Rapport de Stage

Apprentissage de Structure Musicale par Réseaux Neuronaux Convolutionnels

Auteur :
Alice Cohen-Hadria

Encadré par :
Geoffroy Peeters

Organisme d'accueil :
IRCAM - Équipe
Analyse-Synthèse

Master ATIAM

Année universitaire 2015/2016

Résumé

Ce rapport de 2ème année de Master porte sur l'estimation de structure musicale. Ici, on se restreint à estimer, à partir d'un signal audio de musique, des frontières délimitant différentes parties du signal. Avec l'apparition d'ensembles de données grande échelle, la mise en œuvre d'algorithmes d'apprentissage supervisé est désormais possible. Plus particulièrement, on utilise ici un modèle de réseaux de neurones convolutionnels (ConvNet). La contribution principale de stage est l'utilisation de la profondeur de la couche d'entrée des réseaux et de représentations en matrice d'auto-similarité pour les ConvNets.
Nous effectuons, dans ce document, un état de l'art de l'estimation de la structure musicale. Nous présentons ensuite en détail les réseaux de neurones et les méthodes d'apprentissage associées à ces réseaux. Enfin, les travaux réalisés en stage sont exposés.

Mots-clés : Structure musicale, réseaux de neurones convolutionnels, apprentissage supervisé, optimisation.

Abstract

In this report we study the automatic estimation of musical structure. The goal is here to estimate automatically, from an audio track, the set of boundaries, which circumscribe a music signal's different parts. With the appearance of large annotated dataset, it is now possible to use supervised learning algorithms to deal with this problem. We study here the recent proposal to use neural network, and more specifically the convolutional model. Our main contribution is the use of the depth of the input layer and new signal's representation as input of neural network.
We first present the state of the art related to the estimation of music structure. We then describe, in details, the neural model, and the convolutional neural networks. Finally, we expose our contributions during this internship.

Mots-clés : musical structuration, convolutional neural networks, supervised learning, gradient descent optimization.

Remerciements

Je tiens à remercier mon encadrant Geoffroy Peeters, tout d'abord pour m'avoir permis de réaliser ce stage sur un sujet qui me passionne réellement. Ensuite, pour son encadrement, sa patience et sa gentillesse tout au long de mon travail de stage. Enfin, pour la confiance qu'il m'accorde pour poursuivre ces travaux en thèse sous sa direction.

Je tiens également à remercier tous mes camarades de promotion ATIAM, et plus particulièrement ceux avec qui j'ai pu passer mes six mois de stage au sein de l'IRCAM : Hugo, Maxime, Théis, Guillaume et Tristan. Merci également aux doctorants de l'équipe, Luc, Céline, Damien (pour leur soutien le vendredi soir) et surtout Ugo, qui fut un co-bureau fantastique. Je leur souhaite à tous toute la réussite qu'ils méritent dans leur future carrière académique.

Enfin, merci à ma famille et à mes amis Steven, Matthieu, Vincent, Marie, Cynthia et plus particulièrement Ali, qui, grâce à son expérience de *data scientist*, m'a beaucoup aidé pendant ce travail de stage. Pour finir, merci à Ghiles, pour son soutien.

Table des matières

1 Introduction

Ce stage se situe dans le contexte de l'estimation de la structure temporelle d'un morceau de musique (couplet, refrain, pont) à partir de l'analyse du signal audio. Plus précisément, nous nous intéressons à l'estimation des frontières temporelles (pas aux labels assignés aux segments).
Pour estimer cette structure, on commence, dans un premier temps, par extraire du signal audio des descripteurs (généralement des MFCC ou des chromas), qui serviront dans un second temps à construire une nouvelle représentation de la piste étudiée. Deux paradigmes existent pour créer cette représentation : 1) l'approche par états, où l'on considère qu'un signal sonore est une succession de blocs homogènes formant des états représentant la structure de la pièce, et 2) l'approche par séquences, où un signal est considéré comme résultant de répétitions de séquences temporelles partageant des propriétés musicales communes. Cette représentation servira alors de point de départ à différentes méthodes d'extractions d'informations.
Récemment, des ensembles de données annotées en structure musicale sont apparues, permettant dés lors d'utiliser des méthodes d'apprentissage supervisé, dont les réseaux neuronaux. C'est cette proposition que nous étudions dans le cadre de ce stage. Nous étudions également la question de la représentation des données servant à l'apprentissage de tels réseaux.

Cadre Ce stage a eu lieu du 1 février au 31 juillet 2016, sous l'encadrement de Geoffroy Petters, au sein de l'équipe Analyse/Synthèse du laboratoire Science et Technique de la Musique et du Sons (STMS), associant le CNRS, l'UPMC, le ministère de la Culture et l'IRCAM autour d'une thématique de recherche interdisciplinaire sur les sciences et technologies de la musique et du son. Ce travail de stage s'inscrit dans le travail de recherche d'informations musicales *(Music Information (MIR))* de l'équipe, et plus particulièrement de la proposition récente d'utiliser les réseaux de neurones pour l'estimation de structure musicale.

Objectifs Le premier objectif de ce stage est d'étudier la proposition de [Ullrich et al., 2014] d'utiliser les réseaux de neurones convolutionnels pour l'estimation de structure. Une étude bibliographique précise de ce sujet est donc nécessaire. Comprendre précisément ces modèles et leur apprentissage, étudier les résultats donnés par ces apprentissages font partie intégrante de notre travail dans le cadre du stage. L'examen de différentes représentations formera, dans un deuxième temps, la suite de notre travail.

Organisation du rapport Dans la partie 2 nous rappelons les méthodes principales utilisées correspondantes aux approches par états et par séquences.
Dans la partie 3, nous présentons l'ensemble de données utilisé dans le cadre de ce travail.
Dans la partie 4, nous expliquons, en détail, le fonctionnement des réseaux de neurones convolutionnels.
Dans la partie 5, nous proposons une explication des différentes méthodes d'optimisation mises en œuvre dans l'apprentissage des réseaux de neurones.
Nous présentons ensuite dans la partie 6 la méthode de [Ullrich et al., 2014] pour l'estimation de structure par réseaux de neurones convolutionnels de laquelle nous sommes partis.
Enfin, dans la partie 7, nous exposons nos contributions aux méthodes existantes.

2 État de l'art : Découverte de structure musicale

Notre travail s'inscrit dans le domaine de la découverte de structure musicale (*Music Structure Discovery*). Bien que la définition de structure musicale reste sujet à discussion (celle-ci peut varier en fonction de son paradigme d'utilisation, de l'annotateur ; voir par exemple les travaux [Peeters and Deruty, 2009], [Bimbot et al., 2012] et [Smith et al., 2011]), nous considérons ici celle-ci comme définie par un ensemble d'annotation. Dans notre travail, nous nous intéressons uniquement à la détection des frontières délimitant les segments formant la structure. Ces frontières correspondent à des 'instants de césure' dans le morceau. Ces instants délimitent alors des segments pouvant être étiquetés a posteriori [Paulus and Klapuri, 2009].

2.1 Représentation du signal

Tout d'abord se pose la question de la représentation du contenu du signal audio. Pour cela, on extrait généralement des descripteurs audio. Parmi les nombreux descripteurs existant (voir [Peeters, 2004a]), deux sont généralement utilisés pour l'estimation de structure :

- **MFCCs** : Les Mel-Frequency Cepstral Coeffcients (MFCC) décrivent l'enveloppe spectrale et donnent donc une indication sur le timbre du signal. Les MFCCs découlent du cepstre. Ils reposent sur la transformée en cosinus discrète du logarithme du mel-spectre de puissance. On utilise généralement uniquement les coefficients basses quéfrences (les 12 ou 13 premiers coefficients).

- **Chromas** : Les Chromas (ou Pitch Class Profile) fournissent une représentation compacte de l'importance des classes de hauteurs présentes dans le signal autour d'un instant donné. Un vecteur de chroma représente la contribution de chacune des 12 classes de hauteurs de notes possibles [1] (en ignorant l'octaviation) ou la distribution de l'énergie suivant ces 12 classes. Ils sont généralement calculés par filtrage d'un spectrogramme, par bancs de filtres de bande passante d'un demi-tons centrés sur chaque hauteur possible.

