éléments qui fournissent divers traitements, dépendamment de leurs capacités. Ils renferment usuellement un modèle d'apprentissage machine, bien que ce ne soit pas nécessairement le cas. Les traitements et exécutants mis à

disposition peuvent être les alimenteurs, entraîneurs, prédicteurs, enregistreurs et chargeurs.

Trois modèles sont actuellement disponibles :

- `ConfigurableRectFfn`, pour des réseaux de neurones à propagation avant à taille constante ;
- `ConfigurableVarFfn`, pour des réseaux de neurones à propagation avant dont les couches intermédiaires peuvent avoir des tailles variables ;
- `SparseAutoencoder`, pour des auto-encodeurs éparses.

3.4.5 Initialisateurs et préparateurs

Les initialisateurs sont les éléments d'entrée du programme. Ils mettent en place et configurent les pistes, les modèles, et les traitements qui sont utiles à leur tâche. Pour ce faire, ils s'appuient notamment sur les préparateurs, dont l'objectif est d'effectuer un tri et une sélection des données et fichiers qui leur sont confiés. Typiquement, un préparateur gère des fichiers, en extrait les données, et s'assure de les rendre exploitables au moyen des traitements adéquats, sur lesquels peuvent être connectés les traitements que l'initialisateur appelant établi. Leur usage est exposé en section 3.5.2.

Les trois préparateurs actuellement disponibles sont :

- `AudioFilePreparator`, pour le chargement de fichiers de données audio (impliquant leur conversion vers un format de travail si nécessaire) ;
- `MidiFilePreparator`, pour le chargement de fichiers de données musicales MIDI ;
- `SoundFontPreparator`, pour le chargement de données provenant de fontes sonores.

Les initialisateurs peuvent les utiliser pour appliquer leurs traitements à des don-

nées convenablement préparées.

3.4.6 Ordonnanceurs

Une fois les pistes, modèles, et traitements instanciés et configurés, le programme passe la main à un ordonnanceur. Ce dernier effectue la gestion de l'exécution macroscopique des traitements, leur exécution détaillée étant à la charge de leur exécutants. L'ordonnanceur place un traitement qu'il juge opportun d'effectuer dans un fil d'exécution disponible. Pour faire cette sélection un ordonnanceur se base sur les traitements déjà achevés afin de poursuivre la gestion intégrale d'une piste pour la compléter au plus vite, libérant alors l'espace mémoire qu'elle occuppe. Ceci dans l'objectif de n'en utiliser qu'une quantité raisonnable, sans sacrifier au temps d'exécution global. Le nombre de fils d'exécution, et donc de traitements simultanés, est fixé par l'utilisateur, avec un choix par défaut dépendant de la machine.

Nous avons actuellement implémenté deux ordonnanceurs :

- `MonoThreadScheduler`, utilisé si un seul fil d'exécution est imposé ou disponible ;
- `MultiThreadScheduler`, utilisé dans les autres cas.

Notre programme est conçu pour permettre l'implémentation de nouveaux ordonnanceurs, notamment pour l'exploitation de MPI [12].

---

12. *Message Passing Interface*, norme de communication pour des systèmes de grappes massivement parallèles à mémoire distribuée, informations et documentation disponibles à `https://www.mpi-forum.org/docs/`.

3.5 Usage et fonctionnalités

3.5.1 Ligne de commande

Une interaction classique est disponible en ligne de commande, permettant d'exécuter Mélodium ou d'obtenir des informations documentaires.

```
melodium [options] <initializer> [params]
```

Les paramètres et commandes les plus utiles sont ici décrits.

- `--graph`, génère un graphe au format GraphViz représentant l'enchaînement des traitements ; les graphes d'exécution de ce mémoire sont générés par ce moyen.
- `--threads`, assigne le nombre de processus avec lesquels Mélodium doit composer son ordonnancement, par défaut cela équivaut au nombre de cœurs existants sur la machine.
- `--noexec`, n'exécute pas les traitements, les initialise seulement, et génère éventuellement le graphe si demandé ; ceci est utile pour vérifier la cohérence des tâches demandées, avant de lancer la charge de travail réelle sur une machine tierce par exemple.
- `--list-{initializers,models,treatments}`, liste les éléments disponibles du type demandé.
- `--info-{initializers,models,treatments}`, affiche la documentation de l'élément demandé, un exemple est disponible en Annexe B.

3.5.2 Initialisateurs

Le comportement par défaut de Mélodium est de considérer le premier paramètre textuel non relatif à une option comme l'initialisateur à utiliser. Les éléments

d'appel de la ligne de commande apparaissant après le nom de l'initialisateur sont considérés comme ses options.

Parmi les initialisateurs se trouvent notamment :

— `midigenerator`, permettant la génération de pistes MIDI correspondant aux besoins requis, paramétré pour les générer avec un contenu déterminé ou avec des composantes aléatoires ;

— `script`, qui va suivre les instructions définies dans un fichier script pour établir les traitements à exécuter, c'est ce dernier que nous développons dans le chapitre suivant.

3.6 Performances

L'outil Mélodium est utilisé pour l'intégralité des travaux expérimentaux présentés ici. Il exécute avec succès des expériences manipulant de larges quantités de données avec accès concurrents. Plus de 1 500 000 traitements sont nécessaires à certaines expérimentations, qui ont été menées à bien en quelques dizaines d'heures sur des machines haute performance, comme décrit en Section 5.6. L'Annexe D donne davantage de renseignements sur les ressources utilisées.

CHAPITRE IV

LANGAGE DE MANIPULATION DE DONNÉES AUDIO ET MUSICALES

Un interpréteur de scripts est présent dans Mélodium. Il vise à simplifier la création d'expérimentations sans passer systématiquement par l'implémentation d'un initialisateur dédié et la recompilation, permettant alors à des utilisateurs sans connaissance technique approfondie des technologies associées, ni même du fonctionnement interne, d'utiliser le programme.

Le langage utilisé au sein de ces scripts se veut paramétrique plus que programmatique. Il permet de paramétrer des éléments existants au sein du programme mais pas d'en créer de nouveaux. Une conséquence de ceci est que l'ordre d'apparition des instructions n'a aucune importance, tant que les blocs sont cohérents. Ce langage est implicitement parallèle et multipistes (cette notion est abordée en section 4.7). Les éléments spécifiés dans un script sont tous susceptibles d'être exécutés parallèlement, sous réserve d'une indépendance entre eux.

La syntaxe est globalement inspirée des langages C et Dot. On retrouve ainsi des mots-clés précédant les déclarations, des blocs délimités par des accolades, et des assignations et paramétrages similaires à des appels de fonction. Les éléments présents dans ce langage sont multiples, sont abordés ici les principaux, ainsi que quelques règles. Par ordre d'apparition : les paramètres et leurs types, les traitements, les séquences, les connexions, les préparateurs, les modèles, et finalement

la notion de multipistes. Divers exemples de scripts fonctionnels sont disponibles en Annexe C.

4.1 Paramètres et types

Les paramètres existent dans différents contextes pour lesquels les détails seront fournis au moment opportun. Les paramètres prennent des valeurs de type booléennes, numériques, chaînes de caractères, ou tableaux, et sont généralement nommés. Il est à noter qu'il ne s'agit jamais d'une déclaration, mais d'une assignation, tout paramètre existant implicitement avec une valeur par défaut. La syntaxe d'un paramètre est :

```
nom = <valeur>
```

Selon le type de la valeur, la syntaxe est comme suit :

- booléen : « `true` » ou « `false` », littéralement ;

  ```
  startFromZero = true
  ```

- nombre : nombre réel en notation décimale classique « `-1.234` », la décimale se faisant au point ;

  ```
  duration = 1.35
  ```

- chaîne de caractères : chaîne de caractères quelconque encadrée de doubles guillemets droits « `"` », avec échappement pour les caractères guillemets « `"` » et contre-oblique « `\` » par contre-oblique-guillemets et double-contre-oblique respectivement « `"...\"...\\..."` » ;

  ```
  type = "blackmanharris92"
  ```

- tableau : l'un des trois types précédents, selon leur syntaxe, séparés par des virgules « `,` », le tout encadrés de crochets « `[` » et « `]` ».

```
frequencies = [261.63, 277.18, 293.66, 311.13, 329.63]
```

4.2 Traitements

Les traitements sont déclarés par leur nom suivi de la liste de paramètres qui leur sont associés, encadrée de parenthèses. L'ordre des paramètres est arbitraire, et les paramètres non spécifiés sont assignés à leur valeur par défaut. Un traitement peut soit être déclaré avec le nom de son type, soit porter un nom arbitraire et avoir son type spécifié avant la liste de paramètres.

```
// Traitement de type 'Spectrum' avec ses paramètres laissés à défaut.
Spectrum()
// Traitement de type 'Spectrum' avec la taille fixée à 4096.
Spectrum(size=4096)

// Traitement de type 'SpectralPeaks' avec ses paramètres.
SpectralPeaks(sampleRate=44100, orderBy="magnitude")
// Traitement nommé 'Pics' de type 'SpectralPeaks' avec ses paramètres.
Pics(SpectralPeaks, sampleRate=44100, orderBy="magnitude")
```

4.3 Séquences

Les séquences sont les éléments qui abritent les traitements et les lient. Une séquence définit la tâche qui va être effectuée lors de l'exécution, elle est l'élément principal du script. Une séquence est déclarée par le mot-clé « `sequence` », suivi du nom de la séquence, et de son contenu encadré d'accolades « `{ /*...*/ }` ».

```
sequence maSequence {
    /* ... */
}
```

4.4 Connexions

Les connexions sont les éléments qui définissent les liens entre les traitements. Une connexion indique deux choses : quels sont le traitement antérieur et le traitement postérieur, et quelle est la donnée éventuellement émise et reçue. La syntaxe de base est de lier deux traitements par un tiret-chevron-fermant « -> », en détaillant le nom de la donnée en sortie et en entrée au moyen du point. Ici, la sortie `audio` du traitement `MonoLoader` est connectée à l'entrée `signal` de `FrameCutter` :

```
MonoLoader.audio -> FrameCutter.signal
```

Il est également possible d'enchaîner les déclarations de connexions en les positionnant successivement, et spécifier le nom de donnée de sortie et d'entrée en les séparant d'une virgule. Ici, à la suite de l'exécution de `FrameCutter`, la sortie `frame` est connectée à l'entrée `frame` de `Windowing` :

```
MonoLoader.audio -> FrameCutter.signal,frame -> Windowing.frame
```

Si plusieurs données doivent être transmises entre deux mêmes traitements, leurs déclarations sont distinctes. Le nombre de tirets précédant le chevron peut être arbitraire, afin de faciliter la lecture par alignement et structurer le code.

```
SpectralPeaks.frequencies -> HPCP.frequencies
SpectralPeaks.magnitudes --> HPCP.magnitudes
```

Dans certains cas spécifiques, il se peut qu'aucune donnée ne soit transmise entre traitements ; on omet alors simplement le nommage de donnée.

```
Feeder -> Trainer
```

4.5 Préparateurs

Les préparateurs sont chargés de préparer les données d'entrée à partir des fichiers. Un préparateur est un élément contenu dans une séquence, qui ne peut en avoir au plus qu'un. Il est déclaré par le mot-clé « `preparator` », suivi du nom désiré, puis du type de préparateur entre parenthèses « `(...)` », et enfin de son contenu encadré d'accolades « `{ /*...*/ }` ».

```
preparator audioFiles(AudioFile) {
        /* ... */
}
```

Dans le corps d'un préparateur sont assignés ses différents paramètres, comprenant au moins `sampleRate`, `frameSize`, et `hopSize`. Ces trois valeurs correspondent respectivement au taux d'échantillonnage, à la taille des trames, et au décalage entre trames. Le chemin des répertoires ou fichiers à traiter pour le préparateur sont donnés anonymement, d'après la syntaxe des chaînes de caractères. Le comportement qu'adopte un préparateur vis-à-vis de ces chemins, ainsi que les paramètres supplémentaires dont il dispose, sont détaillés dans la documentation de son type.

```
preparator audioFiles(AudioFile) {
        sampleRate = 44100
        frameSize = 4096
        hopSize = 2048

        "/home/quentin/Musique"
        "/home/quentin/Bureau/Enregistrements"
}
```

Les traitements établis par un préparateur sont accessibles dans la séquence. Ils sont désignés par le nom donné au préparateur, suivi du type du traitement encadré de crochets « `[...]` ». Leur usage est le même que celui d'un traitement classique. Ici, la sortie `audio` du traitement `MonoLoader` établi par le préparateur `audioFiles` est connectée à l'entrée `signal` de `FrameCutter` :

```
audioFiles[MonoLoader].audio -> FrameCutter.signal
```

Il est également possible de les baptiser spécifiquement comme cela se ferait avec un traitement instancié explicitement. Ici, le traitement `MonoLoader` établi par le préparateur `audioFiles` est appelé `AudioLoader` :

```
AudioLoader(audioFiles[MonoLoader])
AudioLoader.audio -> FrameCutter.signal
```

Chaque type de préparateur voit les traitements qu'il établi décrits dans sa documentation.

4.6 Modèles

Les modèles sont des entités existant indépendamment des séquences. Un modèle est déclaré par le mot-clé « `model` », suivi du nom désiré, puis du type de modèle entre parenthèses « `(...)` », et enfin de son contenu encadré d'accolades « `{ /*...*/ }` ». Dans le corps d'un modèle sont assignés ses différents paramètres, totalement dépendants du type de modèle.

```
model reseau(ConfigurableRectFfn) {
        layers = ["PReLU", "PReLU", "Sigmoid"]
        inputSize = 1200
        layersSize = 400
        outputSize = 128
}
```

Tout modèle déclaré rend accessible dans les séquences les types de traitements qui y sont associés. On instancie un traitement lié à un modèle en utilisant le nom du modèle suivi du type de traitement encadré de crochets « `[...]` ».

```
FFNFeeder(reseau[MlpackModelFeeder])
HPCP.hpcp ----------------------> FFNFeeder.data
MidiAnalyst.midiTrackContent1 --> FFNFeeder.label
```

L'usage de ces traitements est le même que celui d'un traitement classique, avec cependant une éventuelle contrainte logique dans l'ordre d'exécution, dépendamment des fonctionnalités du traitement. On alimentera ainsi un modèle avant d'effectuer son apprentissage, tout comme on le chargera avant de demander des prédictions.

4.7 Multipistes

Les traitements d'un type monopiste (l'essentiel des traitements), bien que spécifiés une seule fois au sein d'un script, sont répliqués pour chaque piste fournie par le préparateur. Les traitements d'un type multipistes (tel que `Evaluator`, qui effectue des évaluations comparatives entre pistes) ou indépendant (tel que `MlpackModelLoader`, un chargeur de modèles) ne sont eux instanciés qu'une unique fois, et leurs connexions sont dupliquées pour chaque traitement monopiste auxquels ils sont connectés.

Ainsi, l'exemple suivant, en considérant que le préparateur `audioFiles` relève trois fichiers audio, donnera le schéma d'exécution en Figure 4.1 :

```
AudioLoader(audioFiles[MonoLoader])
FFNFeeder(reseau[MlpackModelFeeder])
FFNTrainer(reseau[MlpackModelTrainer])

AudioLoader.audio -> FrameCutter.signal,frame -> FFNFeeder.data
FFNFeeder ------------------------------------> FFNTrainer
```

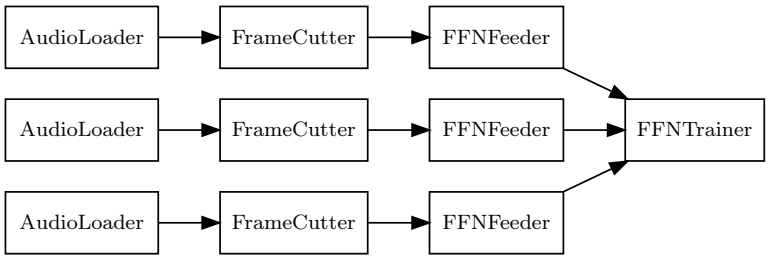


Figure 4.1: Exemple d'exécution multipistes.

CHAPITRE V

ÉTUDES EMPIRIQUES DE DÉTECTION ET CLASSIFICATION DES NOTES DE MUSIQUE

5.1 Choix et constitution des jeux de données

Les jeux de données disponibles pour travailler sur l'apprentissage et la transcription automatique de la musique sont limités. Ceci est dû à la difficulté particulière de disposer à la fois d'enregistrements réels et de données MIDI synchronisées avec ces premiers. Il existe quelques jeux de données populaires dans la littérature, et des études font également usage de fontes sonores pour effectuer une génération automatique de pistes audio synchronisées avec l'information MIDI.

5.1.1 Fonte sonore en jeu monophonique

Pour effectuer un apprentissage monophonique sur de multiples instruments, tel que présenté en Section 1.1, nous nous basons sur la fonte sonore Fluid R3 GM & GS, créée par Frank Wen et Toby Smithe, disponible sous licence MIT [13]. Elle comporte plus de 1 400 échantillons sonores correspondants à plus de 190 instruments. Une fonte sonore, ou *soundfont*, est un ensemble d'échantillons audio décrivant un ou

13. Disponibles notamment sous `https://packages.debian.org/sid/fluid-soundfont-gm` et `https://packages.debian.org/sid/fluid-soundfont-gs`.

plusieurs instruments. Ces échantillons sont associés à des spécifications de modulation et transformations sonores pour permettre à des synthétiseurs de restituer les sonorités des instruments contenus dans la fonte.

En raison du grand nombre d'instruments et des ressources nécessaires pour effectuer un entraînement, un représentant par famille instrumentale a été choisi. Les caractéristiques sonores étant partagées au sein des familles, on peut émettre l'hypothèse raisonnable qu'un modèle apprenant avec succès sur un membre d'une famille avec des paramètres spécifiques est apte à reproduire cet apprentissage sur les autres membres.

Le Tableau 5.1 expose les instruments sélectionnés ainsi que l'étendue associée. L'étendue est calquée sur la tessiture propre à chaque instrument, telle qu'échantillonnée dans la fonte sonore d'origine. Bien qu'un synthétiseur MIDI soit tout à fait capable de générer le jeu de tout instrument à n'importe quelle note, il n'est pas pertinent de s'intéresser à des sonorités impossibles à produire en réalité.

Rangs	Famille	Numéro	Instrument	Tessiture [14]
000-007	Pianos	*000*	*Piano à queue*	26-108
008-015	Percussions chromatiques	*013*	*Xylophone*	54-108
016-023	Orgues	*019*	*Orgue d'église*	39-91
024-031	Guitares	*024*	*Guitare classique*	40-87
032-039	Basses	*032*	*Basse acoustique*	28-48
040-047	Cordes	*040*	*Violon*	28-72
048-055	Orchestre	*048*	*Ensemble de cordes*	36-43
056-063	Cuivres	*057*	*Trombone*	48-84
064-071	Instruments à anches	*064*	*Saxophone soprano*	41-84
072-079	Flûtes	*075*	*Flûte de pan*	61-90
080-095	Synthétiseurs solos	*080*	*Signal carré*	72-93

Tableau 5.1: Instruments sélectionnées pour l'apprentissage monophonique.

14. 26 étant le réo, 28 le mio, 36 le do1, 39 le ré♯1, 40 le mi1, 41 le fa1, 43 le sol1, 48 le do2, 54 le fa♯2, 61 le do♯3, 72 le do4, 84 le do5, 87 le ré♯5, 90 le fa♯5, 91 le sol5, 93 le la5, et 108 le do7.

Les signaux sonores générés sont constitués d'une seconde de chaque note de l'étendue, espacées d'une seconde de silence. Le temps de jeu d'une seconde permet de rendre compte des variations sonores de l'instrument, le son comprenant alors l'attaque comme le maintien de la note. La seconde d'espacement permet d'avoir le déclin de la note précédente ainsi que du silence bruité et pur.

5.1.2 Échantillons et œuvres en jeu polyphonique

Pour effectuer des apprentissages sur des timbres de piano provenant de différents instruments, ainsi que des apprentissages en polyphonie, nous faisons usage de la base de donnée MAPS [15], qui est une base de données d'enregistrements audio de piano réel et synthétisé synchronisés avec leur transcription en MIDI. Les enregistrements contiennent des notes isolées et sons monophoniques, des accords usuels ou alétoires, ainsi que des œuvres complètes. Très utilisée dans la recherche en transcription automatique de la musique, elle fût établie par Valentin Emiya et est disponible sous licence Creative Commons By-Nc-Sa par l'institut Télécom ParisTech en France (Emiya, 2008; Emiya *et al.*, 2010).