Matrice d'auto-similarité Muni d'une mesure de distance (euclidienne, cosinusoïdale, divergence symétrisée de Kullback–Leibler, ...), on peut alors calculer la similarité entre chaque couple de vecteurs de descripteurs représentant deux instants t_i et t_j. L'ensemble de ces similarités peut être représenté sous forme d'une matrice d'auto-similarité (Self-Similarity Matrix ou SSM) S, telle que :

$$S(t_i, t_j) = d(v_{t_i}, v_{t_j})$$

où d est une mesure de distance.

1. On se place ici principalement dans le cadre de la musique occidentale savante ou populaire, où l'on utilise généralement un découpage du ton en demi-tons.

2.2 Méthodes d'estimation de frontières

Ces représentations (MFCC, Chroma ou SSM) servent ensuite à l'estimation des frontières de structure. Pour cela, on constate une rupture chronologique dans les méthodes proposées correspondant à l'apparition de larges ensembles de données annotés en structure. Avant l'apparition de ces bases, l'estimation était effectuée de manière non-supervisée (utilisation des noyaux de convolution en damier de [Foote, 2000], résultante d'un apprentissage non-supervisé par HMM [Peeters et al., 2002], *structuring features* de [Serrà et al., 2012]). Progressivement, de larges ensembles de données annotés sont apparus : RWC ([Goto, 2006]), Isophonics ([Mauch et al., 2009]), SALAMI ([Smith et al., 2011]) et Quaero ([Bimbot et al., 2011]). L'estimation de ces frontières peut dès lors être transformée en un problème d'apprentissage supervisé. Du fait de l'apparition de larges ensembles de données annotés en structure différentes techniques supervisées sont utilisées (LDA ([McFee and Ellis, 2014]), ConvNet ([Ullrich et al., 2014])).

Dans cette partie, nous résumons les principales approches utilisées dans ces deux domaines.

Ces approches non-supervisées peuvent être divisées selon les deux paradigmes proposés dans [Peeters, 2004b] : l'approche par état et l'approche par séquence.

2.2.1 Approche non-supervisée par état

On considère ici qu'un signal sonore est une succession de blocs homogènes, formant des états. Une césure est alors l'instant de la fin d'un état et du début d'un autre. Cette approche inclut les approches par nouveauté, ou l'on cherche des marqueurs de transitions entre états contrastés, et les approches par homogénéité, où l'on cherche à déterminer des passages partageant des propriétés musicales communes.

[Foote, 2000] propose d'appliquer un noyau de convolution le long de la diagonale de la SSM obtenue à partir des MFCCs du signal. Celle-ci fournit une fonction de nouveauté. Le noyau de convolution est en forme d'échiquier dont le noyau unitaire s'exprime :

$$C = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Le premier terme mesure l'auto-similarité des segments de part et d'autres de l'instant considéré, le second leur dissimilarité respective. La différence de ces deux termes donne alors la nouveauté du signal au point central.
Avec un produit de Kronecker, on peut facilement augmenter la taille du noyau. On détecte ainsi des nouveautés de granularités plus ou moins élevées (avec un petit noyau, on distingue par exemple les attaques des notes ; avec un noyau de taille plus importante, des phrases musicales, ou des parties entières). Une hybridation de cette méthode et d'une approche par séquence est proposée dans [Kaiser and Peeters, 2013], où l'on met en place des noyaux de corrélation détectant des césures entre blocs homogènes et inhomogènes (entre un état et une séquence).

Dans la méthode [Peeters et al., 2002], la segmentation temporelle est obtenue comme résultat de la labélisation temporelle des trames. Dans cette approche, les descripteurs audio sont donnés en entrée de chaînes de Markov cachées (Hidden Markov Model, HMM). Le décodage du HMM permet d'assigner un état à chaque instant. La variation temporelle des états fournit les frontières. Le HMM est lui-même initialisé par un algorithme de clustering

(type K-Means). Cet algorithme ne prend pas en compte les relations temporelles entre les trames, i.e il ne considère pas les états comme une succession d'événements temporellement ordonnés, un raffinement du résultat par HMM est donc nécessaire. Le HMM amène un lissage du résultat. Les descripteurs audio utilisés dans cette méthode sont les *"dynamic features"* (aujourd'hui appelé modulation spectrum).

2.2.2 Approche non-supervisée par séquence

Dans cette approche, on ne considère plus les segments comme des blocs homogènes, mais comme des séquences répétées selon certaines propriétés musicales dans le temps. Ces répétitions sont figurées dans la SSM par des diagonales parallèles à la diagonale principales.

L'approche proposée par [Goto, 2003] repose sur l'expression de la SSM non pas en matrice temps-temps mais en matrice temps-lag. Si S est la matrice de similarité, la matrice de lag R se calcule comme :

$$R(i, i-j) = S(i, j), \text{ pour } i - j > 0$$

Dans cette matrice, une répétition à décalage constant s'exprime par une ligne verticale. Cette propriété est utilisée par l'auteur pour détecter les segments répétés. Dans un premier temps, les lags contenant préférentiellement des répétitions sont localisés par détection de pics sur la somme à travers les temps. Pour chacun des candidats lags, la colonne correspondante de la matrice est analysée pour détecter le début et la fin du segment. Dans le cas de [Goto, 2003], la matrice est calculée à partir des Chromas. L'auteur cherche à localiser le refrain d'un enregistrement de musique populaire. L'auteur propose également, afin de s'abstraire des modulations de refrain [1], de calculer les mesures de similarité sur toutes les transpositions possibles (12) des vecteurs de chroma. On obtient, après analyse et avec certains a priori [2], les instants de début et fin du refrain mais aussi des parties considérées comme identiques qui ne sont pas forcément le refrain (le couplet par exemple).

[Muller et al., 2013] propose de formuler le problème de l'extraction de ces diagonales comme un problème d'optimisation résolu par programmation dynamique avec une adaptation de l'algorithme de *Dynamic Time Warping* (DTW). Celui-ci permet l'estimation du segment optimal tel que ses répétitions expliquent au mieux le contenu de la matrice SSM. Les chemins reliant le segment et ses répétitions dans la SSM sont appelés famille de chemins optimaux (*optimal path family*). Pour cela, on assigne une mesure d'adéquation (*fitness*) à chaque segment candidat. Celle-ci est la combinaison d'une métrique de longueur de segments sélectionnés et d'une estimation de recouvrement (ou de *coverage*) de la piste. Les instants de plus forte *fitness* sont alors les instants les plus représentatifs du signal.

[Serrà et al., 2012] se concentre sur l'estimation des frontières de structure. Comme Goto, la méthode repose sur le traitement de la matrix de lag. Alors que Goto effectue une somme à travers les temps pour localiser les candidats lags de répétitions, les auteurs proposent ici l'utilisation d'une matrice de lag symétrisée et utilisent la variation à travers le temps des

1. Il est fréquent, en musique populaire, de retrouver le refrain transposé d'un ou deux tons.
2. Par exemple, on considère qu'un segment en fin d'enregistrement, si il présente de fortes similarité avec d'autres, a de grandes chances d'être le refrain.

lignes de cette matrice (appelée structuring features). Lorsqu'un segment répété commence dans un morceau, toutes ces répétitions démarreront au même instant dans cette matrice.

Finalement, dans [Bisot, 2014],la méthode de [Muller et al., 2013] est étendue à l'estimation de l'ensemble de la structure (et non-seulement du segment le plus répété). La méthode [Serrà et al., 2012] est également étendue en y incluant la méthode de Goto utilisée comme probabilité a priori des lag-dimensions sur le vecteur *structuring feature*.

2.2.3 Approche supervisée

Avec l'apparition d'ensembles de données grande échelle, on peut désormais mettre en place des méthodes d'apprentissage supervisé. Le principal ensemble est *Structural Analysis of Large Amounts of Musical Information* (SALAMI), proposé dans [Smith et al., 2011], que nous utilisons dans le cadre ce travail [1].