Quatre jeux sont constitués depuis cette base :

- *Isolé* : Des notes sont jouées seules, indépendamment les unes des autres. Ceci constitue un jeu de données similaire à celui généré en 5.1.1.
- *Aléatoires* : Des notes sont jouées simultanément dans des combinaisons aléatoires, sans considération musicale préalable, avec une polyphonie comprise entre deux et sept voix.
- *Usuels* : Des accords usuels sont joués. Ce sont des données comportant une connaissance intrinsèque des règles musicales.

15. Disponible à l'adresse `http://www.tsi.telecom-paristech.fr/aao/2010/07/08/maps-database-base-de-sons-de-piano-pour-la-transcription-automatique-de-musique/`.

— *Musiques* : Des œuvres musicales « classiques » sont présentées sous forme d'enregistrement audio couplé à leur annotation MIDI. Il va de soi qu'une considération des règles musicales est présente dans ces données, qui sont aussi susceptibles d'introduire un biais culturel ou stylistique.

Tous ces jeux sont composés en sélectionnant aléatoirement un certain nombre de fichiers les constituant pour obtenir une durée de signal d'environ une heure (un fichier unitaire n'étant jamais sectionné). Cette sélection aléatoire est effectuée indépendamment pour chaque expérimentation.

5.2 Procédures de prétraitement

Disposer des fichiers audio, sonores comme musicaux, est une chose, avoir des vecteurs d'entrée pertinents pour les modèles d'apprentissage en est une autre. Beaucoup de travaux reposent sur l'analyse du signal ou de son spectre sans transformations majeures. Nous avons décidé d'utiliser des descripteurs plus spécifiques.

5.2.1 HPCP

Le HPCP, Section 2.2.4, est ici exploité dans sa forme classique, de telle manière que les fréquences analysées s'étendent de 40 Hz à 5 000 Hz, que la déviation est estimée par rapport au la3 440 Hz, et que les huit premières harmoniques soient considérées. On utilise la fonction cosinus pour la pondération, et nous dimensionnons la sortie à 120, qui est un bon niveau de détail, le HPCP ne produisant que des vecteurs de taille multiple de 12. Le code fonctionnel est disponible en Annexe C.1.

5.2.2 MFCC & GFCC

En supplément du HPCP, nous expérimentons également les descripteurs MFCC et GFCC. Le HPCP seul ayant à priori comme faiblesse de ne pas rendre compte des différentes octaves possibles. Pour MFCC et GFCC nous effectuons un filtrage préalable du spectre selon les bandes décrites au Tableau 5.2. Chacune des bandes est dimensionnée à 12, pour chaque note de l'octave, le total des dix bandes des deux descripteurs faisant 240. Le code fonctionnel est disponible en Annexe C.3.

Fréquence basse	Fréquence haute	Octave correspondante
15,90 Hz	31,25 Hz	-1
31,25 Hz	62,50 Hz	0
62,50 Hz	125,00 Hz	1
125,00 Hz	250,00 Hz	2
250,00 Hz	500,00 Hz	3
500,00 Hz	1 000,00 Hz	4
1 000,00 Hz	2 000,00 Hz	5
2 000,00 Hz	4 000,00 Hz	6
4 000,00 Hz	8 000,00 Hz	7
8 000,00 Hz	16 000,00 Hz	8

Tableau 5.2: Bandes de filtrage de fréquences.

Ce filtrage permet de distinguer chaque octave indépendamment des autres. MFCC et GFCC étant des descripteurs psychoacoustiques, nous pouvons considérer qu'ils représentent davantage le son tel que nous le percevons plutôt que tel qu'il est en termes de physique pure.

5.2.3 Spectre

En plus de tout cela, des expérimentations portant sur la combinaison du HPCP et du spectre simple sont faites. Un spectre est un grand vecteur d'entrée duquel il peut être difficile d'extraire l'information. Ces expérimentations nous permettent

de nous assurer qu'il n'y a pas de perte majeure d'informations de par les transformations évoquées précédemment, mais également de vérifier si elles sont utiles ou nécessaires.

5.3 Représentation et paramétrage des données

Qu'il s'agisse du HPCP, MFCC, GFCC, ou du spectre, les procédures de prétraitement requièrent un paramétrage supplémentaire préalable concernant l'échantillonnage et le découpage de trames. Plusieurs valeurs courantes existent dans le domaine du traitement audio, présentées au Tableau 5.3.

Taux
8 000 Hz
11 025 Hz
16 000 Hz
32 000 Hz
44 100 Hz
48 000 Hz

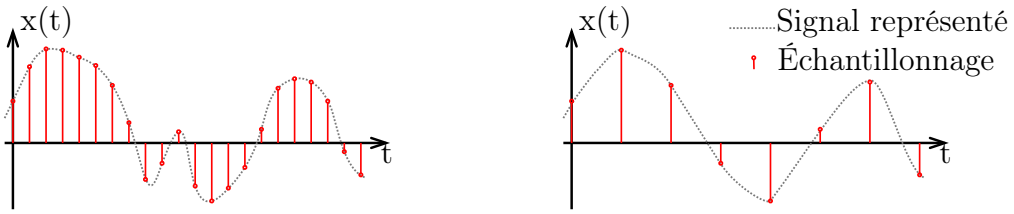
(a) Taux usuels.

Taille des trames	Sauts de trames		
128	128	64	32
256	256	128	64
512	512	256	128
1 024	1 024	512	256
2 048	2 048	1 024	512
4 096	4 096	2 048	1 024

(b) Découpages usuels.

Tableau 5.3: Synthèse des possibilités de configuration de lecture et découpage de signaux.

Les tailles de trames définissent la largeur des sections qui sont extraites du signal. Chaque taille exprime le nombre de valeurs qui y sont prélevées. La durée du signal contenu dans une trame est donc proportionnelle au taux d'échantillonage du signal d'origine.

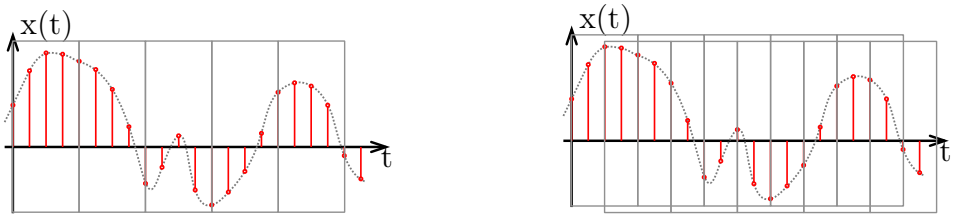


(a) Taux $3 \times x$.

(b) Taux $1 \times x$.

Figure 5.1: Variations d'échantillonnage d'un signal de même durée.

Les sauts de trame correspondent au décalage entre le début d'une trame et le début de la trame suivante ; par nature il doit être strictement positif et inférieur ou égal à la taille de trame.



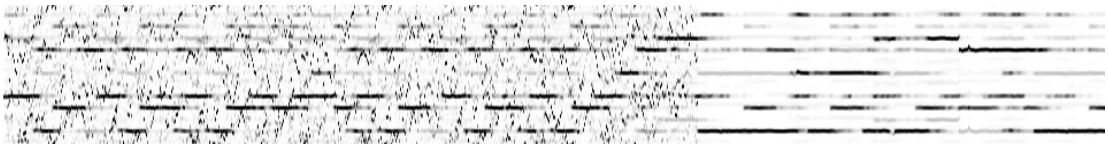
(a) Taille de trame et dimension de saut égales.

(b) Saut inférieur de moitié à la taille de trame.

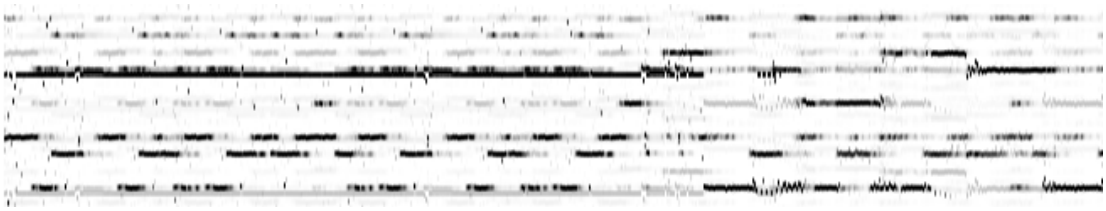
Figure 5.2: Variation de découpage et sauts de trames [16].

Ces paramètres sont importants car ils peuvent significativement changer la représentation que les données auront lors de leur utilisation en vecteur d'entrée.

16. Découpage par taille de trame 4 et saut de 4 sur 5.2a ; et taille de trame 4 avec saut de 2 sur 5.2b.



(a) Échantillonage 16 000 Hz, trames de 256, et sauts de 256.



(b) Échantillonage 44 100 Hz, trames de 2 048, et sauts de 1 024.

Figure 5.3: Variations de la représentation par le descripteur HPCP d'un même signal selon le paramétrage de lecture et découpage.

Concernant les vecteurs obtenus, il est nécessaire de les cumuler. Un instant musical est défini par sa trame instantanée ainsi que par celles la précédant. Le cas contraire reviendrait à donner la dernière syllabe d'un mot et vouloir en obtenir le sens.

Nous avons déterminé au moyen d'expérimentations préalables que l'accumulation de 10 trames est viable. Davantage complexifie l'interprétation du vecteur d'entrée, et moins ne permet pas aux algorithmes d'extraire les classes recherchées. De même, il ressort que les paramètres de prétraitement les plus satisfaisants sont un taux d'échantillonage de 44 100 Hz, une taille de trame à 4 096, et un saut entre trames de 2 048.

5.4 Constitution des architectures

Nous avons constitué quatre types principaux d'architectures de réseaux de neurones, évoluant depuis une base commune (Figure 5.4) :

(a) des réseaux de neurones pleinement connectés à propagation avant à taille

constante ;

(b) des réseaux de neurones pleinement connectés à propagation avant à taille décroissante ;

(c) des auto-encodeurs éparses suivis de réseaux de neurones pleinement connectés à propagation avant à taille constante ;

(d) des auto-encodeurs éparses suivis de réseaux de neurones pleinement connectés à propagation avant à taille décroissante.

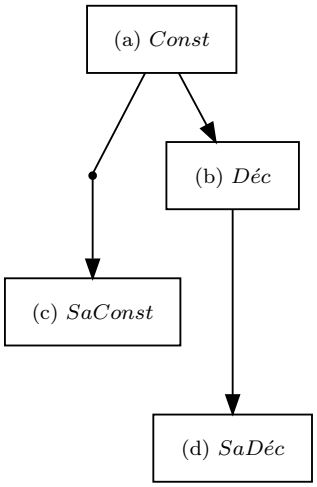
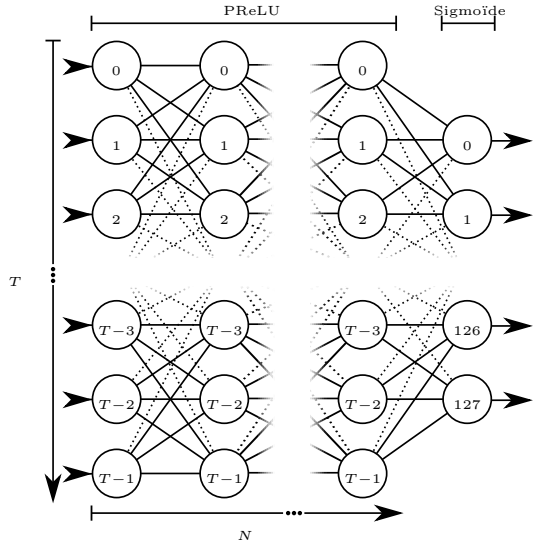


Figure 5.4: Types d'architectures d'expérimentations.

La taille de la couche d'entrée de toute expérimentation dépend des descripteurs utilisés. Les dimensions et configurations des auto-encodeurs et couches intermédiaires dépendent du type d'architecture et de l'expérimentation effectuée. Tous les réseaux de neurones à propagation avant décrits ici exploitent l'unité linéaire rectifiée paramétrique comme fonction d'activation (*Parametric Rectified Linear Unit*, PReLU). À l'extrémité finale s'ajoute une couche supplémentaire de même taille que la pénultième, reposant sur la fonction sigmoïde. La couche de sortie est systématiquement de taille 128, pour chacune des possibilités du MIDI. Cette constance est appliquée dans le soucis d'établir une méthode générique pour tout

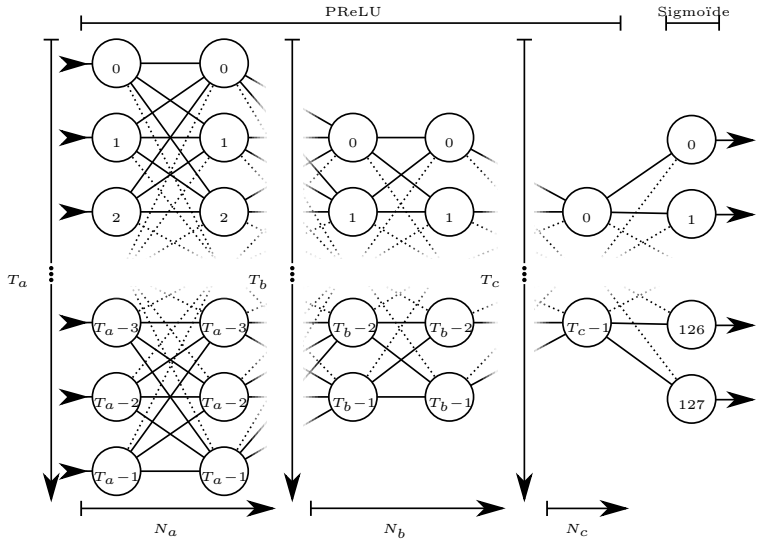
instrument, même si aucun instrument n'a de tessiture comportant ces 128 notes.

Tel que présenté par la Figure 5.5, les architectures à taille constante (a) sont constituées de N couches de taille T fixe pleinement connectées. Les architectures à taille décroissante (b) sont constituées de N couches pleinement connectées dont la taille T_N décroît en proportion propre à chaque expérimentation. Les architectures comportant un auto-encodeur (c) et (d) utilisent un auto-encodeur éparse (*Sparse Autoencoder*, SA) ayant T_E comme taille de la couche d'entrée, et T_I comme taille de couche intermédiaire.

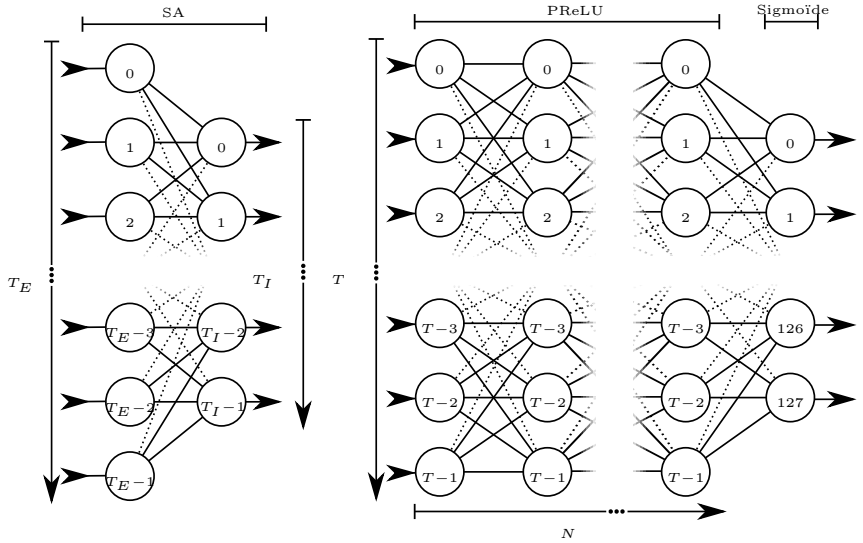


(a) Taille constante.

Figure 5.5: Schéma des architectures.

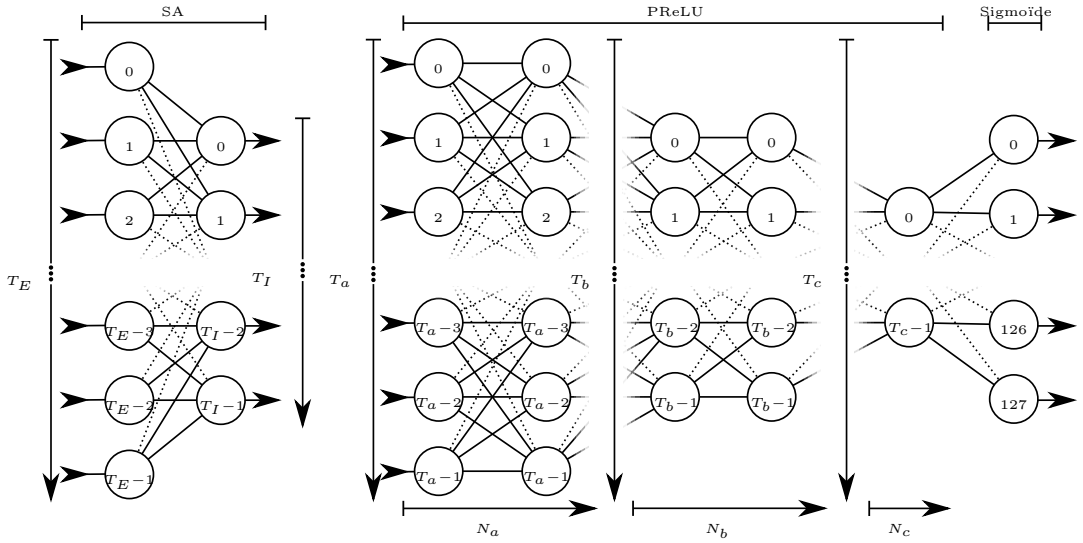


(b) Taille décroissante.



(c) Auto-encodeur et taille constante.

Figure 5.5: Schéma des architectures (cont.)



(d) Auto-encodeur et taille décroissante.

Figure 5.5: Schéma des architectures (cont.)

5.5 Modèles expérimentaux

Comme décrit en section 5.2, nous utilisons comme descripteurs soit HPCP seul, noté H ; soit HPCP, MFCC, et GFCC combinés, noté HMG ; soit HPCP et le spectre combinés, noté HS. En nous basant sur les architectures montrées précédemment, nous avons établi cinquante modèles expérimentaux, incluant les variations, afin de résoudre notre problématique de détection des notes. À ces modèles expérimentaux sont appliqués distinctement les quinze jeux de données décrits en section 5.1, ce qui nous porte à un total de sept-cent-cinquante expérimentations. L'objectif est non seulement de déterminer une architecture efficace selon le niveau de détection souhaité, mais également d'effectuer un comparatif des performances et compromis raisonnables. Nous voulons aussi mettre en évidence des affinités entre certaines architectures et des instruments et cas applicatifs.

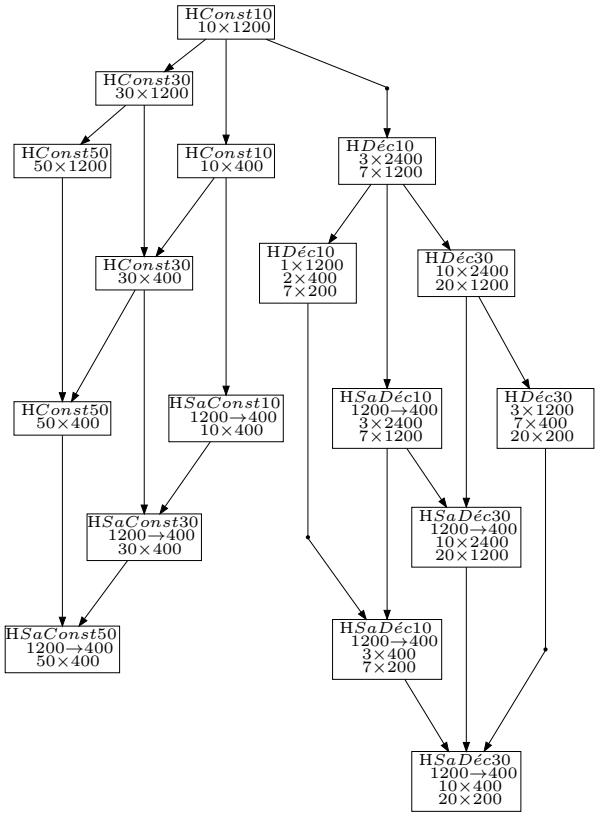


Figure 5.6: Hiérarchie des modèles expérimentaux H et partage de propriétés.

Les modèles expérimentaux sont détaillés dans la Figure 5.6 pour le descripteur H, la Figure 5.7 pour les descripteurs HS, et la Figure 5.8 pour les descripteurs HMG. La notation $E \rightarrow I$ décrit les auto-encodeurs, et $N \times T$ les nombres de couches et tailles associées, par ordre de progression descendant (la couche de sortie systématique de taille 128 est omise).

Nous établissons des modèles expérimentaux où les réseaux sont dimensionnés à 10, 30, et 50 couches ; taillés sur une largeur égale à leur vecteur d'entrée, ou contraints à des tailles inférieures. Les architectures en décroissance sont établies avec plusieurs degrés de contraintes de taille. Les auto-encodeurs éparses sont entraînés pour reproduire leur vecteur d'entrée, avec comme valeur de régularisation $L2\ \lambda = 0,0001$; de divergence $KL\ \beta = 3$; et densité $\rho = 0,01$. Les auto-encodeurs

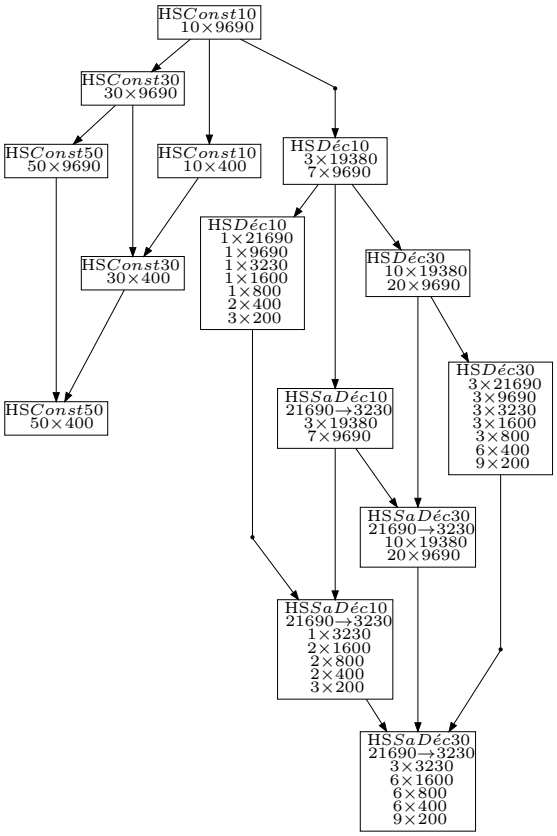


Figure 5.7: Hiérarchie des modèles expérimentaux HS et partage de propriétés.

éparses sont chaînés à des réseaux également évalués distinctement et solitairement par d'autres expérimentations, pour faire un comparatif. Enfin, pour HMG (Figure 5.8), en plus d'un auto-encodeur prenant l'intégralité du vecteur d'entrée, nous combinons trois auto-encodeurs en parallèle correspondant distinctement à un des trois descripteurs ($3Sa$).

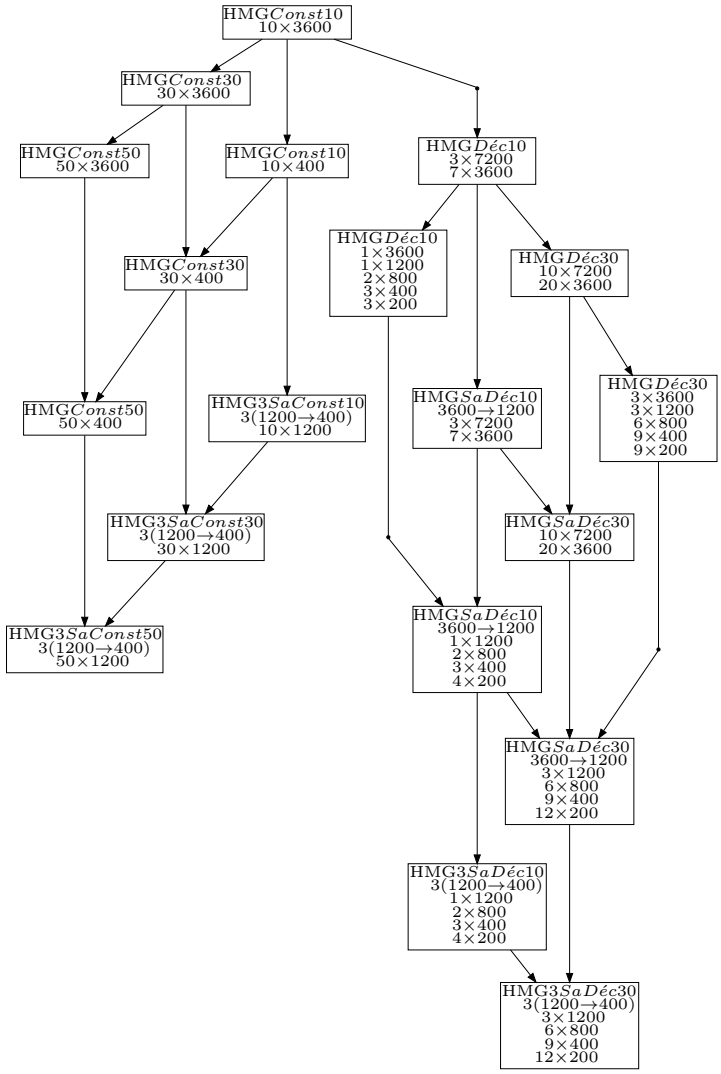


Figure 5.8: Hiérarchie des modèles expérimentaux HMG et partage de propriétés.

5.6 Évaluation et exécution

Évoluant dans un problème de classification à labels multiples (un par note), l'évaluation repose essentiellement sur le score F1 englobant tous les labels-notes. Nous effectuons aussi une évaluation binaire relative à chaque note afin de s'assurer qu'une expérimentation ne souffre pas d'une déficience localisée à certaines parties de la tessiture, et qui pourrait être masquée dans le score global. Une évaluation

est réalisée à chaque fin d'époque d'apprentissage.

Les expérimentations sont exécutées grâce à Calcul Québec [17] et Calcul Canada [18], sur le calculateur Béluga de l'École de Technologie Supérieure [19] (ÉTS). Chaque roulement est effectué sur quatre cœurs de processeur Intel Gold 6148 Skylake cadencés à 2,40 GHz. Les durées d'exécution varient de quelques heures à sept jours, la mémoire de moins de 4 Go à 100 Go. Les ressources nécessaires aux différentes expérimentations, fortement corrélées à la taille du jeu de donnée exploité, sont détaillées en Annexe D. En tout et pour tout, un peu plus de 17 To d'enregistrement de données ont été requis, en comptant les jeux de données, les configurations, et les résultats d'évaluation.

17. Site officiel à `https://www.calculquebec.ca/`

18. Site officiel à `https://www.calculcanada.ca/`

19. Site officiel à `https://www.etsmtl.ca/`

CHAPITRE VI

RÉSULTATS ET ANALYSES DU COMPORTEMENT DES ARCHITECTURES ET MODÈLES D'APPRENTISSAGE MACHINE

Nos analyses montrent des résultats très hétérogènes parmi les expérimentations effectuées. Certaines sont en échec plat lorsque d'autres exposent un fort potentiel.

6.1 Échecs d'apprentissage

De tous les résultats expérimentaux, nous extrayons d'abord deux constats d'échecs d'apprentissage. Le premier porte sur les expérimentations reposant sur le spectre (HS), le second sur les architectures à taille décroissante (*Déc*), qu'elles impliquent ou non un auto-encodeur.

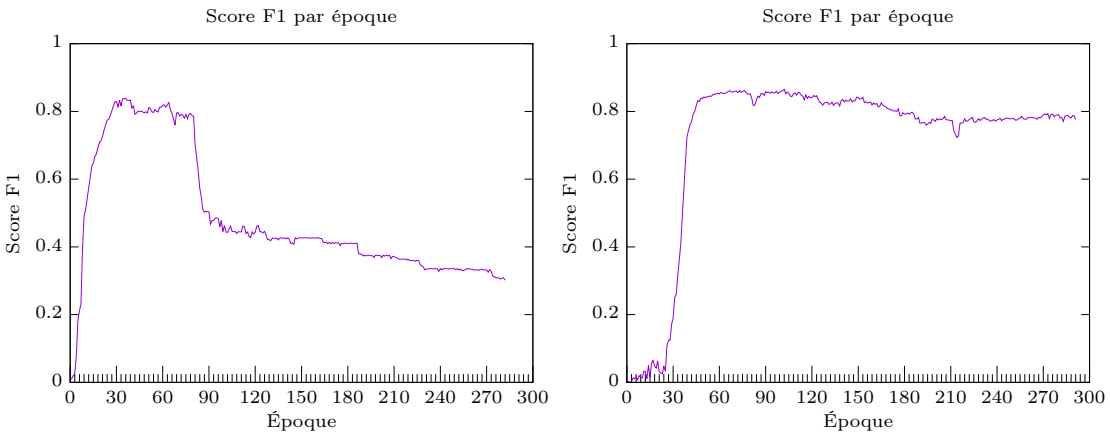
Le premier cas étaye notre postulat initial concernant la difficulté d'extraction de l'information du spectre, même lorsqu'il est combiné au HPCP. Les auto-encodeurs ne réussissent pas non plus à modéliser le spectre. Les architectures et dimensions appliquées dans nos expérimentations sont ainsi inadéquates à l'exploitation du descripteur spectral.

Le second cas tend à montrer l'inadéquation d'un réseau en décroissance à l'extraction de notes depuis chacun des descripteurs utilisés. L'explication plausible est que les descripteurs sont résolus tardivement dans les couches intermédiaires,

donc que la réduction de leur taille depuis l'amont n'est pas souhaitable.

6.2 Architectures fructueuses

Les expérimentations H et HMG montrent des résultats bien plus intéressants. Tout d'abord, les réseaux de dimension 10 à taille constante (*Const*10) parviennent dans l'ensemble à effectuer un apprentissage sur ces descripteurs dans un cadre monophonique mono-instrumental, à savoir sur les jeux issus de Fluid R3 GM (sous-section 5.1.1) et le jeu *Isolé* de MAPS (sous-section 5.1.2).



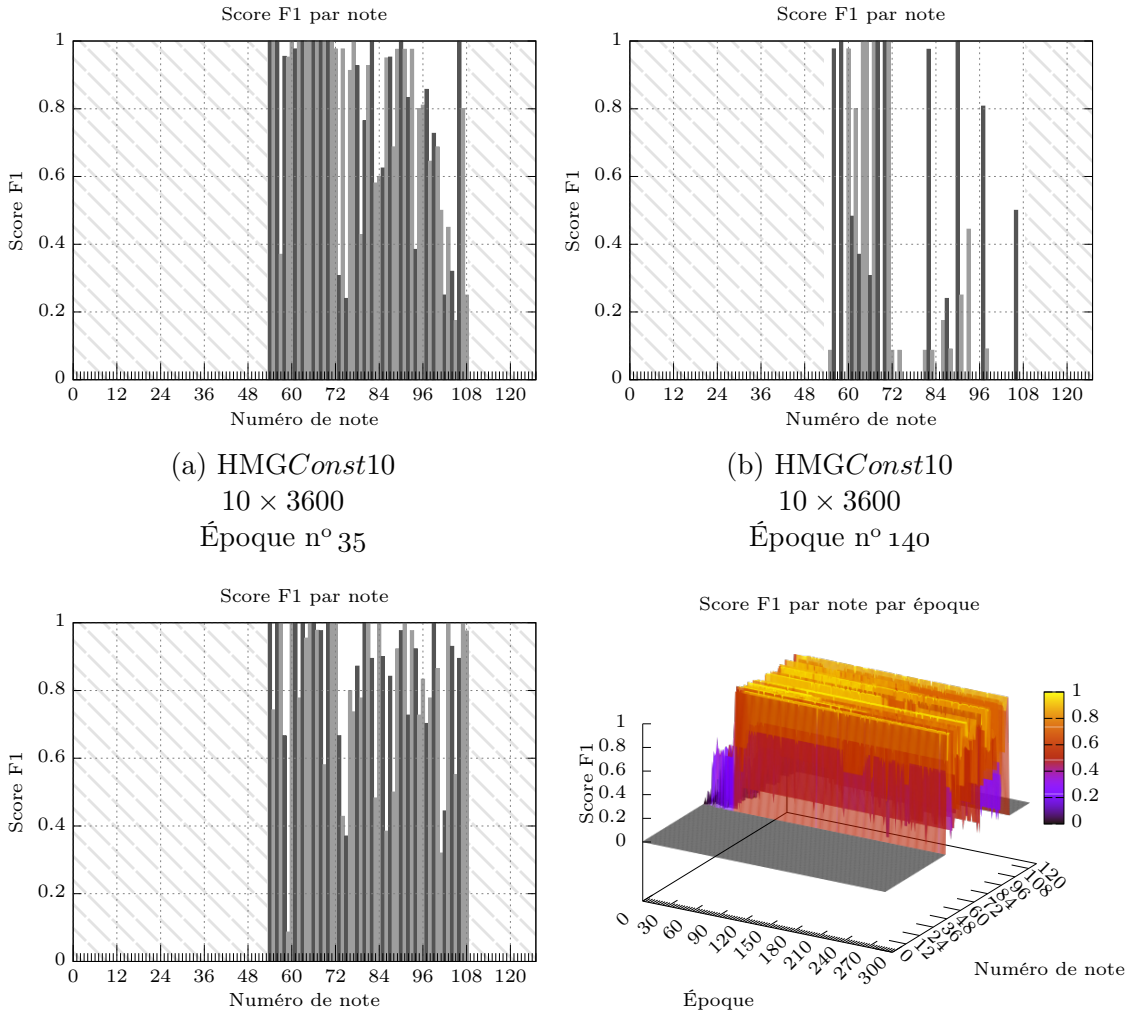
(a) HMG*Const*10
10×3600

(b) HMG*Const*10
10×400

Figure 6.1: Apprentissage monophonique sur l'instrument *013 Xylophone*.

Ensuite, on constate une atténuation significative du sur-apprentissage lorsque le réseau est contraint à une taille inférieure au vecteur d'entrée, comme illustré en Figure 6.1. On peut également constater en Figure 6.2 les variations d'apprentissage entre les différentes notes de la tessiture. Ces analyses rendent comptent du déclin constaté en Figure 6.1a entre les époques n° 35 (Figure 6.2a) et n° 140

(Figure 6.2b). Nous constatons également qu'un bon score global peut masquer des lacunes localisées à certaines notes (Figure 6.2c), même si le jeu de données contient chaque note de façon équivalente (sous-section 5.1.1). De même, il n'y a pas de changement significatif dans le score individuel des labels attribués au fil des époques (Figure 6.2d).



(a) HMG*Const*10
10×3600
Époque n° 35

(b) HMG*Const*10
10×3600
Époque n° 140

(c) HMG*Const*10
10×400
Époque n° 140

(d) HMG*Const*10
10×400
Séquence totale

Figure 6.2: Détail d'époques d'apprentissage monophonique sur l'instrument *013 Xylophone*.

6.3 Succès des auto-encodeurs

L'usage d'auto-encodeur se révèle extrêmement efficace et améliore drastiquement les performances d'apprentissage avec le descripteur HPCP. Ainsi, dans un cadre monophonique mono-instrumental, H$SaConst10(1200 \rightarrow 400; 10 \times 400)$ est en capacité d'apprentissage générique sur la majorité des instruments. Le détail des résultats est disponible en Annexe E.1.

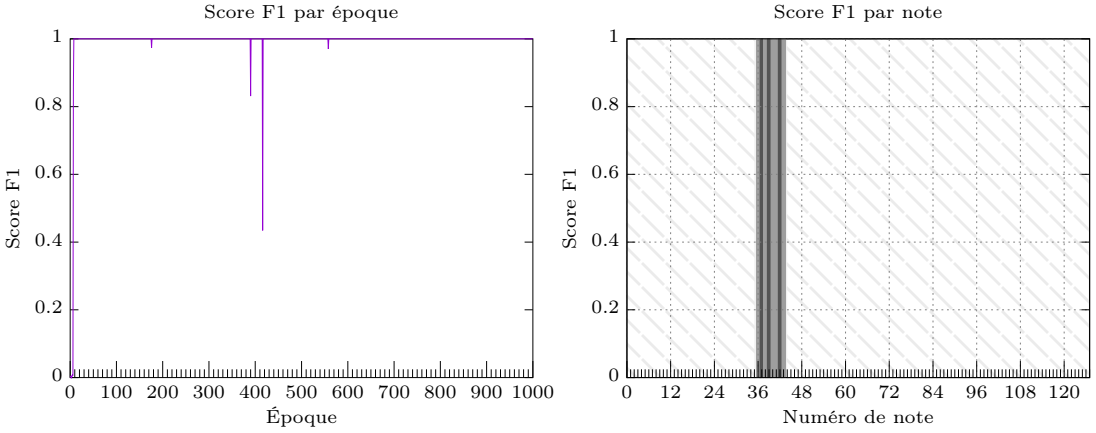
Numéro	Instrument	Score F1
000	*Piano à queue*	0,99
013	*Xylophone*	0,95
019	*Orgue d'église*	0,00
024	*Guitare classique*	0,97
032	*Basse acoustique*	0,99
040	*Violon*	0,98
048	*Ensemble de cordes acoustiques*	1,00
057	*Trombone*	0,98
064	*Saxophone soprano*	0,99
075	*Flûte de pan*	0,99
080	*Signal carré*	1,00

Tableau 6.1: Scores d'apprentissage monophonique mono-instrumental du modèle H$SaConst10(1200 \rightarrow 400; 10 \times 400)$ à l'époque n° 50.

Les scores exposés au Tableau 6.1 montrent de très bons résultats d'apprentissage. Ces scores ne signifient pas un sur-apprentissage du modèle, car dans un cadre monophonique mono-instrumental, le descripteur HPCP est proche d'être déterministe en fonction de la note jouée. Le modèle est alors capable d'effectuer l'analyse des classes harmoniques de façon très performante. La Figure E.2 disponible en annexe expose le détail des scores de chaque note à l'époque n° 50.

Nous remarquons que les instruments *048 Ensemble de cordes acoustiques* et *080 Signal carré* sont au score 1. Cela s'explique pour le premier par la faible étendue

de sa tessiture (36 à 43, soit do1 à sol1), Figure 6.3 ; et pour le second par le caractère « synthétique » de son timbre, qui est totalement déterministe, Figure 6.4. Les tessitures sont spécifiées au Tableau 5.1.

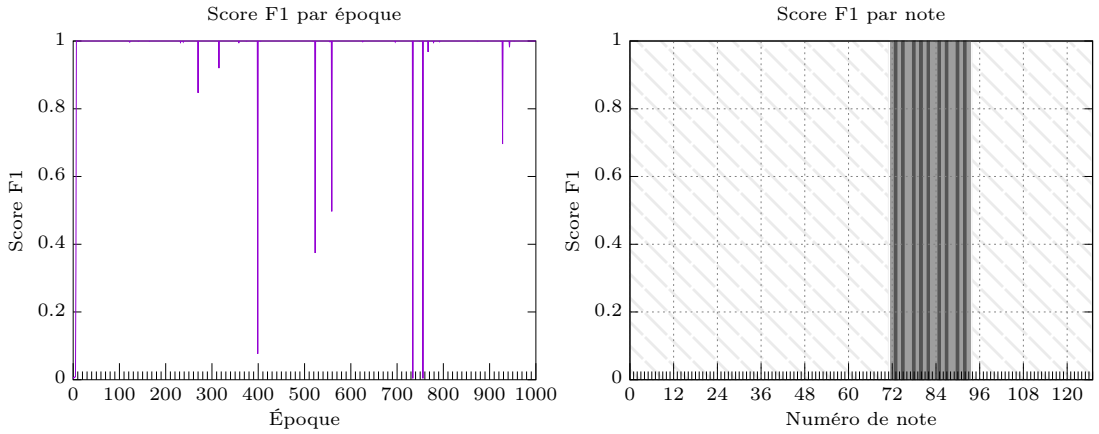


(a) H*SaConst*10
1200 → 400
10 × 400
Global

(b) H*SaConst*10
1200 → 400
10 × 400
Époque n° 50

Figure 6.3: Apprentissage monophonique sur l'instrument *048 Ensemble de cordes acoustiques*.

Aussi, un échec d'apprentissage pour H*SaConst*10(1200 → 400; 10 × 400) est à relever concernant l'instrument *019 Orgue d'église*, Figure 6.5a. L'explication est qu'un orgue ne peut pas strictement correspondre au cadre monophonique mono-instrumental, puisque chaque tuyau qui le compose a un timbre propre. Un orgue s'assimile ainsi davantage à un cas monophonique multi-instrumental. De façon plus globale, aucune architecture contenant un auto-encodeur n'est parvenue à apprendre sur cet instrument. Les expérimentations HMG*Const*10(10 × 400) et H*Const*10(10 × 400), qui n'ont pas d'auto-encodeurs, montrent une difficulté particulière d'apprentissage de cet instrument, comme illustré en Figures 6.5b et 6.5c.