[McFee and Ellis, 2014] propose un algorithme supervisé visant à adapter automatiquement les *features* acoustiques et structurelles d'un ensemble d'entrainement utilisées, soit trouver une transformation linéaire optimale des *features* afin de mettre en avant une structure du morceau.
L'idée est ici de donner un poids à chacune des *features* en multipliant X (matrice construite en empilant différentes *features*) par une matrice W, décrivant celle-ci, et telle qu'elle maximise la distance entre les segments tout en minimisant la variance de chacun, avec l'algorithme de Fisher-Linear-Discriminant-Analysis relâché (apprentissage supervisé). On effectue ensuite un clustering sur le produit $W^T X$.

Finalement, Schlüter et Grill proposent dans [Ullrich et al., 2014], [Grill and Schlüter, 2015] et [Schluter and Bock, 2014] d'utiliser des réseaux neuronaux convolutionnels pour détecter des instants de césure. C'est cette proposition qui constitue le point de départ de notre travail dans le cadre de ce stage. Nous présentons donc plus en détail cette méthode dans la section 6.

Ainsi, nous nous intéressons dans le cadre de ce travail de stage aux méthodes d'apprentissage supervisé. Pour les mettre en œuvre, il faut disposer d'un ensemble de données annotés. Nous présentons dans la section suivante l'ensemble de données que nous avons utilisé.

1. Cf section 3

3 Ensemble de données SALAMI

Un des ensembles de données de référence dans le cadre la structure musicale est la base *Structural Analysis of Large Amounts of Musical Information* (SALAMI, voir [Smith et al., 2011]). Créée en 2011, elle possède plus de 1400 titres et 2400 annotations correspondantes. Cette base répond à l'idée d'une analyse de structure d'une grande variété de musique. C'est pourquoi SALAMI possède, en part presque égale, de la musique classique, jazz, populaire et du monde (voir figure 1).

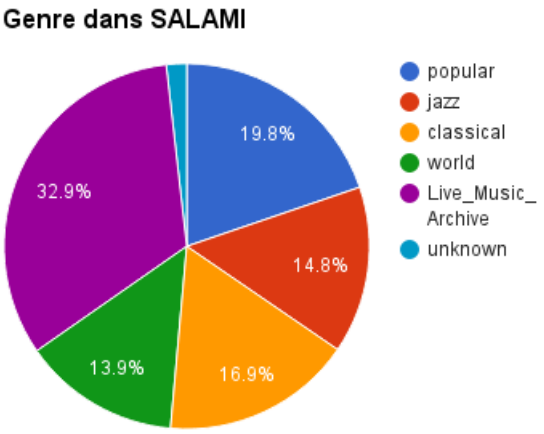

FIGURE 1 – Répartition des genres dans SALAMI

Cet ensemble de données a été construit comme une sous partie d'un autre ensemble de données plus grand, Codaich (voir [Mckay, 2006]).

Les annotations Le format des annotations de SALAMI est basé sur la sémantique proposée dans [Peeters and Deruty, 2009]. On distingue trois labels figurant trois descriptions différentes de la structure : similarité musicale, fonction [1] et instrumentation. Différents niveaux de granularité sont adoptés pour l'annotation de similarité.

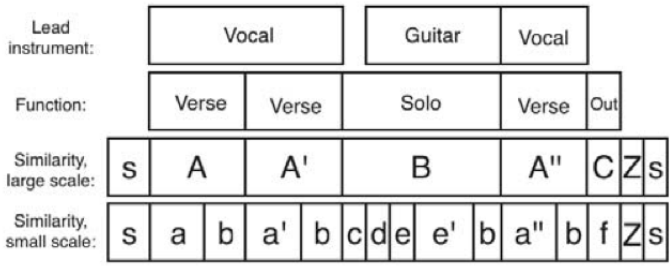

FIGURE 2 – Exemple de format d'annotations, tiré de [Smith et al., 2011]

1. On limitera également le vocabulaire associé aux différentes fonctions.

Les annotations ont été réalisées par huit étudiants en musicologie ou en composition, qui ont été entrainés pendant 6 semaines à la tache d'annotation.
Dans le cadre de ce travail, nous avons utilisé uniquement les informations de frontières, soit les temps auxquels étaient annotés les césures de fonction. Nous n'utiliserons qu'une seule granularité et ne prêterons pas attention aux labels de ces parties. Nous n'avons, dans le cadre de ce travail, à disposition que 700 titres parmi les 1400.
La prédiction à effectuer ici concernera donc uniquement de instants de césure dans la piste.

Ainsi, nous possédons des données annotées pour l'estimation de structure, variées et fiables. Pour l'estimation de ces structures, nous utilisons les réseaux de neurones convolutifs. Nous présentons ce modèle dans la partie suivante.

4 Les réseaux de neurones

4.1 Un neurone

Le modèle du neurone informatique, ou neurone formel, s'inspire d'un fonctionnement simplifié d'un neurone biologique. La modélisation simplifiée de McCulloch et Pitts datant de 1943 le conçoit comme recevant, d'autres neurones, des impulsions électriques de poids variables par ses dendrites. En passant dans le neurone, et si la somme des impulsions dépasse un certain seuil, l'influx électrique reçu est transformé, dans le corps du neurone, en une nouvelle impulsion électrique, que l'on peut modéliser comme un 1 ou un 0 (activation ou non activation). Cette activation est ensuite transmise par l'axone aux neurones connectés à celui-ci.
On construit, en se calquant sur cette modélisation, un neurone formel avec :

- Les **entrées** x du neurone, se présentant comme un vecteur de dimension N.
- Un ensemble de **poids** W, modélisant les poids variables des différentes entrées. W est un vecteur de dimension N également. À chaque entrée correspond un poids.
- Un **biais** b représentant le seuil d'activation du neurone.
- La somme a des N "impulsions électrique" reçues est donc obtenue par :

$$a = \sum_{n=1}^{N} W_n x_n + b$$

- Une **fonction d'activation** h, rendant compte de l'influx électrique de sortie, appliquée à la somme des entrées pondérées par W et du biais b. On utilise en général la fonction sigmoïde [1] :

$$\text{sigmoïde}(a) = \frac{1}{1 + e^{-a}}$$

  Également utilisée, la fonction de tangente hyperbolique est en réalité une translation de la fonction sigmoïde telle qu'elle soit centrée en 0.
  L'inconvénient d'utiliser de telles fonctions d'activation est le problème de saturation. En effet, en $+\infty$ et en $-\infty$, la fonction sigmoïde possède deux asymptotes : $y = 0$ et $y = 1$. Ainsi, plus l'entrée de cette fonction devient grande en valeur absolue, plus la valeur de la sigmoïde tend vers une valeur constante. Sa dérivée sera donc nulle. Or, on verra dans la partie 5 que ce gradient de la fonction d'activation est utile à l'apprentissage, et que son annulation pose problème.
  On peut donc, pour palier à ce problème utiliser une fonction d'activation différente comme la (*Rectified Linear Unit*, ReLU), qui s'écrit comme :

$$\text{ReLU}(a) = \max(0, a)$$

Muni de ces trois composants, on peut alors modéliser un neurone comme exposé en figure 3. Un neurone formel est aussi appelé un perceptron. D'un point de vue géométrique, l'équation d'un perceptron représente un hyperplan dans l'espace des entrées. C'est donc un algorithme de classification linéaire.

1. En particulier pour ses propriétés de dérivation : sigmoïde'(a) = sigmoïde(a) + (1 − sigmoïde(a)). En outre, cette fonction a l'avantage d'être bornée par 0 et 1, e qui la rend interprétable comme une probabilité.