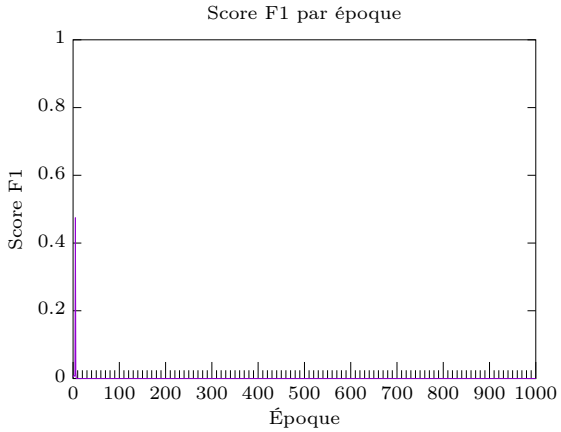
(a) H*SaConst*10
$1200 \to 400$
10×400
Global

(b) H*SaConst*10
$1200 \to 400$
10×400
Époque n° 50

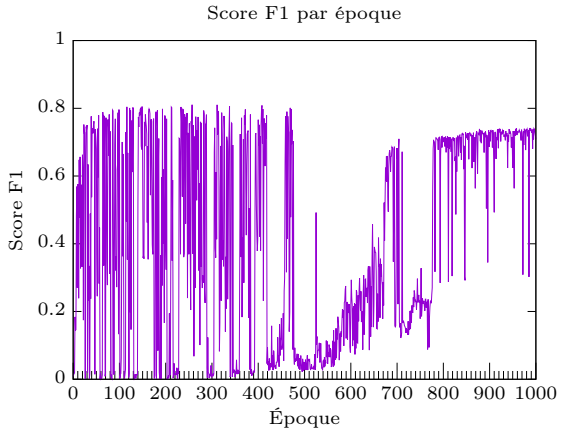
Figure 6.4: Apprentissage monophonique sur l'instrument *080 Signal carré*.

6.4 Apprentissage polyphonique

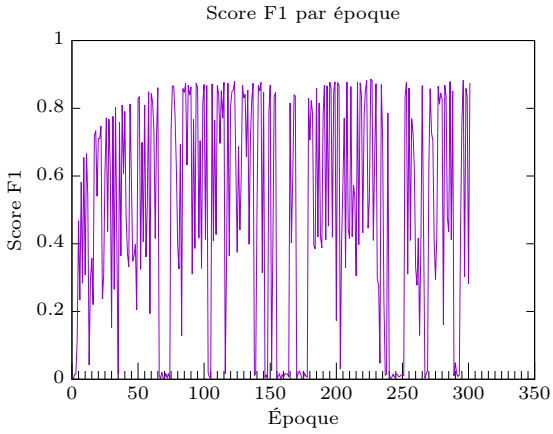
Nous constatons une capacité d'apprentissage de H*SaConst*10$(1200 \to 400; 10 \times 400)$ dans un cadre polyphonique mono-instrumental. Sur le jeu *Musiques* de MAPS un score F1 allant jusqu'à 0,85 est relevé, illustré en Figure 6.6a. Si l'apprentissage est davantage variable au fil des époques qu'en situation monophonique mono-instrumental (Figure 6.6b), on remarque que le modèle ne varie pas significativement en performance sur des notes précises lorsque l'on sélectionne des époques ayant de bons scores, comme entre l'époque n° 30 (Figure 6.6c) de score 0,84, et l'époque n° 221 (Figure 6.6d) de score 0,85. La forme en cloche que prend le détail d'apprentissage des différentes notes est dû à la distribution du jeu de données *Musiques*, contenant davantage de notes des octaves 2 et 3 (48 à 72), autour desquels sont composées la plupart des œuvres au piano classiques.



(a) H$SaConst10$
$1200 \rightarrow 400$
$10 \times \ 400$

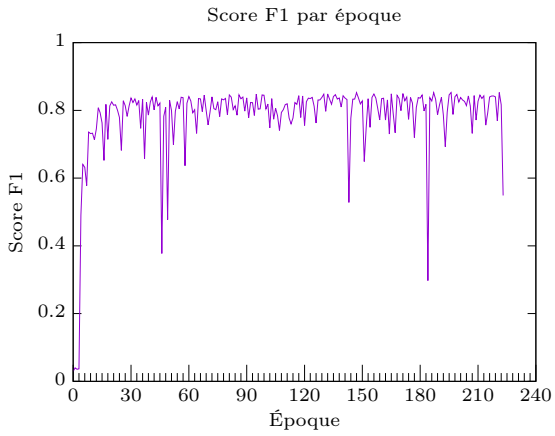


(b) H$Const10$
10×400

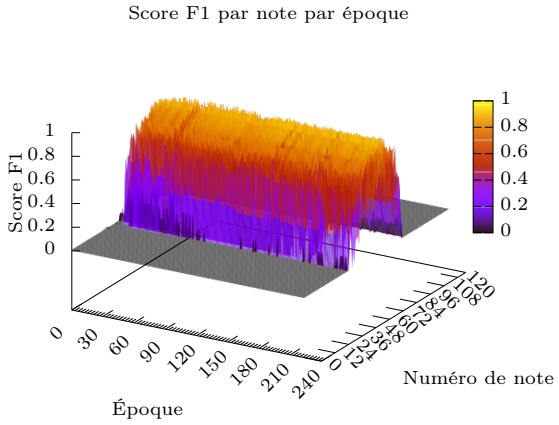


(c) HMG$Const10$
10×400

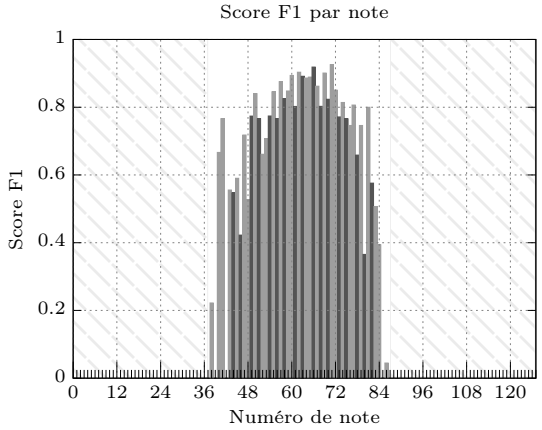
Figure 6.5: Apprentissage monophonique sur l'instrument *019 Orgue d'église*.



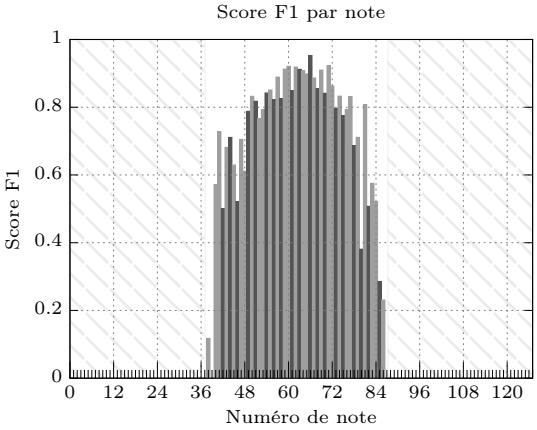
(a) H*SaConst*10
1200 → 400
10 × 400
Global



(b) H*SaConst*10
1200 → 400
10 × 400
Séquence totale



(c) H*SaConst*10
1200 → 400
10 × 400
Époque n° 30



(d) H*SaConst*10
1200 → 400
10 × 400
Époque n° 221

Figure 6.6: Apprentissage polyphonique sur le jeu *Musiques*.

CHAPITRE VII

PROTOCOLE DE CRÉATION DE MODÈLE DE TRANSCRIPTION MONOPHONIQUE MONO-INSTRUMENTAL

Nous proposons dans ce chapitre un protocole permettant pour un instrument donné d'obtenir un modèle de transcription automatique de la musique dans un cadre monophonique mono-instrumental. Ce protocole se découpe en quatre parties : (1) la sélection ou l'établissement des données requises ; (2) le traitement audio préalable et l'application de descripteur de signal ; (3) l'architecture du modèle de classification ; et enfin (4) son entraînement. Ce protocole est une généralisation de l'expérimentation H*SaConst* examinée en Section 6.3.

7.1 Données requises

Un instrument est préalablement choisi, pour lequel il est nécessaire de disposer d'au moins un enregistrement de chaque note de la tessiture. Il est possible d'accumuler des enregistrements provenant de différents instruments physiques du même type, ou bien d'une même famille instrumentale, afin de construire un modèle en capacité de reconnaître une plus grande diversité de timbres. Ces enregistrements doivent comporter :

— un silence préalable au jeu de chaque note, même minime, afin de capturer l'attaque en intégralité ;

— comporter la phase d'attaque, de maintien, et de déclin, s'il y a lieu ;
— poursuivre sur un silence équivalent au temps où la note est musicalement considérée comme jouée, ou d'au moins une seconde après que le seuil d'audition soit passé.

À chaque enregistrement doit être couplé l'information musicale de jeu synchronisée. Soit on dispose d'une base de données comportant les échantillons sonores couplés à l'information musicale, tel que MAPS pour le piano ; soit on en constitue une par l'enregistrement physique sonore et musical du jeu de l'instrument sélectionné. On peut également faire usage d'une fonte sonore et d'un synthétiseur, pour lequel on aura préparé une piste MIDI permettant de correspondre aux critères évoqués ici.

7.2 Traitement audio

Afin de préparer les données audio et d'en extraire le descripteur HPCP, une séquence de prétraitement doit être appliquée aux enregistrements. La Figure 7.1 synthétise les informations de paramétrage, les spécifications précises d'implémentation avec Mélodium sont disponibles en annexe C.4.

Le traitement audio à effectuer sur les enregistrements est tel que :

1. le chargement des fichiers est effectué en monocanal 44 100 Hz ;
2. le découpage de trames audio est effectué avec :
   — une taille de trame de 4 096 ;
   — un saut de trame de 2 048 ;
3. le fenêtrage est effectué avec une fenêtre de Blackman-Harris 92 dB ;
4. le spectre est calculé ;
5. les pics spectraux sont extraits avec :
   — une fréquence minimale de 40 Hz ;

— une fréquence maximale de 5 000 Hz ;

— un ordonnancement par magnitude ;

6. le HPCP est calculé avec :
   — une considération des 8 premières harmoniques ;
   — une fréquence de référence fixée au la_3 440 Hz ;
   — une pondération par cosinus ;
   — une sortie normalisée, et dimensionnée à 120 ;
7. une accumulation de chaque trame avec les 9 précédentes, soit un cumul total de 10.

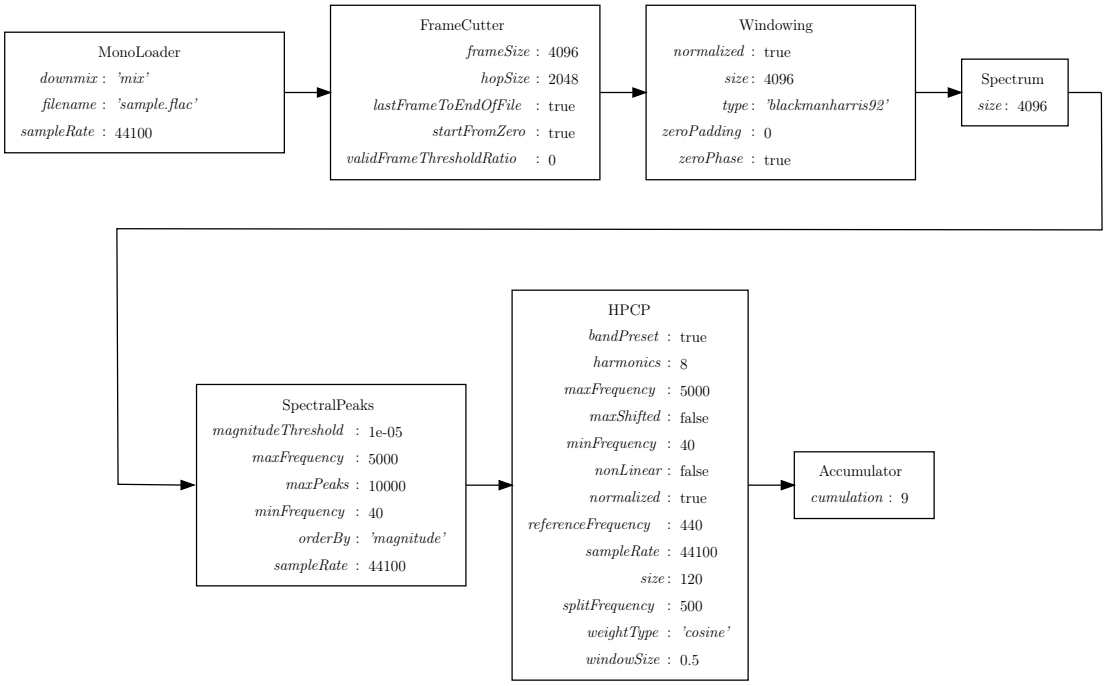


Figure 7.1: Prétraitements pour la transcription monophonique.

7.3 Architecture

Le modèle d'apprentissage machine correspond strictement à celui utilisé pour l'expérimentation H$SaConst10(1200 \to 400; 10 \times 400)$ décrite au Chapitre 5. Son architecture, illustrée en Figure 7.2, se décrit ainsi :

1. un auto-encodeur éparse paramétré tel qu'ayant :
   — une taille de la couche visible à 1 200 ;
   — une taille de la couche intermédiaire à 400.
2. un réseau de neurones à propagation avant pleinement connecté paramétré tel que :
   — il dispose de 10 couches intermédiaires ;
   — chaque couche intermédiaire est dimensionnée à 400 neurones ;
   — les neurones des couches intermédiaires reposent sur la fonction d'activation PReLU ;
   — la couche de sortie est dimensionnée à 128 neurones ;
   — les neurones de la couche de sortie reposent sur la fonction sigmoïde.

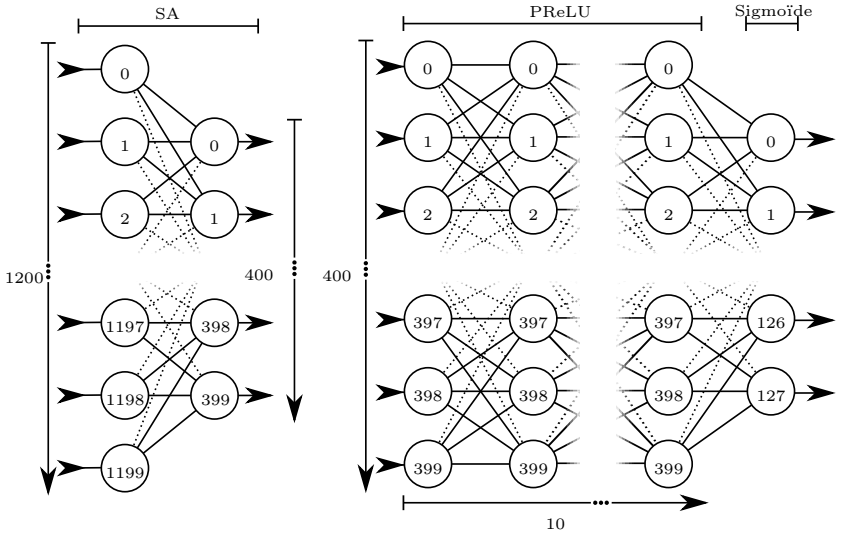


Figure 7.2: Architecture du modèle de transcription automatique monophonique.

7.4 Apprentissage

L'auto-encodeur est préalablement entraîné sur l'ensemble des enregistrements traités tel que décrit en section 7.2, de manière à reproduire le plus fidèlement

les données en entrée ; avec comme valeur de régularisation $L2\ \lambda = 0,0001$, de divergence $KL\ \beta = 3$, et densité $\rho = 0,01$. Le réseau de neurones à propagation avant est ensuite entraîné sur les données de sortie de l'auto-encodeur, avec comme table de vérité les annotations musicales d'origine du signal donné présentement à l'auto-encodeur. Une cinquantaine d'époques d'apprentissage sont nécessaires pour l'obtention d'un modèle fonctionnel.

Un tel modèle est alors capable d'effectuer une transcription automatique de la musique dans un cadre monophonique pour l'instrument sélectionné, avec un score F1 supposé dépasser 0,9. Il est à noter que l'on peut utiliser cette même démarche en utilisant non pas des données monophoniques mais polyphoniques pour parvenir à un modèle de transcription automatique polyphonique. Cependant, le score F1 pourrait ne pas être aussi bon, ou bien le nombre d'époques nécessaires différer pour obtenir un apprentissage satisfaisant. L'état actuel de nos recherches ne permet pas de garantir le succès de ce modèle pour le cadre polyphonique dans un cas autre que celui du piano.

CONCLUSION

Dans ces travaux, nous avons présenté les tenants et des aboutissants du domaine de la transcription automatique de la musique. Nous avons proposé une échelle de progression dans la complexité du problème auquel nous avons affaire. Composée de cinq niveaux, elle permet de classer les travaux existants, et de repérer à quel niveau d'avancement un modèle se situe.

Ayant constaté le manque d'outillage dédié à ce champ de recherche, nous avons créé un logiciel nommé Mélodium, couplé à un langage de script permettant de l'exploiter. Ce logiciel a été conçu en ayant comme objet le traitement automatique et efficace de longs vecteurs de données représentant des signaux audio et des données musicales, avec une forte résilience aux erreurs. Le langage associé vise à faciliter la manipulation des données et la réalisation d'expérimentations, en se concentrant sur les traitements mis à disposition par notre logiciel, sans requérir de l'utilisateur une connaissance technique poussée.

En nous appuyant sur les travaux existants, et en combinant des approches cognitives et d'apprentissage machine, nous avons mené à bien diverses expérimentations au moyen de l'outil nouvellement créé. Un aspect spécifique était d'utiliser des fontes sonores, largement répandues dans le monde de l'audio et de la musique, pour apprendre les caractéristiques sonores des instruments. En combinant des auto-encodeurs éparses et des réseaux de neurones à propagation avant, nous avons pu arriver à des résultats fructueux. Nous avons pu établir une méthode générique pour répondre avec la plupart des instruments au cas monophonique mono-instrumental, premier niveau de notre échelle ; et disposons d'un modèle

satisfaisant pour la polyphonie mono-instrumentale appliquée au piano.

Conformément à l'échelle proposée, une grande marge de progression existe dans la continuité de ces travaux. L'outil Mélodium est appelé à évoluer, d'autres expérimentations sont vouées à être réalisées, et l'atteinte d'un modèle de transcription automatique distinguant de grands ensembles d'instruments avec fiabilité semble atteignable.

APPENDICE A

LISTE DES TRAITEMENTS DISPONIBLES DANS MÉLODIUM

Le tableau A.1 expose la liste des 197 traitements disponibles dans Mélodium au moment de la rédaction de ce mémoire. Leur fonctionnement ainsi que les rôles associés sont décrits en 7.3. Un part significative d'entre eux proviennent ou sont adaptés de la bibliothèque Essentia (Bogdanov *et al.*, 2013), constituant la majeure partie des traitements de rôle *analyste*. L'obtention de la documentation ainsi qu'un exemple sont donnés en annexe B.

Nom	Rôle
Accumulator	Analyste
AfterMaxToBeforeMaxEnergyRatio	Analyste
AllPass	Analyste
AudioOnsetsMarker	Analyste
AutoCorrelation	Analyste
BandPass	Analyste
BandReject	Analyste
BarkBands	Analyste
Beatogram	Analyste
BeatsLoudness	Analyste
BeatTrackerDegara	Analyste
BeatTrackerMultiFeature	Analyste
BinaryOperator	Analyste
BinaryOperatorStream	Analyste
BPF	Analyste
BpmHistogramDescriptors	Analyste
CentralMoments	Analyste
Centroid	Analyste
ChordsDescriptors	Analyste
ChordsDetection	Analyste
ChordsDetectionBeats	Analyste
Clipper	Analyste
Concatenator	Analyste
Crest	Analyste
CrossCorrelation	Analyste
CsvExporter	Exporteur de données
CubicSpline	Analyste
Danceability	Analyste
DCRemoval	Analyste
DCT	Analyste
Decrease	Analyste
Derivative	Analyste
DerivativeSFX	Analyste
Differentiator	Analyste
Dissonance	Analyste
DistributionShape	Analyste
Duration	Analyste
DynamicComplexity	Analyste
EasyLoader	Analyste
EffectiveDuration	Analyste
Energy	Analyste
EnergyBand	Analyste
EnergyBandRatio	Analyste
Entropy	Analyste
Envelope	Analyste
EqloudLoader	Analyste
EqualLoudness	Analyste
ERBBands	Analyste
Evaluator	Analyste
FfmpegConverter	Converter

Nom	Rôle
Flatness	Analyste
FlatnessDB	Analyste
FlatnessSFX	Analyste
FluidsynthRender	Rendu audio
Flux	Analyste
FrameCutter	Analyste
FrameToReal	Analyste
FrequencyBands	Analyste
GeometricMean	Analyste
GFCC	Analyste
HarmonicBpm	Analyste
HarmonicPeaks	Analyste
HFC	Analyste
HighPass	Analyste
HighResolutionFeatures	Analyste
HPCP	Analyste
HprModelAnal	Analyste
HpsModelAnal	Analyste
IIR	Analyste
ImageExporter	Exporteur de données
Inharmonicity	Analyste
InstantPower	Analyste
JsonExporter	Exporteur de données
Key	Analyste
KeyExtractor	Analyste
Larm	Analyste
Leq	Analyste
LevelExtractor	Analyste
LogAttackTime	Analyste
LoopBpmConfidence	Analyste
LoopBpmEstimator	Analyste
Loudness	Analyste
LoudnessVickers	Analyste
LowLevelSpectralEqloudExtractor	Analyste
LowLevelSpectralExtractor	Analyste
LowPass	Analyste
LPC	Analyste
MaxFilter	Analyste
MaxMagFreq	Analyste
MaxToTotal	Analyste
Mean	Analyste
Median	Analyste
MelBands	Analyste
Meter	Analyste
MFCC	Analyste
MidiExporter	Exporteur de données
MidiTracksAnalyst	Analyste
MidiTracksFiller	Analyste
MidiVoicesAnalyst	Analyste
MinToTotal	Analyste

Tableau A.1: Liste des traitements disponibles dans Mélodium.