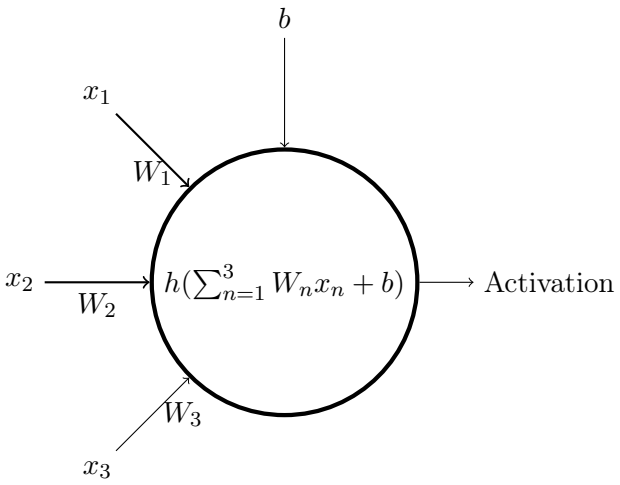

FIGURE 3 – Schéma d'un neurone à 3 entrées.

4.2 Perceptron multi-couches

Afin de complexifier la frontière de décision d'un perceptron unique, on peut alors former un ensemble de perceptron inter-connectés, appelé réseau de neurone ou perceptron multicouche. Une couche d'un tel réseau est constitué de N neurones, produisant N activations. Ces sorties peuvent devenir alors de nouvelles entrées pour une nouvelle couche de neurones.

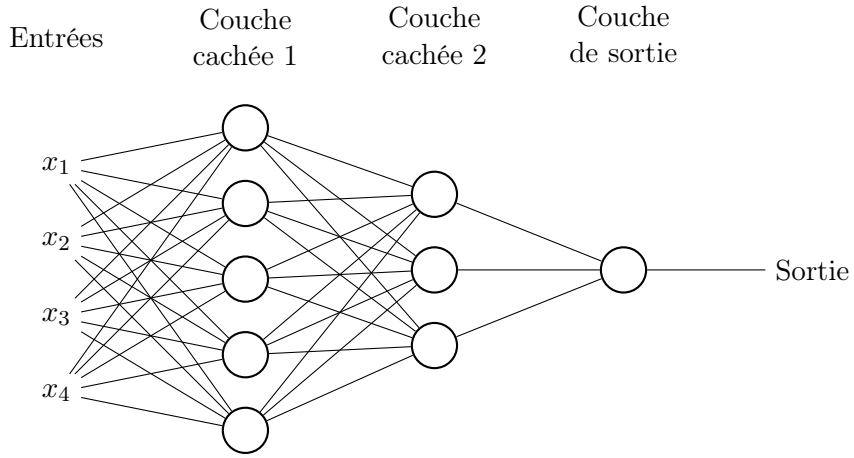

FIGURE 4 – Exemple d'un réseau neurone à deux couches cachées.

Les couches à l'intérieur du réseau sont appelées couches cachées, ou couches fortement connectées (*fully connected*).

4.3 Les réseaux de neurones convolutionnels

Ce type de réseau est inspiré du fonctionnement biologique du cortex visuel. Les prémices de ceux-ci datent de 1980 avec [Fukushima, 1980] et le neo-cognitron, ancêtre des réseaux de neurones convolutionnels. [Lecun et al., 1998] se base sur ce modèle, en proposant le premier exemple de réseaux neuronaux convolutionnels (ou Convolutionnal Neural Networks, CNN ou ConvNet), pour la reconnaissance de l'écriture manuscrite.

4.3.1 Bloc convolutionnel

Un réseau de neurones convolutionnel est un modèle proche d'un perceptron multi-couche. Ils sont composés de neurones organisés en couche. Ces neurones ont des poids, des biais et une fonctions d'activation. Le réseau en lui-même possède également une fonction de coût. La principale différence est, qu'avec les ConvNets, on peut prendre en compte le fait que l'entrée soit une image et en exploiter les propriétés intrinsèques. Les ConvNets prennent en effet en entrée une matrice ou un tenseur [1].

Interprétation neuronale Les neurones des couches convolutionnelles possèdent des propriétés dérivées de celle du cortex visuel. Dans celui-ci, on trouve des neurones sensibles chacun à une région réduite du champ visuel. Chacune de ces cellules agit comme un détecteur d'une certaine caractéristique du champ visuel, et ce quelque soit l'endroit où cette caractéristique est présente, permettant d'exploiter la forte corrélation locale présente dans les images.
Pour répliquer ce comportement, les neurones d'une couche convolutionnelle possèdent des poids en commun, répliqués sur l'ensemble de l'image (voir figure 5). On appelle ces poids, poids partagés (*shared weigths*).

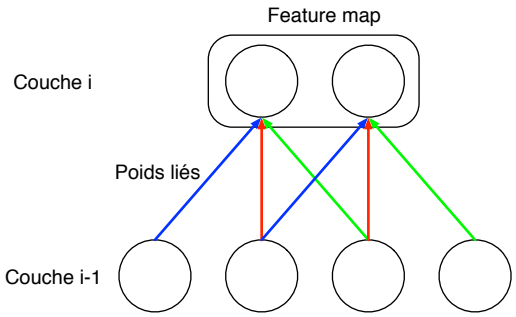

FIGURE 5 – Exemple de poids partagés (les poids de la même couleur sont identiques)

Chaque groupe de neurones est ainsi répliqué à travers toute l'image et forme un détecteur d'un motif particulier de l'image. Plusieurs de ces détecteurs forment alors un volume de neurone et une couche convolutionnelle (voir figure 6).

1. Dans le cas d'images en couleur, on a par exemple un tenseur de profondeur 3, représentant les trois composantes RGB de l'image.

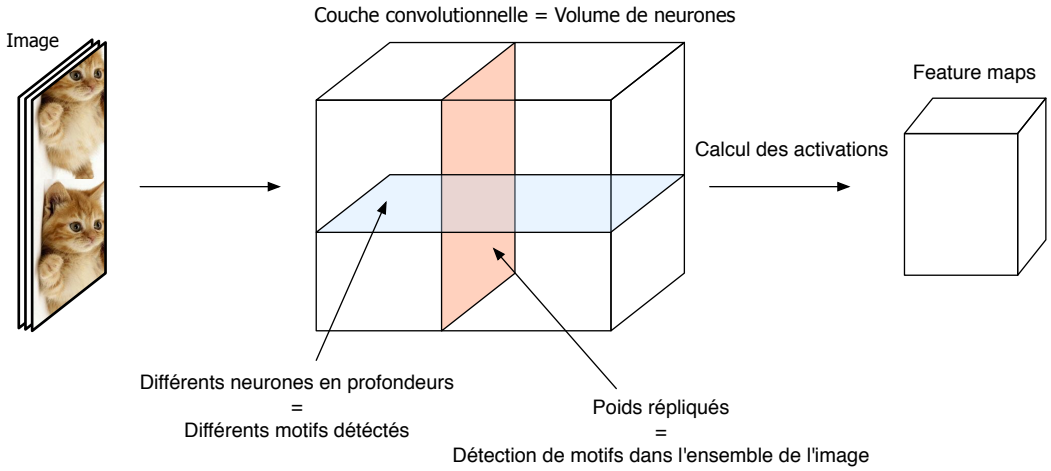

FIGURE 6 – Couche convolutionnelle - interprétation neuronale

Interprétation avec filtres Chaque surface (en rouge dans la figure 6) du volume présenté plus haut peut donc être considérée comme la réplication d'un filtre. Les poids liés de nos neurones deviennent les coefficients d'un filtre, que l'on convolue avec l'image d'entrée.

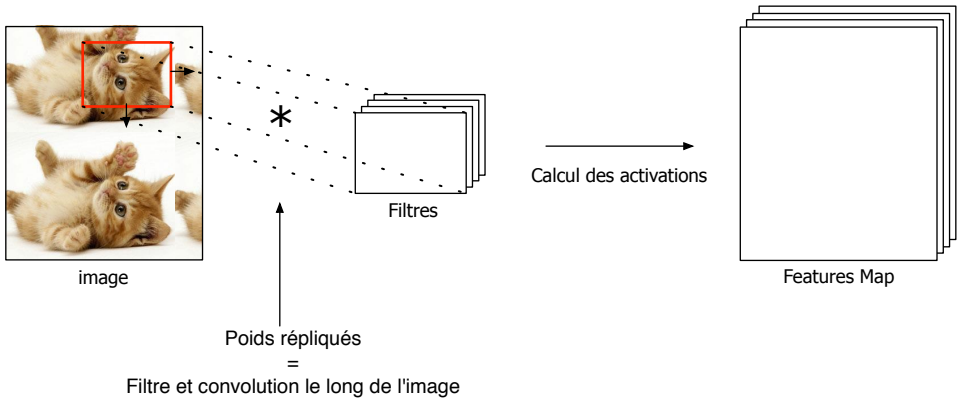

FIGURE 7 – Couche convolutionnelle - interprétation avec filtres

Formellement, on pose y^k la sortie du k-ième filtre de notre bloc convolutionnel, dont W^k est la matrice des poids du filtre, et b^k le biais. On exprime alors y^k comme :

$$y^k_{ij} = h((W^k \star x)_{ij} + b^k)$$

Considérations géométriques Plaçons nous dans le cadre général avec une couche convolutionnelle prenant en entrée un tenseur $\mathbf{x}$ de dimension $H \times W \times D$. Nous allons convoluer ce tenseur par D'' filtres de taille $H' \times W' \times D$. Le tenseur de filtres $\mathbf{f}$ est donc un tenseur de dimension 4. Le résultat de la convolution de $\mathbf{x}$ par $\mathbf{f}$ est également un tenseur $\mathbf{y}$ de dimension $H'' \times W'' \times D''$. H'' et W'' sont fonction de H, W, W' et H' mais également de deux autres hyper-paramètres des ConvNets : le pas de la convolution (le *stride*) et le *zero padding* effectué sur l'entrée.

Stride et zero-padding Le *stride* contrôle le chevauchement des convolutions de l'entrée et donc la dimension de la sortie. Si le *stride* est unitaire, le chevauchement est important, ce qui conduit à une grande dimensionalité de la sortie. En augmentant celui-ci, on réduit le chevauchement et la dimension de la sortie.
Le *zero-padding* est effectué sur les bordures de l'image, afin d'en contrôler la taille et celle de la sortie de la couche.
Dans notre cas, nous utilisons un *stride* unitaire et un *zero padding* nul. La taille de la sortie $\mathbf{y}$ est donc $H'' \times W'' \times D''$ avec $H'' = 1 + H - H'$ et $W'' = 1 + W - W'$.

4.3.2 Pooling

Après un bloc convolutionnel, il est fréquent de retrouver un bloc de *pooling*. Paramétré par deux entiers n et m, il permet d'extraire une quantité des sous matrices de tailles $m \times n$ de l'entrée, en fonction d'un certain pas d'avancement. C'est en général une opération de *max-pooling* qui est effectuée (voir figure 8).

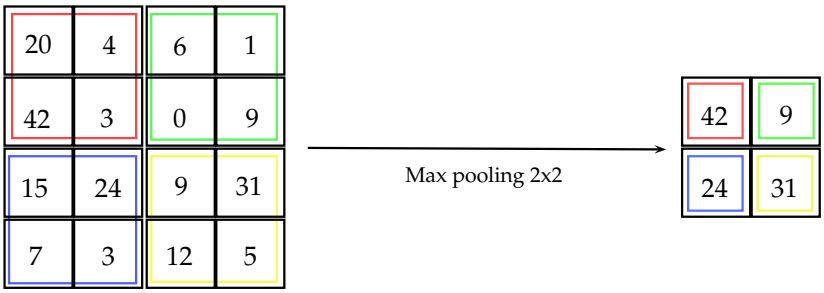

FIGURE 8 – Exemple de max pooling (2x2), pas de deux

Ce bloc est particulièrement utile pour deux raisons. Premièrement, il permet une réduction de la dimensionalité des données à l'intérieur du réseau, et donc une réduction du nombre de paramètres appris, endiguant ainsi le sur-apprentissage.
De plus, cette opération induit une forme d'invariance par translation des données, rendant le ConvNet plus robuste à la détection de motifs particuliers dans les entrées.

Ainsi, une couche de ConvNet se présente comme en figure 9, avec une alternance de couche convolutionnelle et de max-pooling.

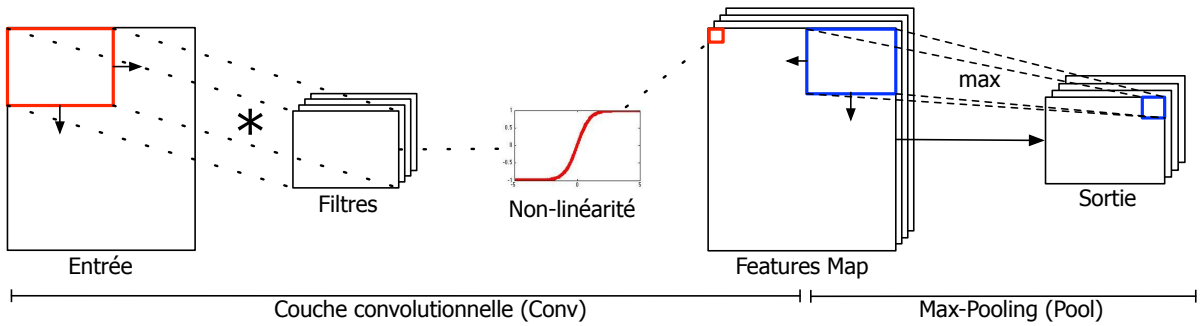

FIGURE 9 – Bloc convolutionnel et Max-Pooling

Pour construire un réseau, il suffit de connecter les sorties d'un bloc aux entrées d'un autre. Une couche fortement connectée est traditionnellement ajouté en fin de réseau, couplé avec un algorithme de classification linéaire (dans le cadre d'un problème de classification), ou d'une régression linéaire (dans le cadre d'un problème de régression). Dans notre cas d'estimation de structures, nous faisons en sorte d'avoir en sortie de notre réseaux une probabilité de césure [1] (voir partie 6).

Après avoir construit notre architecture de réseau, il faut ensuite apprendre ses poids. La partie suivante présente les différents algorithmes d'apprentissage actuels.

1. 1 figurant une césure à l'instant considéré, 0 pas de césure.

5 L'apprentissage des réseaux de neurones

5.1 La descente de gradient

L'un des buts d'un réseau de neurones est de pouvoir classifier des données. Il est cependant difficile d'imaginer trouver les différents paramètres de notre réseau permettant cette classification de manière purement algébrique. Nous allons donc entraîner le réseau de manière itérative sur un ensemble de données étiquetés, nous plaçant donc dans le cadre de l'apprentissage supervisé. L'idée est de modifier itérativement les poids W (ou, ici, les coefficients de nos filtres) pour réduire l'erreur de notre algorithme de classification par rapport à une sortie attendue.

5.1.1 Fonction d'erreur

On commence tout d'abord par définir l'erreur faite par le réseau. Traditionnellement, on choisit l'erreur des moindres carrés. Soit y la sortie attendue (l'étiquette du jeu de données), x l'entrée associée et $h(x)$ la sortie de notre réseau de neurones. $\mathcal{L}(x, y)$, l'erreur associée à cette donnée est définie comme :

$$\mathcal{L}(x, y) = \frac{1}{2}(h(x) - y)^2.$$

On peut également, comme c'est le cas pour [Ullrich et al., 2014], utiliser l'entropie croisée binaire (ou *binary cross-entropy*) :

$$\mathcal{L}(x, y) = y \ln(h(x)) + (1 - y) \ln(1 - h(x)).$$

A noter qu'il existe de nombreuse fonction d'erreur (aussi appelées fonction de coût) à choisir en fonction du problème considéré. Sur l'ensemble du jeu de données, on a, pour M exemples :

$$\mathcal{L} = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}(x^{(m)}, y^{(m)}),$$

la moyenne de l'erreur.