Nom	Rôle
MlpackModelFeeder	Alimenteur de modèle
MlpackModelLoader	Chargeur de modèle
MlpackModelPredictor	Prédicteur de modèle
MlpackModelSaver	Enregistreur de modèle
MlpackModelTrainer	Entraîneur de modèle
MonoLoader	Analyste
MonoWriter	Analyste
MovingAverage	Analyste
MultiPitchKlapuri	Analyste
MultiPitchMelodia	Analyste
Multiplexer	Analyste
NoiseAdder	Analyste
NoveltyCurve	Analyste
NoveltyCurveFixedBpmEstimator	Analyste
OddToEvenHarmonicEnergyRatio	Analyste
OnsetDetection	Analyste
OnsetDetectionGlobal	Analyste
OnsetRate	Analyste
OverlapAdd	Analyste
PeakDetection	Analyste
PercivalBpmEstimator	Analyste
PercivalEnhanceHarmonics	Analyste
PercivalEvaluatePulseTrains	Analyste
PitchContours	Analyste
PitchContourSegmentation	Analyste
PitchContoursMelody	Analyste
PitchContoursMonoMelody	Analyste
PitchContoursMultiMelody	Analyste
PitchFilter	Analyste
PitchMelodia	Analyste
PitchSalience	Analyste
PitchSalienceFunction	Analyste
PitchSalienceFunctionPeaks	Analyste
PitchYin	Analyste
PitchYinFFT	Analyste
PowerMean	Analyste
PowerSpectrum	Analyste
PredominantPitchMelodia	Analyste
RawMoments	Analyste
ReplayGain	Analyste
Resample	Analyste
ResampleFFT	Analyste
RhythmDescriptors	Analyste
RhythmExtractor2013	Analyste
RhythmExtractor	Analyste
RhythmTransform	Analyste
RMS	Analyste
RollOff	Analyste
Scale	Analyste
Scaler	Analyste

Nom	Rôle
SilenceRate	Analyste
SineSubtraction	Analyste
SingleBeatLoudness	Analyste
Slicer	Analyste
SoundFontAnalyst	Analyste
SparseAutoencoderFeeder	Alimenteur de modèle
SparseAutoencoderPredictor	Prédicteur de modèle
SparseAutoencoderTrainer	Entraîneur de modèle
SpectralCentroidTime	Analyste
SpectralComplexity	Analyste
SpectralContrast	Analyste
SpectralPeaks	Analyste
SpectralWhitening	Analyste
Spectrum	Analyste
Spline	Analyste
SprModelAnal	Analyste
SprModelSynth	Analyste
SpsModelAnal	Analyste
SpsModelSynth	Analyste
StochasticModelAnal	Analyste
StochasticModelSynth	Analyste
StrongDecay	Analyste
StrongPeak	Analyste
SuperFluxExtractor	Analyste
SuperFluxNovelty	Analyste
SuperFluxPeaks	Analyste
TCToTotal	Analyste
TempoScaleBands	Analyste
TempoTap	Analyste
TempoTapDegara	Analyste
TempoTapMaxAgreement	Analyste
TempoTapTicks	Analyste
Thresholder	Analyste
TimidityRender	Rendu audio
TonalExtractor	Analyste
TonicIndianArtMusic	Analyste
TriangularBands	Analyste
Trimmer	Analyste
Tristimulus	Analyste
TuningFrequency	Analyste
TuningFrequencyExtractor	Analyste
UnaryOperator	Analyste
UnaryOperatorStream	Analyste
Variance	Analyste
Vibrato	Analyste
WarpedAutoCorrelation	Analyste
Windowing	Analyste

Tableau A.1: Liste des traitements disponibles dans Mélodium (cont.)

APPENDICE B

EXEMPLE DE DOCUMENTATION D'UN TRAITEMENT

Ci-après, un exemple de documentation fournie par Mélodium concernant le traitement `SpectralPeaks` issu de la bibliothèque Essentia (Bogdanov *et al.*, 2013), notamment utilisé pour le descripteur HPCP, relevant du rôle *analystes*, et de type mono-piste.

```
$ melodium --info-treatments SpectralPeaks
SpectralPeaks [Analyst, Mono-track] :
    Description:
        [Spectral] This algorithm extracts peaks from a spectrum. It is
        important to note that the peak algorithm is independent of an input
        that is linear or in dB, so one has to adapt the threshold to fit
        with the type of data fed to it. The algorithm relies on PeakDetection
        algorithm which is run with parabolic interpolation [1]. The exactness
        of the peak-searching depends heavily on the windowing type. It gives
        best results with dB input, a blackman-harris 92dB window and
        interpolation set to true. According to [1], spectral peak frequencies
        tend to be about twice as accurate when dB magnitude is used rather
        than just linear magnitude. For further information about the peak
        detection, see the description of the PeakDetection algorithm.

        It is recommended that the input "spectrum" be computed by the
        Spectrum algorithm. This algorithm uses PeakDetection. See
        documentation for possible exceptions and input requirements
        on input "spectrum".

        References:
          [1] Peak Detection,
          http://ccrma.stanford.edu/~jos/parshl/Peak_Detection_Steps_3.html
    Parameters:
        Real magnitudeThreshold:
            peaks below this given threshold are not outputted
            Default: 0              Admitted values: [-inf, inf]

        Real maxFrequency:
```

the maximum frequency of the range to evaluate [Hz]
Default: *5000* Admitted values: [*0*, *inf*]

Integer *maxPeaks*:
the maximum number of returned peaks
Default: *100* Admitted values: [*1*, *2147483647*]

Real *minFrequency*:
the minimum frequency of the range to evaluate [Hz]
Default: *0* Admitted values: [*0*, *inf*]

String *orderBy*:
the ordering type of the outputted peaks (ascending by frequency or descending by magnitude)
Default: 'frequency' Admitted values: {'frequency', 'magnitude'}

Real *sampleRate*:
the sampling rate of the audio signal [Hz]
Default: *44100* Admitted values: [*0*, *inf*]

Require:

Vector²<Real> *spectrum*:
the input spectrum

Provide:

Vector²<Real> *frequencies*:
the frequencies of the spectral peaks [Hz]

Vector²<Real> *magnitudes*:
the magnitudes of the spectral peaks

APPENDICE C

SCRIPTS MÉLODIUM

C.1 Extraction du HPCP

La séquence `extractionH` suivante est utilisée pour extraire le HPCP à destination des expérimentations H tel que décrit en Section 5.2.

```
sequence extractionH {
    /*
     * Lecture des fichiers MIDI, associés à leur
     * fichiers audio.
     * - taux d'échantillonnage 44100Hz
     * - taille des trames à 4096
     * - saut entre trames de 2048
     * Parcours (récursif) des fichiers du répertoire
     * local 'midis'.
     */
    preparator prep(MidiFile) {
                sampleRate = 44100
                frameSize = 4096
                hopSize = 2048

                "./midis/"
        }

        /*
         * Paramétrage des traitements pour l'extraction du
         * descripteur HPCP, et des données musicales depuis
         * le MIDI.
         */
        MidiAnalyst(prep[MidiTracksAnalyst], presenceOnly=true)
        FrameCutter(frameSize=4096, hopSize=2048, startFromZero=true, lastFrameToEndOfFile=true)
        Windowing(type="blackmanharris92", size=4096)
        Spectrum(size=4096)
        SpectralPeaks(sampleRate=44100, orderBy="magnitude", magnitudeThreshold=0.00001,
        ↪  minFrequency=40, maxFrequency=5000, maxPeaks=10000)
```

```
29          HPCP(sampleRate=44100, windowSize=0.5, harmonics=8, weightType="cosine", size=120)
30          Accumulator(cumulation=9)
31
32          /*
33           * Connexion des traitements.
34           */
35          prep[MonoLoader].audio ---> FrameCutter.signal,frame
36          -> Windowing.frame,frame -> Spectrum.frame,spectrum
37          -> SpectralPeaks.spectrum
38
39          SpectralPeaks.frequencies -> HPCP.frequencies
40          SpectralPeaks.magnitudes --> HPCP.magnitudes
41
42          HPCP.hpcp -> Accumulator.input
43          // Accumulator.output fournit les données pour l'apprentissage.
44  }
```

C.2 Extraction du HPCP et spectre

La séquence `extractionHS` suivante est utilisée pour extraire les descripteurs HPCP et spectre à destination des expérimentations HS tel que décrit en Section 5.2.

```
1   sequence extractionHS {
2       /*
3        * Lecture des fichiers MIDI, associés à leur
4        * fichiers audio.
5        * - taux d'échantillonnage 44100Hz
6        * - taille des trames à 4096
7        * - saut entre trames de 2048
8        * Parcours (récursif) des fichiers du répertoire
9        * local 'midis'.
10       */
11      preparator prep(MidiFile) {
12                  sampleRate = 44100
13                  frameSize = 4096
14                  hopSize = 2048
15
16                  "./midis/"
17          }
18
19          /*
20           * Paramétrage des traitements pour l'extraction des
21           * descripteurs spectre et HPCP, et des données
22           * musicales depuis le MIDI.
23           */
24          MidiAnalyst(prep[MidiTracksAnalyst], presenceOnly=true)
```

```
25          FrameCutter(frameSize=4096, hopSize=2048, startFromZero=true, lastFrameToEndOfFile=true)
26          Windowing(type="blackmanharris92", size=4096)
27          Spectrum(size=4096)
28          SpectralPeaks(sampleRate=44100, orderBy="magnitude", magnitudeThreshold=0.00001,
            ↪  minFrequency=40, maxFrequency=5000, maxPeaks=10000)
29          HPCP(sampleRate=44100, windowSize=0.5, harmonics=8, weightType="cosine", size=120)
30          Accumulator(cumulation=9)
31
32          /*
33           * Connexion des traitements.
34           */
35          prep[MonoLoader].audio ----> FrameCutter.signal,frame
36          -> Windowing.frame,frame --> Spectrum.frame,spectrum
37          -> SpectralPeaks.spectrum
38
39          SpectralPeaks.frequencies -> HPCP.frequencies
40          SpectralPeaks.magnitudes --> HPCP.magnitudes
41
42          /*
43           * Normalisation du spectre.
44           */
45          SpectrumScaler(Scaler)
46          Spectrum.spectrum ---------> SpectrumScaler.input
47
48          /*
49           * Concaténation du spectre et du HPCP.
50           */
51          FinalConcat(Concatenator)
52
53          SpectrumScaler.output -----> FinalConcat.first
54          HPCP.hpcp -----------------> FinalConcat.second
55
56          FinalConcat.output --------> Accumulator.input
57          // Accumulator.output fournit les données pour l'apprentissage.
58  }
```

C.3 Extraction du HPCP, MFCC, et GFCC

La séquence `extractionHMG` suivante est utilisée pour extraire les descripteurs HPCP, MFCC, et GFCC à destination des expérimentations HMG tel que décrit en Section 5.2.

```
1   sequence extractionHMG {
2       /*
3        * Lecture des fichiers MIDI, associés à leur
4        * fichiers audio.
5        * - taux d'échantillonnage 44100Hz
```

```
6         * - taille des trames à 4096
7         * - saut entre trames de 2048
8         * Parcours (récursif) des fichiers du répertoire
9         * local 'midis'.
10        */
11           preparator prep(MidiFile) {
12                   sampleRate = 44100
13                   frameSize = 4096
14                   hopSize = 2048
15
16                   "./midis/"
17           }
18
19           /*
20            * Paramétrage des traitements pour l'extraction du
21            * spectre, utilisé pour l'obtention des descripteurs
22            * HPCP, MFCC, et GFCC.
23            */
24           MidiAnalyst(prep[MidiTracksAnalyst], presenceOnly=true)
25           FrameCutter(frameSize=4096, hopSize=2048, startFromZero=true, lastFrameToEndOfFile=true)
26           Windowing(type="blackmanharris92", size=4096)
27           Spectrum(size=4096)
28
29           /*
30            * Connexion des traitements pour l'obtention du spectre.
31            */
32           prep[MonoLoader].audio -----> FrameCutter.signal,frame
33           -> Windowing.frame,frame ---> Spectrum.frame
34
35           /*
36            * Paramétrage des traitements pour l'extraction du
37            * HPCP, et connexion de ceux-ci.
38            */
39           SpectralPeaks(sampleRate=44100, orderBy="magnitude", magnitudeThreshold=0.00001,
             ↪  minFrequency=40, maxFrequency=5000, maxPeaks=10000)
40           HPCP(sampleRate=44100, windowSize=0.5, harmonics=8, weightType="cosine", size=120)
41           Spectrum.spectrum ----------> SpectralPeaks.spectrum
42           SpectralPeaks.frequencies --> HPCP.frequencies
43           SpectralPeaks.magnitudes ---> HPCP.magnitudes
44
45
46           /*
47            * Paramétrage des traitements pour l'extraction du
48            * descripteur MFCC, avec découpage en 10 bandes de
49            * filtrage, une par octave.
50            */
51           MFCC_m1(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=31.25,
             ↪  lowFrequencyBound=15.9, numberBands=12, numberCoefficients=12)
52           MFCC_0(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=62.5,
             ↪  lowFrequencyBound=31.25, numberBands=12, numberCoefficients=12)
53           MFCC_1(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=125,
             ↪  lowFrequencyBound=62.5, numberBands=12, numberCoefficients=12)
```

```
54          MFCC_2(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=250,
            ↪  lowFrequencyBound=125, numberBands=12, numberCoefficients=12)
55          MFCC_3(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=500,
            ↪  lowFrequencyBound=250, numberBands=12, numberCoefficients=12)
56          MFCC_4(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=1000,
            ↪  lowFrequencyBound=500, numberBands=12, numberCoefficients=12)
57          MFCC_5(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=2000,
            ↪  lowFrequencyBound=1000, numberBands=12, numberCoefficients=12)
58          MFCC_6(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=4000,
            ↪  lowFrequencyBound=2000, numberBands=12, numberCoefficients=12)
59          MFCC_7(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=8000,
            ↪  lowFrequencyBound=4000, numberBands=12, numberCoefficients=12)
60          MFCC_8(MFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=16000,
            ↪  lowFrequencyBound=8000, numberBands=12, numberCoefficients=12)
61
62          /*
63           * Paramétrage de la concaténation des bandes
64           * de filtrage du MFCC.
65           */
66          MFCC_Concat_0(Concatenator)
67          MFCC_Concat_1(Concatenator)
68          MFCC_Concat_2(Concatenator)
69          MFCC_Concat_3(Concatenator)
70          MFCC_Concat_4(Concatenator)
71          MFCC_Concat_5(Concatenator)
72          MFCC_Concat_6(Concatenator)
73          MFCC_Concat_7(Concatenator)
74          MFCC_Concat_8(Concatenator)
75
76          /*
77           * Connexion des traitements pour l'obtention du MFCC.
78           */
79          Spectrum.spectrum --------------> MFCC_m1.spectrum,bands
80          ------------------------------> MFCC_Concat_0.first
81          Spectrum.spectrum --------------> MFCC_0.spectrum,bands
82          -> MFCC_Concat_0.second,output -> MFCC_Concat_1.first
83          Spectrum.spectrum --------------> MFCC_1.spectrum,bands
84          -> MFCC_Concat_1.second,output -> MFCC_Concat_2.first
85          Spectrum.spectrum --------------> MFCC_2.spectrum,bands
86          -> MFCC_Concat_2.second,output -> MFCC_Concat_3.first
87          Spectrum.spectrum --------------> MFCC_3.spectrum,bands
88          -> MFCC_Concat_3.second,output -> MFCC_Concat_4.first
89          Spectrum.spectrum --------------> MFCC_4.spectrum,bands
90          -> MFCC_Concat_4.second,output -> MFCC_Concat_5.first
91          Spectrum.spectrum --------------> MFCC_5.spectrum,bands
92          -> MFCC_Concat_5.second,output -> MFCC_Concat_6.first
93          Spectrum.spectrum --------------> MFCC_6.spectrum,bands
94          -> MFCC_Concat_6.second,output -> MFCC_Concat_7.first
95          Spectrum.spectrum --------------> MFCC_7.spectrum,bands
96          -> MFCC_Concat_7.second,output -> MFCC_Concat_8.first
97          Spectrum.spectrum --------------> MFCC_8.spectrum,bands
98          ------------------------------> MFCC_Concat_8.second
```

```
 99
100         /*
101          * Normalisation du MFCC.
102          */
103         MFCCBandsScaler(Scaler)
104         MFCC_Concat_8.output ------------> MFCCBandsScaler.input
105
106         /*
107          * Paramétrage des traitements pour l'extraction du
108          * descripteur GFCC, avec découpage en 10 bandes de
109          * filtrage, une par octave.
110          */
111         GFCC_m1(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=31.25,
            ↪  lowFrequencyBound=15.9, numberBands=12, numberCoefficients=12)
112         GFCC_0(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=62.5,
            ↪  lowFrequencyBound=31.25, numberBands=12, numberCoefficients=12)
113         GFCC_1(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=125,
            ↪  lowFrequencyBound=62.5, numberBands=12, numberCoefficients=12)
114         GFCC_2(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=250,
            ↪  lowFrequencyBound=125, numberBands=12, numberCoefficients=12)
115         GFCC_3(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=500,
            ↪  lowFrequencyBound=250, numberBands=12, numberCoefficients=12)
116         GFCC_4(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=1000,
            ↪  lowFrequencyBound=500, numberBands=12, numberCoefficients=12)
117         GFCC_5(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=2000,
            ↪  lowFrequencyBound=1000, numberBands=12, numberCoefficients=12)
118         GFCC_6(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=4000,
            ↪  lowFrequencyBound=2000, numberBands=12, numberCoefficients=12)
119         GFCC_7(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=8000,
            ↪  lowFrequencyBound=4000, numberBands=12, numberCoefficients=12)
120         GFCC_8(GFCC, inputSize=4096, sampleRate=44100, highFrequencyBound=16000,
            ↪  lowFrequencyBound=8000, numberBands=12, numberCoefficients=12)
121
122         /*
123          * Paramétrage de la concaténation des bandes
124          * de filtrage du GFCC.
125          */
126         GFCC_Concat_0(Concatenator)
127         GFCC_Concat_1(Concatenator)
128         GFCC_Concat_2(Concatenator)
129         GFCC_Concat_3(Concatenator)
130         GFCC_Concat_4(Concatenator)
131         GFCC_Concat_5(Concatenator)
132         GFCC_Concat_6(Concatenator)
133         GFCC_Concat_7(Concatenator)
134         GFCC_Concat_8(Concatenator)
135
136         /*
137          * Connexion des traitements pour l'obtention du GFCC.
138          */
139         Spectrum.spectrum ---------------> GFCC_m1.spectrum,bands
140         -------------------------------> GFCC_Concat_0.first
```

```
141         Spectrum.spectrum ---------------> GFCC_0.spectrum,bands
142         -> GFCC_Concat_0.second,output -> GFCC_Concat_1.first
143         Spectrum.spectrum ---------------> GFCC_1.spectrum,bands
144         -> GFCC_Concat_1.second,output -> GFCC_Concat_2.first
145         Spectrum.spectrum ---------------> GFCC_2.spectrum,bands
146         -> GFCC_Concat_2.second,output -> GFCC_Concat_3.first
147         Spectrum.spectrum ---------------> GFCC_3.spectrum,bands
148         -> GFCC_Concat_3.second,output -> GFCC_Concat_4.first
149         Spectrum.spectrum ---------------> GFCC_4.spectrum,bands
150         -> GFCC_Concat_4.second,output -> GFCC_Concat_5.first
151         Spectrum.spectrum ---------------> GFCC_5.spectrum,bands
152         -> GFCC_Concat_5.second,output -> GFCC_Concat_6.first
153         Spectrum.spectrum ---------------> GFCC_6.spectrum,bands
154         -> GFCC_Concat_6.second,output -> GFCC_Concat_7.first
155         Spectrum.spectrum ---------------> GFCC_7.spectrum,bands
156         -> GFCC_Concat_7.second,output -> GFCC_Concat_8.first
157         Spectrum.spectrum ---------------> GFCC_8.spectrum,bands
158         ---------------------------------> GFCC_Concat_8.second
159
160         /*
161          * Normalisation du GFCC.
162          */
163         GFCCBandsScaler(Scaler)
164         GFCC_Concat_8.output ------------> GFCCBandsScaler.input
165
166         /*
167          * Paramétrage de la concaténation des *FCC,
168          * et connexion.
169          */
170         AllFCCConcat(Concatenator)
171         MFCCBandsScaler.output ----------> AllFCCConcat.first
172         GFCCBandsScaler.output ----------> AllFCCConcat.second
173
174         /*
175          * Paramétrage de la concaténation des *FCC et
176          * du HPCP, et connexion.
177          */
178         FinalConcat(Concatenator)
179         AllFCCConcat.output -------------> FinalConcat.first
180         HPCP.hpcp -----------------------> FinalConcat.second
181
182         /*
183          * Accumulation de chaque trame avec les 9 précédentes.
184          */
185         Accumulator(cumulation=9)
186         FinalConcat.output --------------> Accumulator.input
187         // Accumulator.output fournit les données pour l'apprentissage.
188 }
```

C.4 Obtention d'un modèle de classification monophonique mono-instrumental

La séquence `HSaConst` procède à l'extraction du HPCP, et l'entraînement d'un auto-encodeur éparse et réseau de neurones pleinements connectés à propagation avant à taille constante sur 50 époques, tels que définis par l'expérimentation H$SaConst10(1200 \rightarrow 400; 10 \times 400)$, voir Chapitre 5.

```
1  /*
2   * Définition d'un modèle d'auto-encodeur
3   * épars.
4   * - taille de la couche d'entrée à 1200
5   * - taille de la couche intermédiaire à 400
6   */
7  model ae(SparseAutoencoder) {
8          visibleSize = 1200
9          hiddenSize = 400
10 }
11
12 /*
13  * Définition d'un modle de réseau de
14  * neurones à propagation avant à taille
15  * constante.
16  * - 10 couches de fonction "PReLU"
17  * - 1 couche de fonction "Sigmoid"
18  * - taille du vecteur d'entrée à 400
19  * - taille des couches intermédiaires à 400
20  * - taille de la couche de sortie à 128
21  */
22 model reseau(ConfigurableRectFfn) {
23         layers = ["PReLU", "PReLU", "PReLU", "PReLU", "PReLU", "PReLU", "PReLU", "PReLU",
           ↪  "PReLU", "PReLU", "Sigmoid"]
24
25         inputSize = 400
26         layersSize = 400
27         outputSize = 128
28 }
29
30 /*
31  * Séquence de création de modèle de classification
32  * monophonique mono-instrumental.
33  */
34 sequence HSaConst {
35     /*
36      * Lecture des fichiers MIDI, associés à leur
37      * fichiers audio.
38      * - taux d'échantillonnage 44100Hz
39      * - taille des trames à 4096
```

```
40          * - saut entre trames de 2048
41          * Parcours (récursif) des fichiers du répertoire
42          * local 'midis'.
43          */
44             preparator prep(MidiFile) {
45                     sampleRate = 44100
46                     frameSize = 4096
47                     hopSize = 2048
48
49                     "./midis/"
50             }
51
52             /*
53              * Paramétrage des traitements pour l'extraction du
54              * descripteur HPCP, et des données musicales depuis
55              * le MIDI.
56              */
57             MidiAnalyst(prep[MidiTracksAnalyst], presenceOnly=true)
58             FrameCutter(frameSize=4096, hopSize=2048, startFromZero=true, lastFrameToEndOfFile=true)
59             Windowing(type="blackmanharris92", size=4096)
60             Spectrum(size=4096)
61             SpectralPeaks(sampleRate=44100, orderBy="magnitude", magnitudeThreshold=0.00001,
               ↪  minFrequency=40, maxFrequency=5000, maxPeaks=10000)
62             HPCP(sampleRate=44100, windowSize=0.5, harmonics=8, weightType="cosine", size=120)
63             Accumulator(cumulation=9)
64
65             /*
66              * Connexion des traitements.
67              */
68             prep[MonoLoader].audio ----> FrameCutter.signal,frame
69             -> Windowing.frame,frame --> Spectrum.frame,spectrum
70             -> SpectralPeaks.spectrum
71
72             SpectralPeaks.frequencies -> HPCP.frequencies
73             SpectralPeaks.magnitudes --> HPCP.magnitudes
74
75             HPCP.hpcp -> Accumulator.input
76
77             /*
78              * Établissement de l'auto-encodeur.
79              */
80             AEFeeder(ae[SparseAutoencoderFeeder])
81             AETrainer(ae[SparseAutoencoderTrainer], trainingOutput="ae_stats.log")
82             AEPredictor(ae[SparseAutoencoderPredictor])
83
84             /*
85              * Alimentation, puis entraînement de
86              * l'auto-encodeur.
87              */
88             Accumulator.output ----> AEFeeder.data
89             AEFeeder -> AETrainer -> AEPredictor
90
```

```
91          /*
92           * Alimentation du réseau de neurones.
93           */
94          FFNFeeder(reseau[MlpackModelFeeder])
95          Accumulator.output --------------> AEPredictor.data
96          AEPredictor.features ------------> FFNFeeder.data
97          MidiAnalyst.midiTrackContent1 --> FFNFeeder.label
98
99          /*
100          * Entraînement du réseau de neurones.
101          */
102         FFNTrainer(reseau[MlpackModelTrainer], trainingOutput="ffn_stats.log", trainings=50)
103         FFNPredictor(reseau[MlpackModelPredictor])
104
105         FFNFeeder ---> FFNTrainer -----> FFNPredictor
106
107         /*
108          * Évaluation du réseau de neurones.
109          * On seuille les labels (binarisation)
110          * prédits avant d'évaluer par
111          * rapport au MIDI.
112          */
113         AEPredictor.features ----------> FFNPredictor.data
114
115         Thresholder(Thresholder)
116         Evaluator(Evaluator, output="ffn_eval.csv")
117
118         FFNPredictor.assignments ------> Thresholder.input
119         Thresholder.output ------------> Evaluator.values
120     MidiAnalyst.midiTrackContent1 -> Evaluator.ground
121 }
```

APPENDICE D

RESSOURCES UTILISÉES PAR LES EXPÉRIMENTATIONS

Les Tableaux D.1 et D.2 montrent les ressources utilisées par les expérimentations décrites au Chapitre 5. La colonne *Expérimentations* se réfère à la notation spécifiée en Section 5.5. Les groupes Fluid R3 GM et MAPS se réfèrent aux expérimentations relatives aux jeux de données décrits en Section 5.1. Toutes ces exécutions se font dans le cadre décrit en Section 5.6.

Certaines valeurs ne sont pas les valeurs effectives complètes, en raison de divers facteurs. Elles sont indiquées par les annotations suivantes :

- $*$: valeur d'allocation demandée, et non valeur effective, dûe à une interruption des journaux statistiques sur le calculateur n'ayant pas d'impact sur l'exécution.
- † : valeur maximale atteinte lors de l'exécution, avant terminaison précoce dûe à l'épuisement des ressources allouées.

Expérimentation	Fluid R3 GM											MAPS			
	000	013	019	024	032	040	048	057	064	075	080	Isolé	Aléatoires	Usuels	Musiques
HMG3*SaConst*10(3(1200→400); 10 × 1200)	16,00*	16,00*	16,00*	16,00*	16,00*	16,00*	16,00*	16,00*	16,00*	16,00*	16,00*	30,00*	30,00*	30,00*	30,00*
HMG3*SaConst*30(3(1200→400); 30 × 1200)	20,00*	20,00*	20,00*	20,00*	20,00*	20,00*	20,00*	20,00*	20,00*	20,00*	20,00*	30,00*	30,00*	30,00*	30,00*
HMG3*SaConst*30(3(1200→400); 50 × 1200)	30,00*	30,00*	30,00*	30,00*	30,00*	30,00*	30,00*	30,00*	30,00*	30,00*	30,00*	40,00*	40,00*	40,00*	40,00*
HMG3*SaDéc*10(3(1200→400); 1×1200; 2×800; 3×400; 4×200)	9,99†	9,96†	10,52†	8,98	4,40	9,98†	1,92	7,46	8,81	6,15	4,58	60,00*	60,00*	60,00*	60,00*
HMG*Const*10(10 × 3600)	10,00†	9,99†	9,98†	10,00†	9,99†	10,00†	5,43	10,00†	10,00†	10,00†	10,00†	28,24	28,70	28,36	29,44
HMG*Const*10(10 × 400)	10,00†	9,94†	9,92†	10,00†	9,96†	9,88†	5,06	9,91†	10,00†	9,96†	9,98	28,64	28,59	28,10	29,99†
HMG*Const*30(30 × 3600)	9,97†	10,00†	9,97†	9,99†	9,98†	9,98†	5,21	10,00†	9,99†	9,99†	9,98†	27,91	28,22	30,00†	30,00†
HMG*Const*30(30 × 400)	9,82	10,00†	9,94†	9,91†	9,99†	10,00†	5,04	9,95†	9,94†	9,98†	9,99†	28,90	28,43	30,00†	28,93
HMG*Const*50(50 × 3600)	9,98†	10,00†	10,00†	9,98†	10,00†	10,00†	4,77	9,99†	10,00†	10,00†	9,97†	28,11	28,21	28,50	29,81
HMG*Const*50(50 × 400)	19,91†	19,88†	19,96†	19,91†	12,68	19,94†	5,08	19,95†	19,92†	17,94	13,32	50,00*	50,00*	50,00*	50,00*
HMG*Déc*10(3×7200; 7×3600)	7,66	7,56	7,25	7,16	6,83	7,41	6,59	7,10	7,35	6,98	6,86	30,00*	30,00*	30,00*	30,00*
HMG*Déc*10(1×3600; 1×1200; 2×800; 3×400; 3×200)	8,43	5,83	5,72	5,89	3,36	6,75	1,86	4,34	4,86	4,10	3,04	28,52	27,87	29,68	28,97
HMG*Déc*10(3×3600; 3×1200; 6×800; 9×400; 9×200)	5,61	4,75	4,26	3,93	2,95	4,68	2,31	3,57	4,46	3,72	3,02	28,39	28,98	28,94	28,57
HMG*SaDéc*10(3600→1200; 3×7200; 7×3600)	6,33	6,12	6,26	6,21	6,00	6,22	5,91	6,07	6,13	6,07	6,03	30,00*	30,00*	30,00*	30,00*
HMG*SaDéc*10(3600→1200; 1×1200; 2×800; 3×400; 4×200)	2,93	2,66	2,69	2,61	2,35	2,61	2,19	2,48	2,56	2,44	2,40	30,00*	30,00*	30,00*	30,00*
HMG*SaDéc*30(3600→1200; 3×1200; 6×800; 9×400; 12×200)	3,00	2,70	2,57	2,57	2,36	2,68	2,22	2,47	2,55	2,43	2,37	30,00*	30,00*	30,00*	30,00*
H*Const*10(10 × 1200)	8,00†	8,00†	7,98†	8,00†	4,52	8,00†	1,91	7,66	7,98	6,24	4,65	29,20	29,19	29,25	29,76
H*Const*10(10 × 400)	16,55	11,07	10,71	9,16	4,41	10,70	1,83	7,60	9,00	6,27	4,61	29,40	29,04	29,33	29,65
H*Const*30(30 × 1200)	8,00†	8,00†	7,99†	8,00†	4,53	8,00†	1,94	7,69	8,00†	6,26	4,71	50,00*	50,00*	50,00*	50,00*
H*Const*30(30 × 400)	16,63	11,12	10,78	9,15	4,41	10,69	1,85	7,59	8,93	6,25	4,60	50,00*	50,00*	50,00*	50,00*
H*Const*50(50 × 1200)	10,00†	9,99†	10,00†	9,23	4,52	10,00†	1,97	7,79	9,13	6,31	4,73	60,00*	60,00*	60,00*	60,00*
H*Const*50(50 × 400)	16,55	11,12	10,73	9,16	4,41	10,80	1,85	7,60	9,00	6,21	4,62	60,00*	60,00*	60,00*	60,00*
H*Déc*10(3×2400; 7×1200)	4,40	3,29	3,32	2,94	1,96	3,23	1,39	2,56	2,94	2,21	2,17	30,00†	29,72	30,00†	29,35
H*Déc*10(1×1200; 2×400; 7×200)	14,99	10,88	10,46	8,95	4,45	10,62	1,92	7,33	8,75	6,06	4,68	50,00*	50,00*	50,00*	50,00*
H*Déc*30(10×2400; 20×1200)	3,75	3,23	3,21	3,07	2,77	3,22	2,68	2,95	3,13	2,91	2,79	33,45	36,87	35,26	27,55
H*Déc*30(3×1200; 7×400; 20×200)	12,77	7,63	8,44	7,25	3,64	8,27	1,62	5,95	5,89	5,30	3,81	40,00†	39,71	39,22	39,61
H*SaConst*10(1200→400; 10×400)	5,94	4,01	3,90	3,35	1,69	3,88	0,81	2,77	3,28	2,32	1,75	22,00	22,00	21,68	21,74
H*SaConst*30(1200→400; 30×400)	5,90	4,01	3,87	3,32	1,70	3,94	0,86	2,85	3,29	2,31	1,77	23,67	23,61	23,69	23,71
H*SaConst*50(1200→400; 50×400)	5,94	4,00	3,90	3,34	1,75	3,92	0,83	2,83	3,27	2,31	1,81	27,72	27,68	27,73	27,70
H*SaDéc*10(1200→400; 3×2400; 7×1200)	1,84	1,52	1,56	1,39	1,26	1,49	1,12	1,40	1,49	1,40	1,23	30,00*	30,00*	30,00*	30,00*
H*SaDéc*10(1200→400; 3×400; 7×200)	5,68	3,89	3,78	3,24	1,67	3,78	0,82	2,70	3,19	2,27	1,71	30,00*	30,00*	30,00*	30,00*
H*SaDéc*30(1200→400; 10×2400; 20×1200)	2,88	2,69	2,71	2,68	2,60	2,71	2,52	2,61	2,65	2,61	2,56	30,00*	30,00*	30,00*	30,00*
H*SaDéc*30(1200→400; 10×400; 20×200)	5,82	4,03	3,87	3,29	1,76	3,90	0,90	2,81	3,30	2,35	1,82	30,00*	30,00*	30,00*	30,00*
HS*Const*10(10 × 9690)	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	80,00*	80,00*	80,00*	80,00*
HS*Const*10(10 × 400)	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	80,00*	80,00*	80,00*	80,00*
HS*Const*30(30 × 9690)	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*
HS*Const*30(30 × 400)	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*	80,00*
HS*Const*50(50 × 9690)	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*
HS*Const*50(50 × 400)	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*	100,00*
HS*Déc*10(1×21690; 1×9690; 1×3230; 1×1600; 1×800; 2×400; 3×200)	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	50,00*	80,00*	80,00*	80,00*	80,00*
HS*Déc*30(3×21690; 3×9690; 3×3230; 3×1600; 3×800; 6×400; 9×200)	59,59	58,09	57,60	57,48	57,16	58,02	56,72	57,61	57,73	57,71	57,40	100,00*	100,00*	100,00*	100,00*
HS*SaDéc*10(21690→3230; 1×3230; 2×1600; 2×800; 2×400; 3×200)	32,99	31,99	31,96	31,58	30,88	30,89	30,67	31,36	31,61	31,11	30,88	80,00*	80,00*	80,00*	80,00*

Tableau D.1: Mémoire utilisée par les expérimentations, en gigaoctets.