5.1.2 Gradient

La descente de Gradient C'est avec cette erreur que nous allons pouvoir modifier les poids de notre réseau, en les mettant à jour de manière à diminuer l'erreur faite par notre réseau de neurones. Soit un réseau à K couches et $\theta^k = \{W^k, b^k\}$ les poids associés à la $k$ième couche. La mise à jour des poids θ^k par descente de gradient est faite en retranchant une fraction du gradient de l'erreur par rapport à ces paramètres, soit :

$$\theta_{ij}^k \leftarrow \theta_{ij}^k - \alpha \frac{\partial \mathcal{L}}{\partial \theta_{i,j}^k},$$

avec α le taux d'apprentissage. En retranchant, à chaque itération, une fraction du gradient de l'erreur, on converge alors vers un minimum de la fonction d'erreur $\mathcal{L}$. L'optimalité de ce minimum n'est pas garantie, on peut donc converger uniquement vers un minimum local.

Interprétation géométrique Du point de vu géométrique, en se plaçant dans l'espace des paramètres, la fonction d'erreur est un hyperplan, qui possède des creux et des bosses, en fonction de la valeur des paramètres. La figure 10 présente un exemple en deux dimensions (un seul paramètre à optimiser).

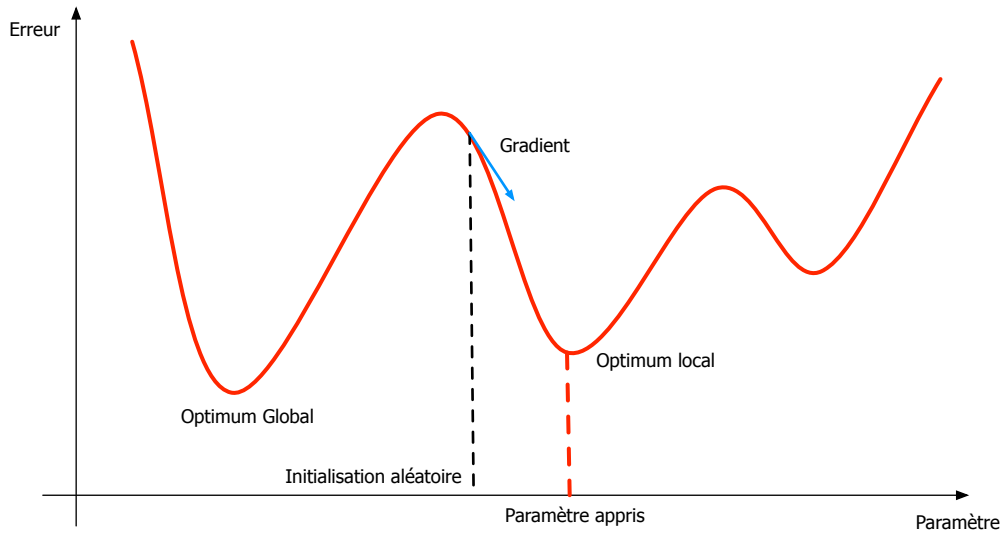

FIGURE 10 – Exemple de descente de gradient à un paramètre

L'initialisation aléatoire des paramètres nous place à un endroit de cette surface, c'est-à-dire à une valeur initiale de paramètres et d'erreur correspondante. La descente de gradient doit son nom au fait que, à partir de ce point initial, on va "descendre" le long de la plus "forte" pente de notre hyperplan d'erreur, le gradient indiquant cette direction. À la manière d'une bille que l'on pousse du haut d'une colline, l'erreur va descendre pour atteindre un minimum local de notre fonction d'erreur.
Après cette procédure, les poids θ^k sont appris tels qu'ils minimisent l'erreur faite sur le jeu de données d'entrainement, l'espoir étant que cet apprentissage est généralisable et possède de bonnes propriétés sur un jeu de test également.

Variantes La descente de gradient telle qu'elle est décrite ici pose $\mathcal{L}$ comme étant l'erreur calculée sur l'ensemble du jeu de données dont on dispose. Si l'ensemble de données est volumineux, des problèmes de tailles mémoire se posent, pour stocker l'intégralité de celle-ci et calculer l'erreur faite. Ensuite, également dans le cadre de grandes bases, la convergence peut être lente, l'algorithme de descente n'effectuant qu'une seule mise à jour pour l'ensemble du jeu de données.

Pour palier à ces problèmes, on peut mettre en place des variantes de l'algorithme de descente de gradient en changeant la fréquence des mises à jour des poids, et faire :

- Une mise à jour des poids du réseau **pour chaque exemple**. C'est l'algorithme du gradient stochastique (*Stochastic Gradient Descent*, SGD). La vitesse de convergence est améliorée, malgré une forte variabilité de la fonction d'erreur.

- Une mise à jour **tous les B exemples** (un ensemble de B exemples est alors appelé un *mini-batch*, ou mini-lots) en utilisant un algorithme appelé *mini-batch gradient descent*. Le gradient calculé avec cette version sera la moyenne des gradient d'erreur sur l'ensemble du mini-lot. Il a l'avantage de réduire les fluctuations de la fonction d'erreur, donc la variance des mises à jour des paramètres et enfin la valeur de ceux-ci. Également, la mise en place d'optimisation par calcul matriciel permet une rapidité d'exécution de cet algorithme.

Limitations Cependant, certaines limitations sont à évoquer quant à cet algorithme. En premier lieu, la convergence vers un optimum local "satisfaisant" , i.e. présentant des propriétés de généralisation de l'apprentissage par exemple ou de convergence vers une erreur suffisamment faible, n'est pas assurée. En particulier, si la fonction d'erreur est fortement non-convexe, comme on le rencontre fréquemment.
D'autre part, le choix du taux d'apprentissage (α) est problématique. Doit-il rester inchangé au cours de l'apprentissage ? Comment choisir, à la main, sa valeur ? Ne devrait-on pas considérer différents taux d'apprentissage pour nos différents paramètres ?
Ces différentes questions soulevées sont adressées par de nouveaux algorithmes de descente de gradient, que nous décrivons ci-après.

5.2 Moment

La première variation de l'algorithme de descente de gradient présentée ici est la descente de gradient avec moment (*Momentum gradient descent*), qui permet d'accélérer la convergence de l'algorithme SGD.
En effet, SGD n'est pas très performant pour optimiser un gradient présentant des rigoles, c'est-à-dire où la surface du gradient dans l'espace des paramètres s'incline plus fortement dans une direction que dans les autres. L'algorithme de descente de gradient avec Momentum pallie ces lacunes en ajoutant au terme de mise à jour des paramètres une fraction de celui calculé à l'itération précédente.
Soit ν_t le Momentum à l'itération t, α le *learning rate* comme défini plus haut, θ l'ensemble des paramètres à optimiser et γ le coefficient de Momentum (également appelé momentum par abus de langage), la mise à jour des paramètres s'effectue comme :

$$\begin{aligned}\nu_t &= \gamma\nu_{t-1} + \alpha\nabla\mathcal{L} \\ \theta &= \theta - \nu_t\end{aligned}$$

Avec cet algorithme, on résout le problème de mauvaise convergence dans le cadre de gradient de fonction localement non convexes.
Cependant le problèmes du choix des hyper-paramètres (*learning rate* et momentum) demeure. De plus, on ne dispose toujours que d'un seul taux d'apprentissage pour l'ensemble des paramètres.

5.3 AdaGrad et AdaDelta

Nous présentons dans cette section les algorithmes AdaGrad (pour *adaptive gradient algorithm*, algorithme du gradient adaptatif, [Duchi et al., 2011]) et AdaDelta ([Zeiler, 2012]) une variante d'AdaGrad.

AdaGrad AdaGrad est un algorithme qui vise à résoudre le problème de l'adaptation du taux d'apprentissage aux différents paramètres à optimiser. En fonction de l'optimisation déjà effectuée, et pour deux paramètres différents, il est intéressant d'avoir des *learning rate* différents. Mais comment choisir ces différents *learning rate* ?
Pour ce faire, [Duchi et al., 2011] propose de le pondérer par une mémoire des mises à jour des paramètres passées. Ainsi le *learning rate* est adapté à chacun des paramètres, avec des mises à jour plus importantes pour les paramètres les moins mis à jour, et plus faibles pour les plus fréquemment mis à jour. L'algorithme 1 présente les différentes étapes d'AdaGrad. Cet algorithme est exprimé sous forme vectorisée, permettant une rapidité d'exécution.