Expérimentation	Fluid R3 GM										MAPS			
	000	013	019	024	032	040	048	057	064	075	080	Isolé	Aléatoires	Usuels	Musiques
HMG3*SaConst*10(3(1200→400); 10 × 1200)	2-10:58:51	2-16:00:49	2-13:15:44	2-13:57:20	2-12:47:05	2-12:47:05	2-14:09:23	2-15:46:02	2-13:01:16	3-03:31:06	2-14:35:05	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HMG3*SaConst*30(3(1200→400); 30 × 1200)	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*
HMG3*SaConst*30(3(1200→400); 50 × 1200)	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HMG3*SaDéc*10(3(1200→400); 1×1200; 2×800; 3×400; 4×200)	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HMG*Const*10(10 × 3600)	1-15:34:35	2-06:06:53	2-12:55:02	2-16:25:03	3-05:32:50	2-09:08:32	2-12:40:00	2-11:36:52	2-13:17:50	2-09:35:38	2-10:37:44	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HMG*Const*10(10 × 400)	1-07:08:06	2-00:48:05	2-02:26:54	2-12:04:31	5-05:16:00	2-08:56:37	6-17:50:13	3-10:02:16	2-09:13:07	3-11:32:53	4-16:10:06	0-06:04:43	0-05:44:45	0-05:01:33	0-06:41:44
HMG*Const*30(30 × 3600)	0-04:15:08	0-06:27:17	0-04:35:02	0-06:01:56	0-12:54:35	0-04:33:20	0-18:54:11	0-06:51:14	0-05:45:37	0-08:55:12	0-11:15:55	0-01:01:44	0-01:22:20	0-00:57:26	0-00:52:36
HMG*Const*30(30 × 400)	1-09:36:00	2-02:49:19	2-04:27:12	2-13:41:55	5-11:26:56	2-06:13:02	7-00:00:08†	3-02:40:36	2-15:16:13	3-20:42:43	5-04:15:17	0-06:16:16	0-05:39:47	0-05:25:09	0-08:17:24
HMG*Const*50(50 × 3600)	0-03:26:56	0-05:36:49	0-05:58:33	0-06:23:55	0-13:42:20	0-05:31:00	0-19:57:41	0-07:49:14	0-06:36:12	0-09:43:54	0-12:45:50	0-01:01:23	0-00:56:02	0-00:55:33	0-00:38:02
HMG*Const*50(50 × 400)	1-13:05:30	2-07:58:54	2-10:30:42	3-00:03:30	6-06:35:40	2-10:32:32	7-00:00:08†	3-14:07:53	2-22:29:27	4-07:04:50	5-18:11:00	0-05:58:56	0-05:58:52	0-05:58:37	0-08:13:10
HMG*Déc*10(3×7200; 7×3600)	0-08:10:04	0-12:14:26	0-12:49:02	0-15:18:31	0-20:19:42	0-13:00:35	0-20:16:17	0-20:10:03	0-15:30:33	0-20:24:54	0-19:59:17	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HMG*Déc*10(1×3600; 1×1200; 2×800; 3×400; 3×200)	2-06:41:50	3-00:00:15†	2-06:03:28	2-06:03:27	2-05:51:15	2-00:28:22	2-00:28:58	2-00:28:22	3-00:00:02†	3-00:00:02†	3-00:00:02†	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*
HMG*Déc*10(3×3600; 3×1200; 6×800; 9×400; 9×200)	3-00:00:19†	3-00:00:19†	3-00:00:19†	3-00:00:19†	3-00:00:19†	3-00:00:19†	3-00:00:19†	3-00:00:19†	3-00:00:22†	3-00:00:22†	3-00:00:22†	0-13:55:24	0-11:37:00	0-12:22:04	0-16:01:03
HMG*SaDéc*10(3600→1200; 3×7200; 7×3600)	3-00:00:22†	3-00:00:22†	3-00:00:22†	3-00:00:22†	3-00:00:22†	3-00:00:05†	3-00:00:05†	3-00:00:05†	3-00:00:05†	3-00:00:05†	3-00:00:05†	1-01:05:43	0-21:36:24	1-00:53:19	1-06:29:55
HMG*SaDéc*10(3600→1200; 1×1200; 2×800; 3×400; 4×200)	2-00:00:26†	2-00:00:26†	2-00:00:26†	2-00:00:26†	2-00:00:26†	2-00:00:26†	2-00:00:26†	2-00:00:26†	2-00:00:26†	2-00:00:26†	2-00:00:26†	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HMG*SaDéc*30(3600→1200; 3×1200; 6×800; 9×400; 12×200)	1-11:15:00	1-07:25:24	1-10:19:32	1-07:21:16	1-04:56:44	1-07:06:08	1-03:59:00	1-05:47:23	1-06:29:01	1-06:04:49	1-08:47:38	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*
H*Const*10(10 × 1200)	3-00:00:24†	1-20:08:52	1-20:07:14	1-17:51:56	1-14:14:13	1-20:36:45	1-11:37:56	1-15:34:46	1-17:16:54	1-11:25:00	1-19:41:58	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*
H*Const*10(10 × 400)	0-09:45:27	0-17:02:07	0-14:48:18	0-16:51:14	0-23:20:20	0-14:43:20	0-19:00:04	0-19:06:12	0-17:41:23	0-22:55:34	0-19:22:05	0-02:13:44	0-02:33:18	0-02:23:53	0-02:44:17
H*Const*30(30 × 1200)	0-05:27:27	0-06:16:05	0-06:47:06	0-06:22:36	0-06:09:19	0-07:54:18	0-07:19:38	0-07:15:40	0-07:27:14	0-07:27:23	0-06:27:31	0-01:01:42	0-01:17:35	0-01:14:00	0-00:52:02
H*Const*30(30 × 400)	0-12:21:51	0-18:29:52	0-18:58:25	0-22:26:15	1-02:20:19	0-19:06:44	1-02:15:19	1-04:04:32	0-22:52:36	1-02:04:37	1-02:03:09	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
H*Const*50(50 × 1200)	0-09:00:54	0-08:16:22	0-09:13:37	0-08:43:57	0-08:41:23	0-08:43:29	0-08:45:03	0-08:39:21	0-08:41:35	0-08:40:16	0-08:38:53	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
H*Const*50(50 × 400)	0-19:43:22	1-05:25:04	1-06:59:23	1-09:21:13	1-09:19:32	1-07:52:43	1-09:43:24	1-08:38:26	1-07:38:03	1-07:53:23	1-13:45:38	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
H*Déc*10(3×2400; 7×1200)	0-12:01:09	0-10:56:49	0-10:59:53	0-10:59:58	0-10:56:47	0-10:49:31	0-10:50:12	0-10:48:51	0-10:51:14	0-11:53:13	0-10:49:00	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
H*Déc*10(1×1200; 2×400; 7×200)	3-00:00:15†	3-00:00:15†	3-00:00:15†	3-00:00:28†	3-00:00:28†	3-00:00:28†	3-00:00:28†	3-00:00:28†	3-00:00:28†	3-00:00:28†	3-00:00:28†	1-15:42:03	1-10:27:39	1-17:57:23	2-01:53:56
H*Déc*30(10×2400; 20×1200)	1-16:45:11	1-17:20:28	1-19:33:41	1-19:19:51	1-18:24:11	1-19:16:19	1-18:19:56	1-17:14:54	1-18:23:29	1-16:23:26	1-16:05:56	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*
H*Déc*30(3×1200; 7×400; 20×200)	3-00:00:15†	3-00:00:24†	3-00:00:24†	3-00:00:24†	3-00:00:24†	3-00:00:24†	3-00:00:24†	3-00:00:24†	3-00:00:14†	3-00:00:15†	3-00:00:14†	7-00:00:27†	7-00:00:27†	7-00:00:27†	7-00:00:27†
H*SaConst*10(1200→400; 10×400)	3-00:00:21†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:03†	3-00:00:19†	0-18:34:02	0-17:00:00	0-19:06:54	0-18:27:25
H*SaConst*30(1200→400; 30×400)	0-07:01:43	0-06:26:33	0-07:38:52	0-06:27:03	0-05:05:15	0-06:34:56	0-04:05:33	0-05:52:03	0-06:01:19	0-05:42:38	0-04:32:49	1-17:18:05	1-21:05:50	1-17:44:57	1-08:03:17
H*SaConst*50(1200→400; 50×400)	0-07:55:33	0-07:23:29	0-07:23:05	0-07:12:03	0-07:06:56	0-07:36:19	0-06:36:44	0-06:57:55	0-07:10:25	0-06:28:20	0-06:46:17	1-17:18:44	1-05:33:15	1-15:10:48	1-16:46:52
H*SaDéc*10(1200→400; 3×2400; 7×1200)	0-10:24:04	0-08:45:58	0-09:49:07	0-08:55:26	0-09:25:03	0-09:41:43	0-08:48:15	0-10:03:51	0-09:32:03	0-09:18:21	0-09:29:55	1-15:27:59	1-13:19:19	1-16:17:12	1-23:13:58
H*SaDéc*10(1200→400; 3×400; 7×200)	2-00:00:23†	2-00:00:27†	2-00:00:27†	2-00:00:27†	2-00:00:27†	2-00:00:27†	2-00:00:27†	2-00:00:27†	2-00:00:27†	2-00:00:27†	2-00:00:27†	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
H*SaDéc*30(1200→400; 10×2400; 20×1200)	0-14:15:39	0-12:47:45	0-12:19:18	0-15:08:55	0-11:32:38	0-12:21:27	0-12:02:17	0-12:15:39	0-12:49:38	0-14:29:37	0-11:36:22	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*
H*SaDéc*30(1200→400; 10×400; 20×200)	3-00:00:14†	3-00:00:14†	3-00:00:14†	3-00:00:14†	3-00:00:14†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	3-00:00:07†	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HS*Const*10(10 × 9690)	1-19:13:46	1-15:37:42	1-09:39:06	1-08:39:22	1-08:46:42	1-11:26:29	1-09:52:43	1-10:56:30	1-06:39:07	1-09:14:31	1-07:50:50	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HS*Const*10(10 × 400)	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HS*Const*30(30 × 9690)	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HS*Const*30(30 × 400)	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HS*Const*50(50 × 9690)	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HS*Const*50(50 × 400)	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HS*Déc*10(1×21690; 1×9690; 1×3230; 1×1600; 1×800; 2×400; 3×200)	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*
HS*Déc*30(3×21690; 3×9690; 3×3230; 3×1600; 3×800; 6×400; 9×200)	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	3-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*	5-00:00:00*
HS*SaDéc*10(21690→3230; 1×3230; 2×1600; 2×800; 2×400; 3×200)	3-00:00:27†	3-00:00:13†	3-00:00:06†	3-00:00:14†	3-00:00:12†	3-00:00:12†	3-00:00:12†	3-00:00:12†	3-00:00:12†	3-00:00:12†	3-00:00:12†	7-00:00:00*	7-00:00:00*	7-00:00:00*	7-00:00:00*

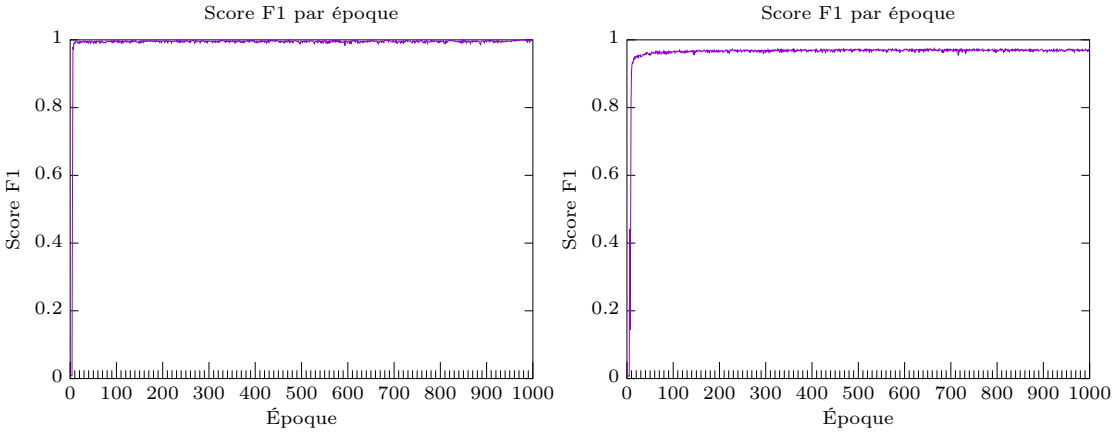
Tableau D.2: Durées des expérimentations, au format J-HH:MM:SS.

APPENDICE E

RÉSULTATS DÉTAILLÉS D'EXPÉRIMENTATIONS

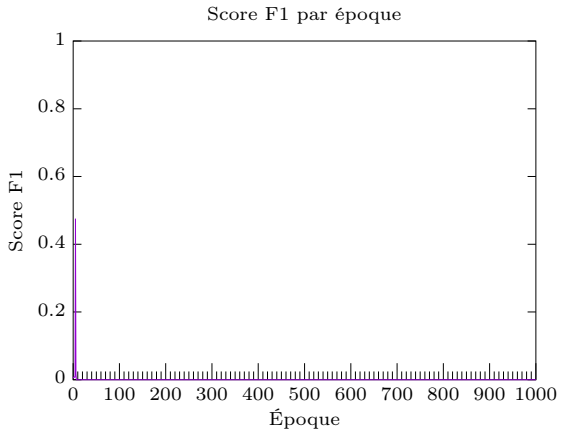
E.1 Apprentissages monophoniques mono-instrumentaux

Les sous-figures E.1 représentent le score F1 de l'expérimentation H$SaConst10(1200 \rightarrow 400; 10 \times 400)$, Section 6.3, sur l'ensemble des notes combinées, pour les époques d'apprentissage nº 0 à nº 1 000.

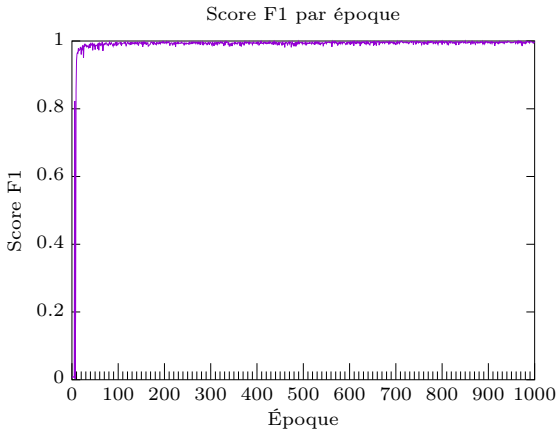


(a) Instrument *000 Piano* (b) Instrument *013 Xylophone*

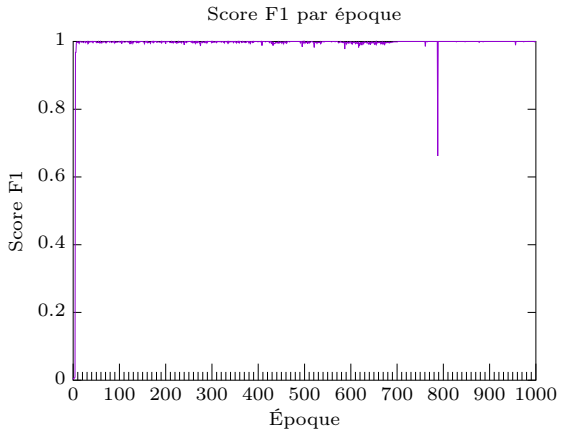
Figure E.1: Apprentissage monophonique du modèle H$SaConst10(1200 \rightarrow 400; 10 \times 400)$



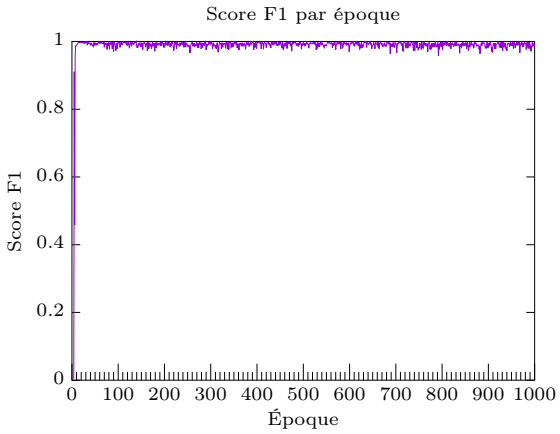
(c) Instrument *019 Orgue d'église*



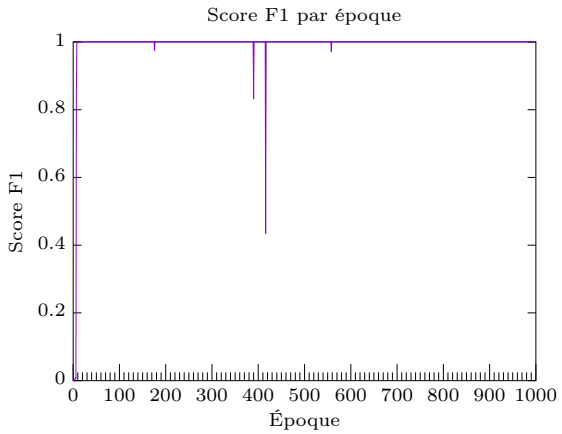
(d) Instrument *024 Guitare classique*



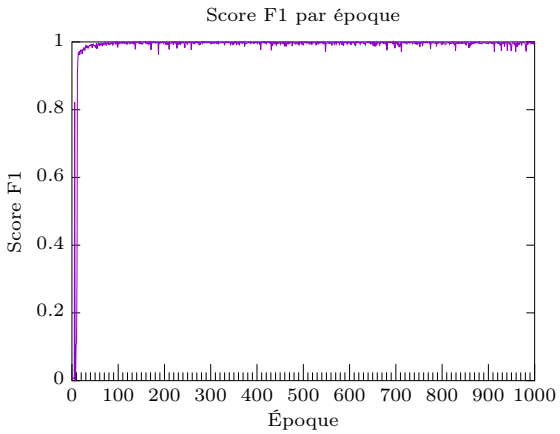
(e) Instrument *032 Basse acoustique*



(f) Instrument *040 Violon*

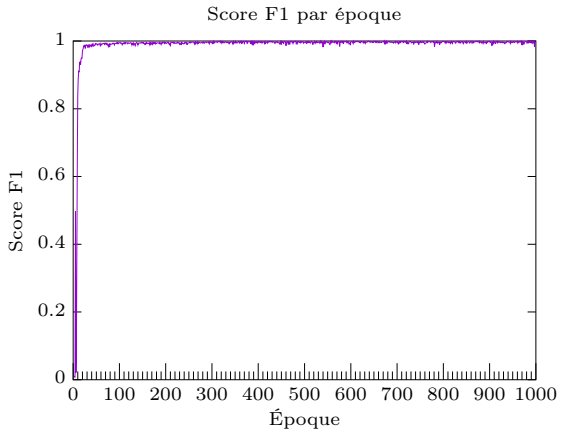


(g) Instrument *048 Ensemble de cordes*

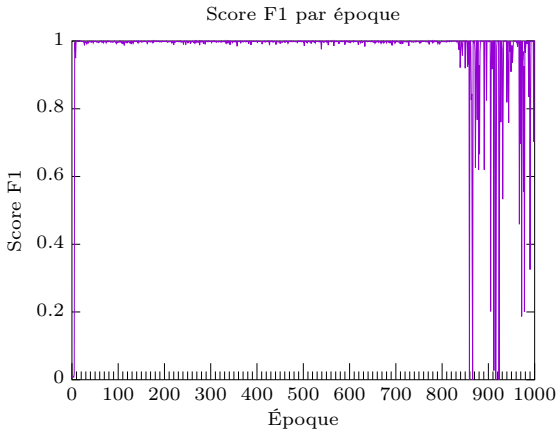


(h) Instrument *057 Trombone*

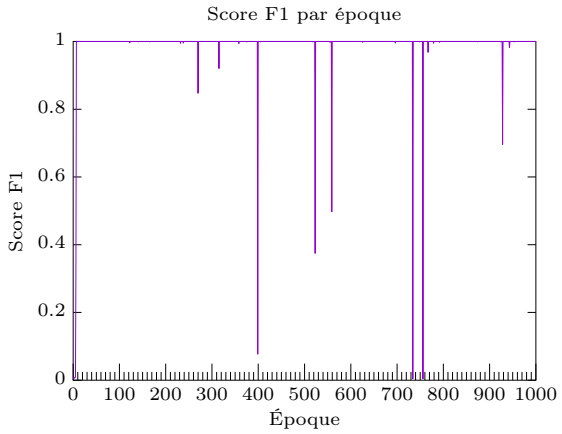
Figure E.1: Apprentissage monophonique du modèle H*SaConst*10(1200 $\rightarrow$ 400; 10×400) (cont.)



(i) Instrument *064 Saxophone soprano*



(j) Instrument *075 Flûte de pan*

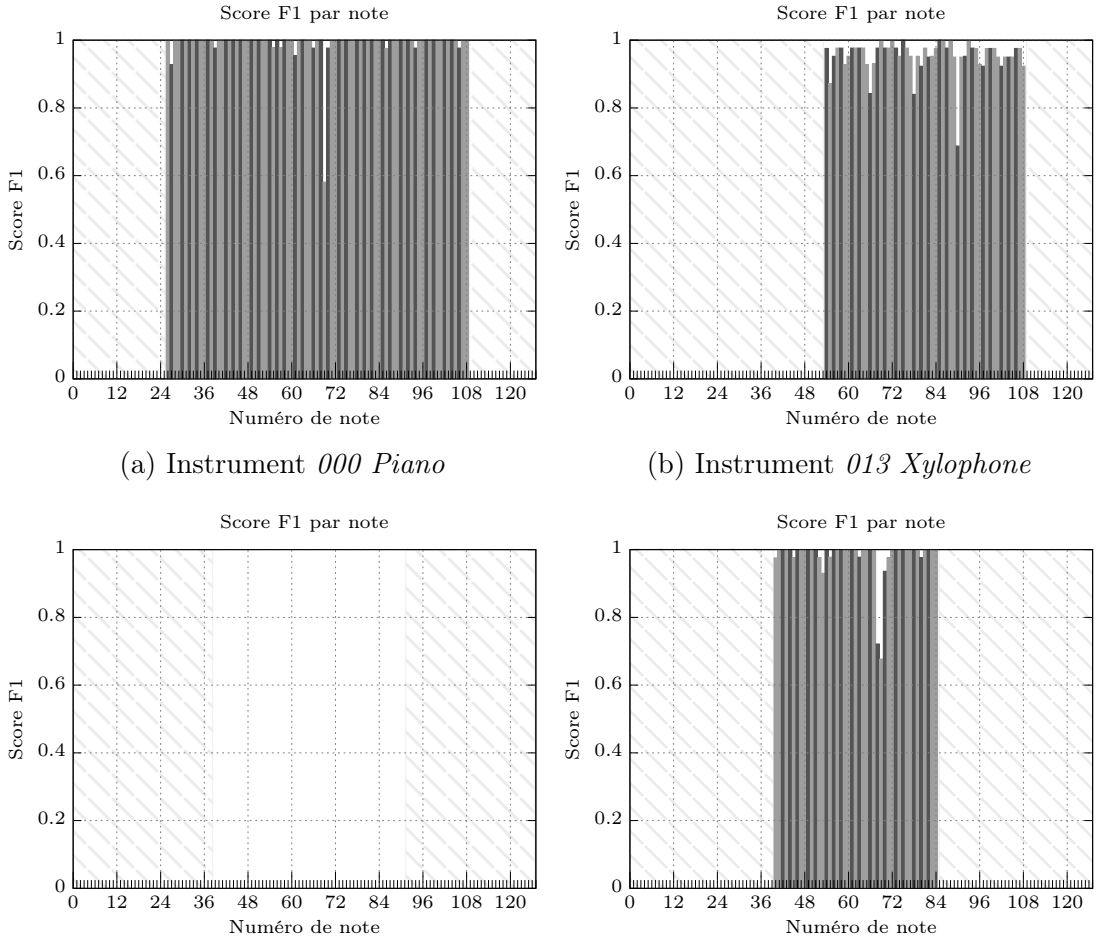


(k) Instrument *080 Signal carré*

Figure E.1: Apprentissage monophonique du modèle H$SaConst10(1200 \rightarrow 400; 10 \times 400)$ (cont.)

E.2 Époques d'apprentissages monophoniques mono-instrumentaux

Les sous-figures E.2 représentent le détail par note du score F1 de l'expérimentation H$SaConst10(1200 \rightarrow 400; 10 \times 400)$, Section 6.3, à l'époque n^o 50. Les notes sont positionnées d'après la norme MIDI, de 0 à 127, et colorées selon l'usage musical. Seules les notes faisant partie de la tessiture sont évaluées (Tableau 5.1), les autres sont en zone barrée.



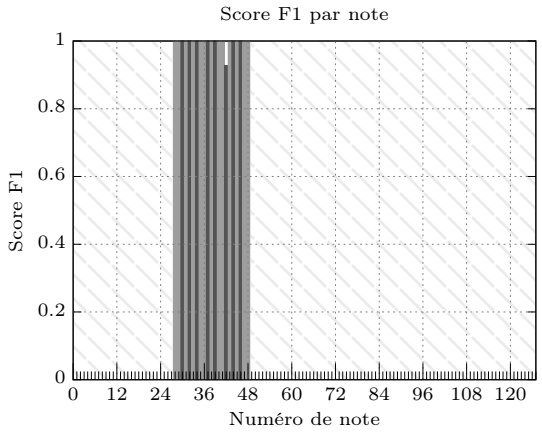
(a) Instrument *000 Piano*

(b) Instrument *013 Xylophone*

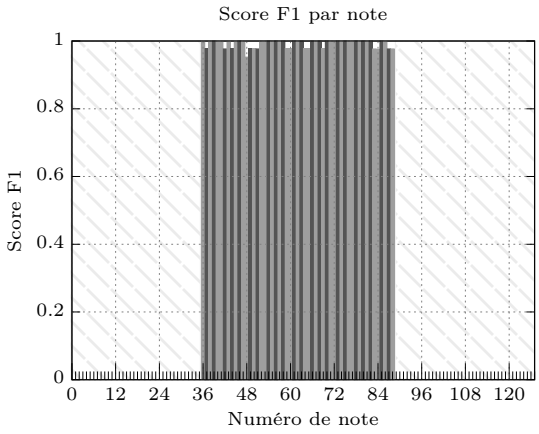
(c) Instrument *019 Orgue d'église*

(d) Instrument *024 Guitare classique*

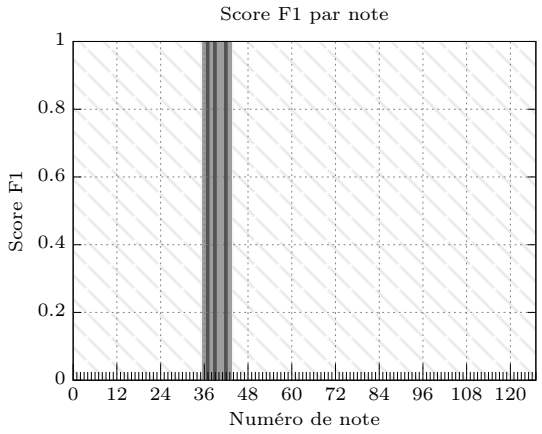
Figure E.2: Époque d'apprentissage monophonique du modèle H$SaConst10(1200 \rightarrow 400; 10 \times 400)$ n^o 50



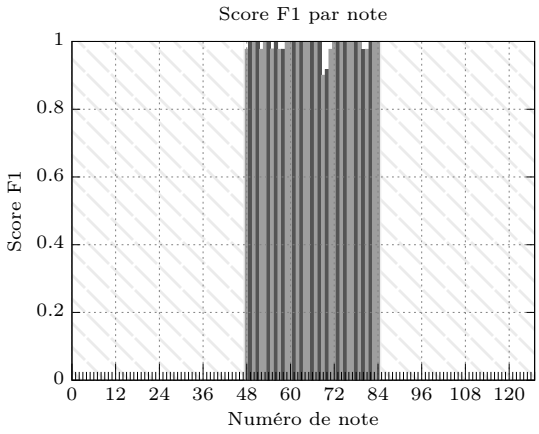
(e) Instrument *032 Basse acoustique*



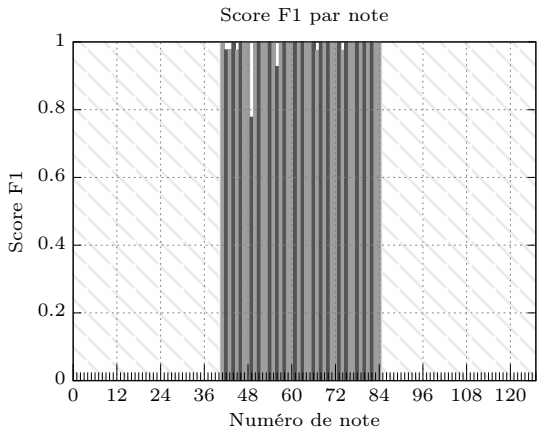
(f) Instrument *040 Violon*



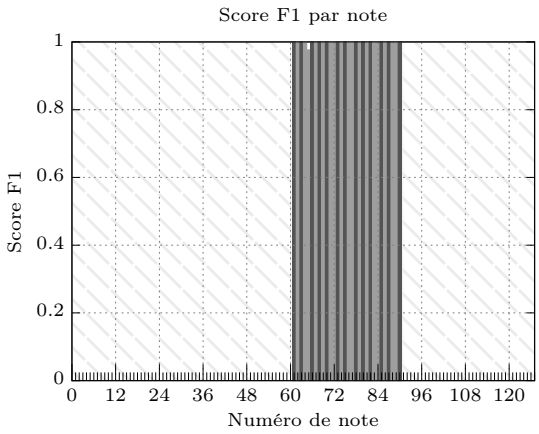
(g) Instrument *048 Ensemble de cordes*



(h) Instrument *057 Trombone*

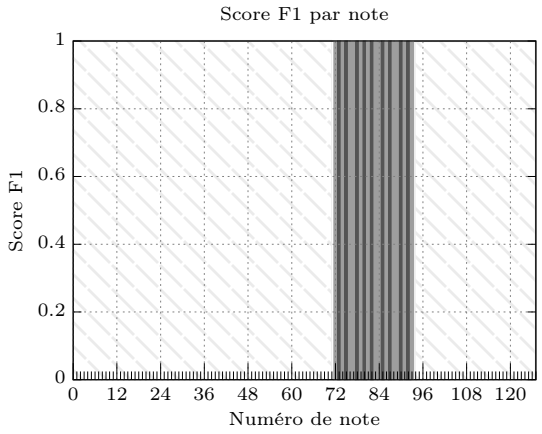


(i) Instrument *064 Saxophone soprano*



(j) Instrument *075 Flûte de pan*

Figure E.2: Époque d'apprentissage monophonique du modèle H$SaConst10(1200 \to 400; 10 \times 400)$ n° 50 (cont.)

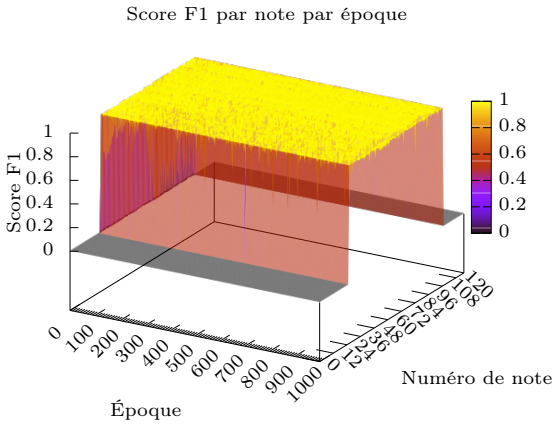


(k) Instrument *080 Signal carré*

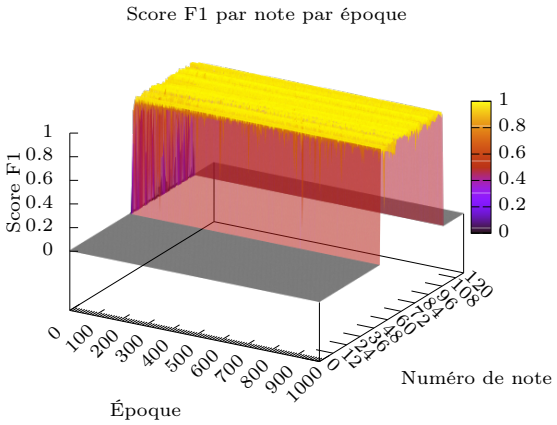
Figure E.2: Époque d'apprentissage monophonique du modèle H$SaConst10(1200 \rightarrow 400; 10 \times 400)$ n° 50 (cont.)

E.3 Séquences d'époques d'apprentissage monophoniques mono-instrumentaux

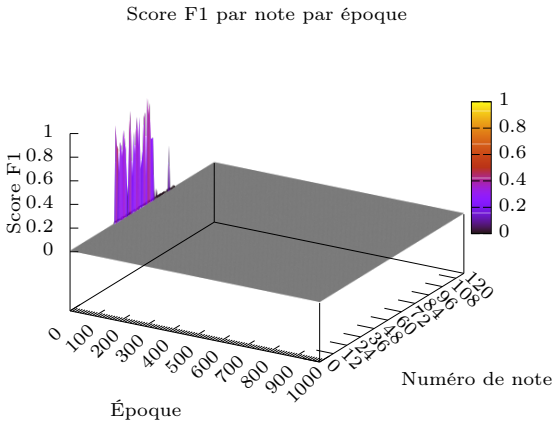
Les sous-figures E.3 représentent le détail complet par note du score F1 pour la séquence d'apprentissage des époques n° 0 à n° 1 000 de l'expérimentation H$SaConst10(1200 \rightarrow 400; 10 \times 400)$, Section 6.3. Les notes sont positionnées selon l'ordre de la norme MIDI, de 0 à 127.



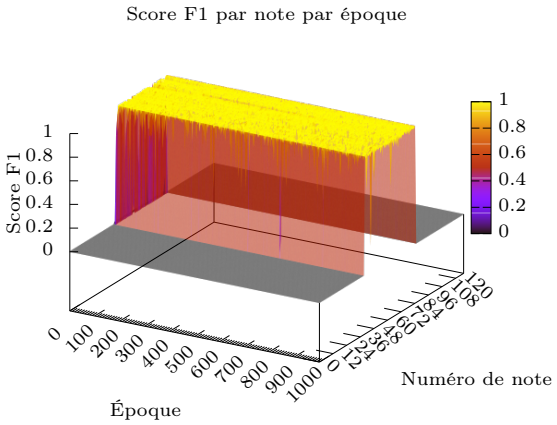
(a) Instrument *000 Piano*.



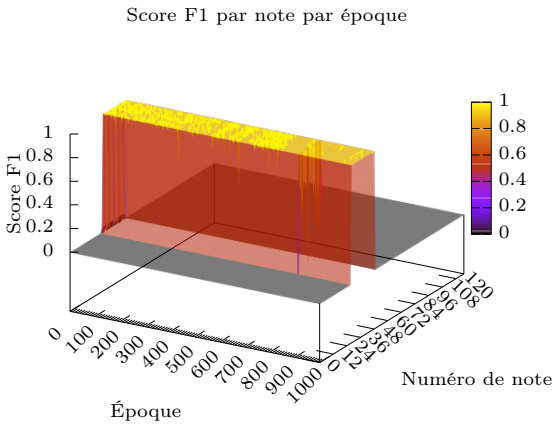
(b) Instrument *013 Xylophone*.



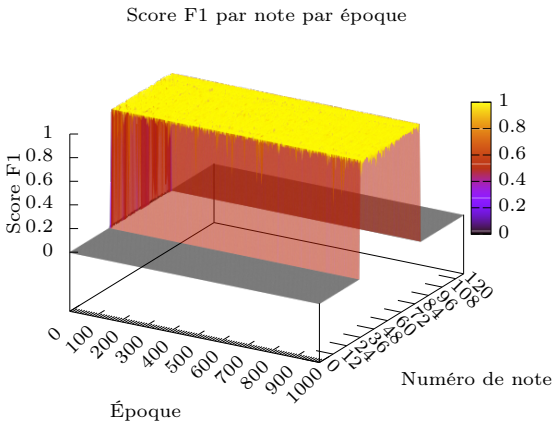
(c) Instrument *019 Orgue d'église*.



(d) Instrument *024 Guitare classique*.

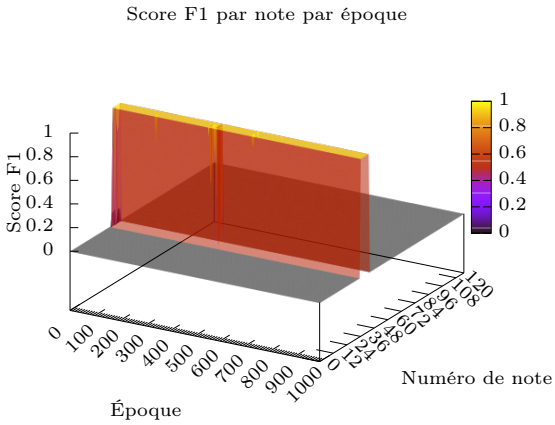


(e) Instrument *032 Basse acoustique*.

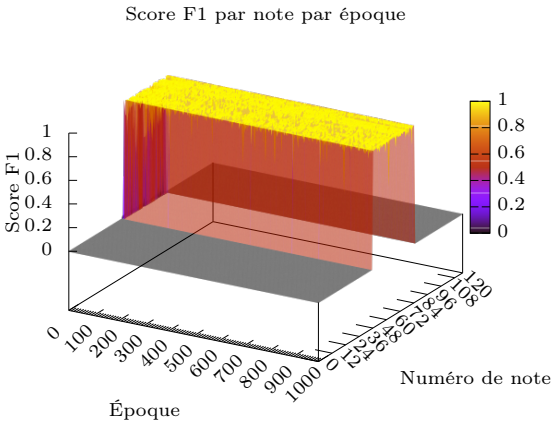


(f) Instrument *040 Violon*.

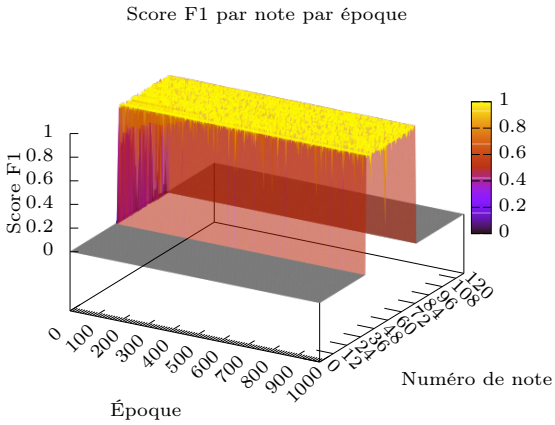
Figure E.3: Séquence d'apprentissage monophonique du modèle H$SaConst10(1200 \rightarrow 400; 10 \times 400)$.



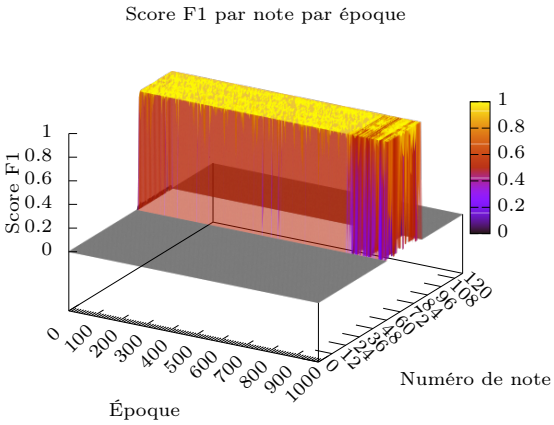
(g) Instrument *048 Ensemble de cordes.*



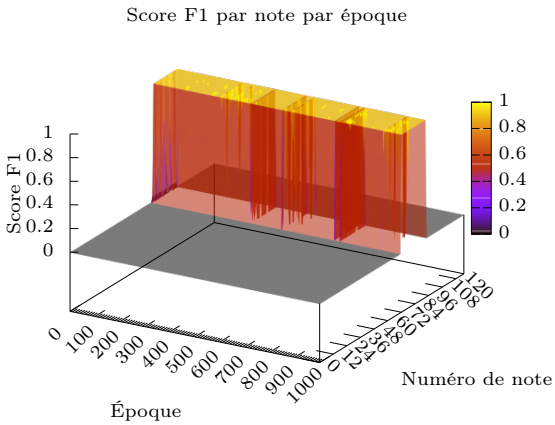
(h) Instrument *057 Trombone.*



(i) Instrument *064 Saxophone soprano.*



(j) Instrument *075 Flûte de pan.*



(k) Instrument *080 Signal carré.*

Figure E.3: Séquence d'apprentissage monophonique du modèle H$SaConst10(1200 \rightarrow 400; 10 \times 400)$ (cont.)

RÉFÉRENCES

Barbedo, J. G. A. et Tzanetakis, G. (2011). Musical instrument classification using individual partials. *IEEE Transactions on Audio, Speech, and Language Processing*, *19*(1), 111–122.

Benetos, E., Ewert, S. et Weyde, T. (2014). Automatic transcription of pitched and unpitched sounds from polyphonic music. Dans *Acoustics, Speech and Signal Processing, 2014 IEEE International Conference on*, 3107–3111. IEEE.

Berenzweig, A. L. et Ellis, D. P. (2001). Locating singing voice segments within music signals. Dans *Applications of Signal Processing to Audio and Acoustics, 2001 IEEE Workshop on the*, 119–122. IEEE.

Bhardwaj, S., Curtin, R. R., Edel, M., Mentekidis, Y. et Sanderson, C. (2018). ensmallen : a flexible C++ library for efficient function optimization. *CoRR*.

Bogdanov, D., Wack, N., Gómez Gutiérrez, E., Gulati, S., Herrera Boyer, P., Mayor, O., Roma Trepat, G., Salamon, J., Zapata González, J. R. et Serra, X. (2013). Essentia : An audio analysis library for music information retrieval. 493–498. International Society for Music Information Retrieval (ISMIR).

Boulanger-Lewandowski, N., Bengio, Y. et Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences : Application to polyphonic music generation and transcription. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, *2*.

Curtin, R. R., Edel, M., Lozhnikov, M., Mentekidis, Y., Ghaisas, S. et Zhang, S. (2018). mlpack 3 : a fast, flexible machine learning library. *Journal of Open Source Software*, *3*(26), 726.

Davis, S. et Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *28*(4), 357–366.

Emiya, V. (2008). *Transcription automatique de la musique de piano*. (Thèse de doctorat). Télécom ParisTech.

Emiya, V., Badeau, R. et David, B. (2010). Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle. *IEEE Transactions on Audio, Speech, and Language Processing*, *18*(6), 1643–1654.

Fujishima, T. (1999). Real-time chord recognition of musical sound : A system using common lisp music. 464–467.

Ganchev, T., Fakotakis, N. et Kokkinakis, G. (2005). Comparative evaluation of various mfcc implementations on the speaker verification task. Dans *Proceedings of the SPECOM*, volume 1, 191–194.

Gómez, E. (2006). Tonal description of polyphonic audio for music content processing. *INFORMS Journal on Computing*, *18*(3), 294–304.

Kelz, R. et Widmer, G. (2017). An experimental analysis of the entanglement problem in neural-network-based music transcription systems. Dans *Audio Engineering Society Conference : 2017 AES International Conference on Semantic Audio*.

Khlif, A. et Sethu, V. (2015). An iterative multi range non-negative matrix factorization algorithm for polyphonic music transcription. Dans *ISMIR - 16th International Society for Music Information Retrieval Conference*, 330–335.

Kimber, D., Wilcox, L. *et al.* (1997). Acoustic segmentation for audio browsers. *Computing Science and Statistics*, 295–304.

Moog, R. A. (1986). Midi : Musical instrument digital interface. *Journal of the Audio Engineering Society*, *34*(5), 394–404.

Moore, B. C. et Glasberg, B. R. (1983). Suggested formulae for calculating auditory-filter bandwidths and excitation patterns. *The Journal of the Acoustical Society of America*, *74*(3), 750–753.

Patterson, R. D., Nimmo-Smith, I., Holdsworth, J. et Rice, P. (1987). An efficient auditory filterbank based on the gammatone function. Dans *a meeting of the IOC Speech Group on Auditory Modelling at RSRE*, volume 2.

Sanderson, C. et Curtin, R. (2016). Armadillo : a template-based C++ library for linear algebra. *Journal of Open Source Software*, *1*(2), 26.

Sanderson, C. et Curtin, R. (2019). Practical sparse matrices in C++ with hybrid storage and template-based expression optimisation. *Mathematical and Computational Applications*, *24*(3), 70.

Shao, Y., Jin, Z., Wang, D. et Srinivasan, S. (2009). An auditory-based feature for robust speech recognition. Dans *Acoustics, Speech and Signal Processing, IEEE International Conference on*, 4625–4628. IEEE.

Sigtia, S., Benetos, E., Boulanger-Lewandowski, N., Weyde, T., Garcez, A. S. d. et Dixon, S. (2015). A hybrid recurrent neural network for music transcription. Dans *Acoustics, Speech and Signal Processing, IEEE International Conference on*, 2061–2065. IEEE.

Sigtia, S., Benetos, E. et Dixon, S. (2016). An end-to-end neural network for polyphonic piano music transcription. *IEEE Transactions on speech and audio processing*, *24*(5), 927–939.

Srinivasan, A., Sullivan, D. et Fujinaga, I. (2002). Recognition of isolated instrument tones by conservatory students. Dans *Proceedings of International Conference on Music Perception and Cognition*, 17–21.

Tzanetakis, G. et Cook, P. (2002). Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, *10*(5), 293–302.

Zhang, T. et Kuo, C.-C. J. (2001). Audio content analysis for online audiovisual data segmentation and classification. *IEEE Transactions on speech and audio processing*, *9*(4), 441–457.