Algorithme 1 AdaGrad

Entrées : α *learning rate*, ε constante.
Entrées : θ_0 paramètres initiaux.
Initialisation : $E[\nabla\mathcal{L}^2]_0 = 0$
for t = 1 : T **do**
$E[\nabla\mathcal{L}^2]_t = E[\nabla\mathcal{L}^2]_{t-1} + \nabla\mathcal{L}_t^2$
$\Delta\theta_t = -\frac{\alpha}{\sqrt{E[\nabla\mathcal{L}^2]_t + \varepsilon}}\nabla\mathcal{L}_t$
$\theta_t = \theta_t - 1 + \Delta\theta_t$
end for

Afin de permettre une meilleur compréhension, on détaille ici les différentes étapes en forme non factorisée.
On pose $\theta_{i,t}$ la valeur d'un paramètre à optimiser à l'itération t, $g_{i,t}$ son gradient [1] à l'itération t. Avec l'algorithme de descente de gradient classique, on aurait :

$$\theta_{i,(t+1)} = \theta_{i,t} - \alpha\ g_{i,t}$$

Avec AdaGrad, on modifie cette mise à jour pour que chaque paramètre ait un taux d'apprentissage propre à ces mises à jour passées. On pose $G_{i,t}$ la somme des gradients carrés [2] par rapport à θ_i jusqu'à l'itération t :

$$G_{i,t} = G_{i,t-1} + g_{i,t}^2$$

La mise à jour des paramètres à optimiser devient alors :

$$\theta_{i,(t+1)} = \theta_{i,t} - \frac{\alpha}{\sqrt{G_{t,i} + \varepsilon}}\ g_{i,t}$$

On a donc un *learning rate* adaptatif selon les paramètres, dont la valeur n'est pas à optimiser manuellement. La plupart des utilisations d'AdaGrad fixe ce taux à 0.01 et laissent les mises à jour le modifier au cours de l'apprentissage. On a donc résolu le problème de l'optimisation du *learning rate*.

Cependant, la faiblesse principale d'AdaGrad est l'accumulation des mises à jour passées au dénominateur du taux d'apprentissage ($\sqrt{E[\nabla\mathcal{L}^2]_t}$). Ces quantités sont toujours positives, ce faisant le taux d'apprentissage ne peut que diminuer, rendant l'apprentissage de plus en plus lent, jusqu'au point où le taux d'apprentissage devient infinitésimal et l'apprentissage s'arrête.

1. de la fonction d'erreur
2. Écrite sous forme matricielle comme $E[\nabla\mathcal{L}^2]$ dans l'algorithme 1

AdaDelta AdaDelta vise à répondre à ce problème. Au lieu de garder en mémoire l'ensemble des mises à jour passées, on ne les conserve que sur une certaine fenêtre de taille ω. La somme des gradients est définie récursivement comme une moyenne des gradients au carré passés. Soit t l'itération considérée, $E[\nabla\mathcal{L}^2]_{t-1}$ la moyenne des gradients au carré passée, on calcule alors :

$$E[\nabla\mathcal{L}^2]_t = \rho E[\nabla\mathcal{L}^2]_{t-1} + (1-\rho)\nabla\mathcal{L}_t^2$$

Ce calcul récursif permet de ne garder en mémoire les mises à jour passée que sur une certaine fenêtre d'itération et non plus sur l'apprentissage passé en totalité. L'inverse de cette moyenne des gradient passés sert de pondération du taux d'apprentissage, comme avec AdaGrad. Comme celui-ci n'est gardé en mémoire que sur une certaine fenêtre (paramétrée par ρ), le problème de dilution du taux d'apprentissage est résolu.

Afin de ne pas avoir à fixer à la main une valeur du taux d'apprentissage (paramètre α dans AdaGrad), [Zeiler, 2012] propose de garder également en mémoire la quantité servant à la mise à jour des poids $\Delta\theta$. On pose alors :

$$E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1-\rho)\Delta\theta_t^2,$$

la moyenne des mise à jour au carré du poids θ. On a alors la mise à jour des poids comme :

$$\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_{t-1}+\varepsilon}}{\sqrt{E[\nabla\mathcal{L}^2]_t+\varepsilon}}\nabla\mathcal{L}_t$$

Non content d'avoir supprimé un hyper-paramètre à optimiser, on obtient également une mise à jour proportionnelle à l'inverse de la matrice hessienne (la démonstration est présentée dans [Zeiler, 2012]). Cette quantité sert dans la méthode d'optimisation de Newton, et permet de meilleur résultats de convergence. Néanmoins, cette matrice est très coûteuse à calculer et cette méthode rarement mise en œuvre. Avec AdaDelta, on a donc une approximation de cette méthode.
L'algorithme 2 détaille la procédure à effectuer.

Algorithme 2 AdaDelta

Entrées : ρ : *decay rate*, ε constante.
Entrées : θ_0, paramètres initiaux.
Initialisation : $E[\nabla\mathcal{L}^2]_0 = 0$, $E[\Delta\theta^2]_0 = 0$
for t = 1 : T **do**
  $E[\nabla\mathcal{L}^2]_t = \rho E[\nabla\mathcal{L}^2]_{t-1} + (1-\rho)\nabla\mathcal{L}_t^2$: calcul de la moyenne des gradient passé
  $\Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_{t-1}+\varepsilon}}{\sqrt{E[\nabla\mathcal{L}^2]_t+\varepsilon}}\nabla\mathcal{L}_t$: calcul de la mise à jour à l'itération t
  $E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1-\rho)\Delta\theta_t^2$: calcul de la moyenne des mises à jour passées
  $\theta_t = \theta_{t-1} + \Delta\theta_t$: calcul des nouveaux paramètres
end for

Il est à noter que la sensibilité d'AdaDelta à ces deux paramètres (ρ et ε) est relativement faible en regard de celle des autres méthodes. On peut donc fixer ρ à 0.99 et ε à 10^{-6} comme conseillé dans [Zeiler, 2012]. Ainsi, grâce à cet algorithme, aucun hyper-paramètres n'est à optimiser.

5.4 Utiliser la descente de gradient pour l'apprentissage d'un réseau

5.4.1 Rétro-propagation et dérivation des fonctions composées

Rétro-propagation Utilisée en combinaison d'un algorithme de descente de gradient, la rétro-propagation de l'erreur à travers un réseau de neurones est utilisée pour l'apprentissage des poids de celui-ci. Cette méthode s'effectue en deux phase :

1. Commencer par une passe de propagation (*forward pass*). On calcule les activations de neurones de chacune des couches jusqu'à la fin du réseau. On calcule alors l'erreur du réseau sur cette prédiction.

2. Propager l'erreur dans le sens contraire de la première passe (*backward pass)*, et effectuant les mises à jour des poids selon la descente de gradient.

La figure 11 montre un exemple de rétro-propagation.

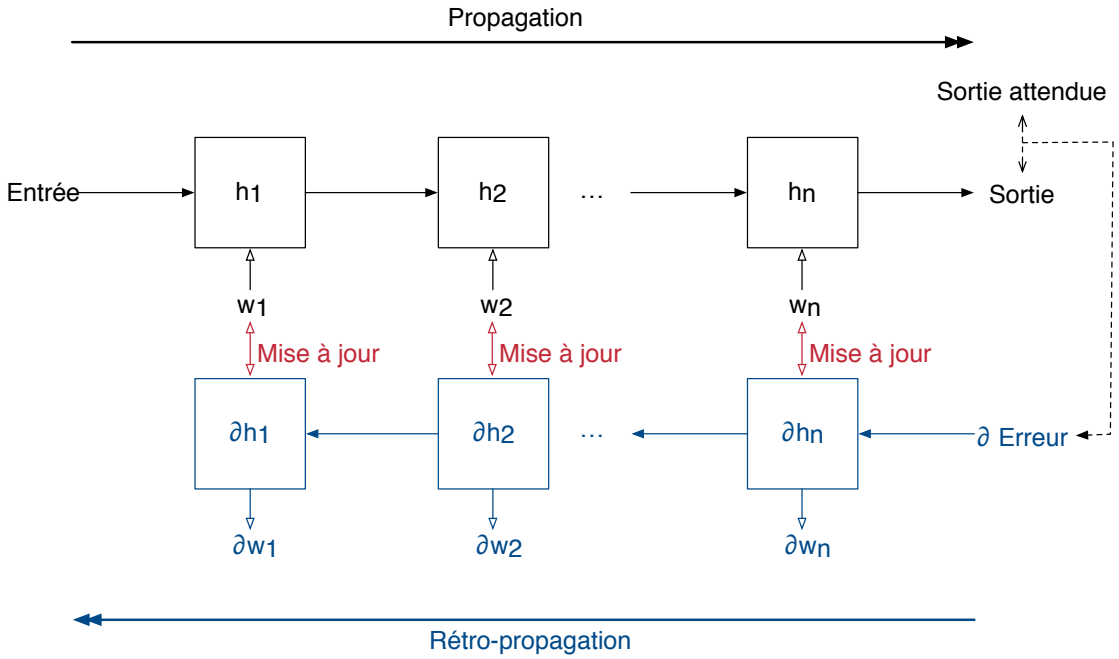

FIGURE 11 – Exemple de rétro-propagation

Afin d'améliorer l'efficacité du calcul des différentes dérivées des poids, on utilise les propriétés de dérivations des fonctions composées (ou *chain rule*).

Chain rule On cherche à calculer, à partir de l'erreur $\mathcal{L}$ faite par le réseau, la dérivée de celle-ci par rapport à un poids de notre réseau w. Grâce aux propriétés de dérivation des fonctions composées, on peut exprimer les dérivées des poids d'une couche en fonction de la couche supérieure lors de la rétro-propagation.

5.4.2 Sur-apprentissage

Le sur-apprentissage est un phénomène qui apparait lorsque les performances d'un réseau sont bonnes sur des données d'entrainement, mais leur généralisation à des données test est mauvaise. On dit alors que le réseau sur-apprend sur les données d'entrainement. Pour endiguer ce phénomène, plusieurs méthodes sont possibles. Il convient avant tout de séparer les données disponibles en plusieurs ensembles.

Train, validation et test set Sur l'ensemble de données disponibles, on construit trois sous-ensemble distincts :

1. L'ensemble **d'entrainement** (*train set*) sert à la phase d'apprentissage uniquement. C'est sur les erreurs faites sur ces données d'entrainement que l'on mettra à jour les poids à apprendre.

2. L'ensemble **de validation** (*validation set*) sert également lors de la phase d'apprentissage. Néanmoins, on n'apprendra jamais sur ces donnes, i.e. on ne modifiera jamais les poids sur les erreurs faites sur le jeu de données de validation. Cet ensemble sert uniquement à surveiller les performances de généralisation de notre réseau. Si la différence d'erreur moyenne faite sur l'ensemble d'entrainement et sur l'ensemble de validation reste faible, alors le sur-apprentissage est faible. A l'inverse, si l'erreur faite sur l'ensemble d'entrainement baisse au fur et à mesure de l'apprentissage mais que l'erreur faite sur l'ensemble de validation reste importante, alors notre réseau sur-apprend.

3. L'ensemble **de test** (*test set*) sert à la phase de test de notre réseau. Ce sont des données sur lesquelles notre réseau n'a pas appris, et on peut donc vérifier les propriétés de bonnes généralisation de l'apprentissage du réseau.

Cette séparation en sous ensemble permet de surveiller les performances de notre réseau et d'observer le sur-apprentissage. Face à celui-ci différentes méthodes existent.

Endiguer le sur-apprentissage Deux méthodes principales existent :

- L'ajout d'une **pénalité** à la fonction de cout. Une régularisation ajoutée à la fonction d'erreur, comme par exemple l'ajout de la norme L1 ou L2 des paramètres à estimer, permet d'endiguer le sur-apprentissage. En effet, on cherche à minimiser la fonction d'erreur. En ajoutant la norme à cette fonction, on minimise donc également la norme considérée. On s'assure ainsi d'avoir uniquement des poids faibles (les poids de valeur importante sont généralement représentatifs du sur-apprentissage).

- Le **dropout** qui permet de rajouter un "bruit" aux données, et rend donc le sur-apprentissage difficile (il est en effet compliqué de surapprendre sur des données que l'on ne possède pas entièrement). Pour cela, l'idée est d'annuler une partie des données d'entrée d'une couche selon un certain taux. À chaque passe, on tire un certain nombre de composantes des données d'entrée et elles ne seront pas prises en compte.

5.4.3 Mesurer l'efficacité de l'apprentissage

Après apprentissage de notre réseau, il faut désormais évaluer ces performances. Une des métriques le plus souvent utilisées est la F-mesure.

Précision et Rappel Commençons par définir la précision et le rappel dont est issue la F-mesure. Ces deux mesures sont issues de combinaisons du nombre de vrai et faux positifs et négatifs :

	Annotation	Prédiction
Vrai positif	Vrai	Vrai
Faux positif	Faux	Vrai
Vrai négatif	Faux	Faux
Faux négatif	Vrai	Faux

TABLE 1 – Vrai et faux positifs et négatifs

La **précision** est alors le nombre de vrai positifs sur le nombre d'exemples prédits comme vrais par le modèle. Comme son nom l'indique, elle permet de mesurer la précision du modèle, soit la confiance que l'on peut attribuer au modèle. Plus la précision est forte, plus une prédiction 'vrai' du modèle peut être considérée comme exacte.

Le **rappel** est le nombre de vrai positifs sur le nombre d'exemples effectivement positifs dans l'ensemble de données (voir figure 12 pour un exemple). Le rappel permet de mettre en évidence l'exhaustivité du modèle.

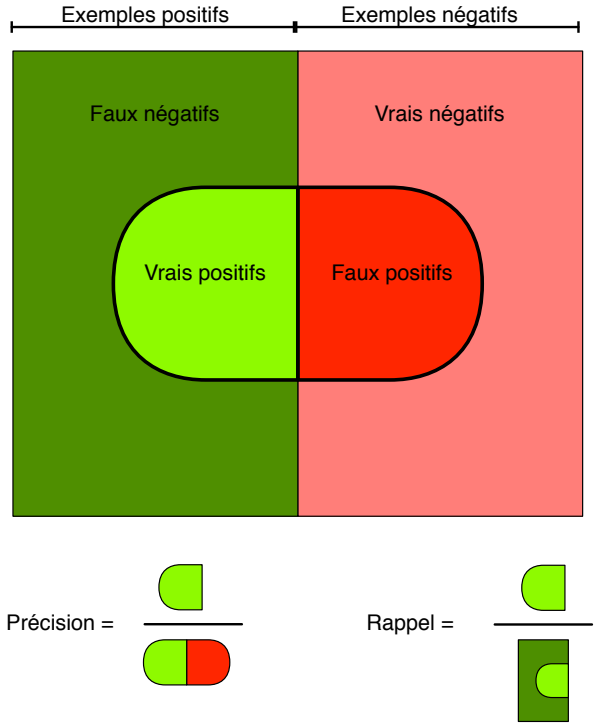

FIGURE 12 – Précision et rappel

F-mesure La F-mesure est une combinaison de la précision et du rappel :

$$\text{F-mesure} = 2 \times \frac{\text{précision} \times \text{rappel}}{\text{précision} + \text{rappel}}$$

Elle mesure l'exactitude du modèle.

Nous avons donc défini le modèle utilisé et les moyens d'apprendre ces modèles. Dans la partie suivante nous présentons comment utiliser ces méthodes pour l'estimation de structures musicales.

6 Utiliser les ConvNets pour détecter des structures musicales

La première partie de ce stage porte sur la ré-implémentation et l'étude de les articles de [Ullrich et al., 2014] et de [Grill and Schlüter, 2015].

6.1 Avec des spectrogrammes

Ensemble de données et objectifs On utilise les fichiers de l'ensemble de données SALAMI à notre disposition(pour *Structural Analysis of Large Amounts of Musical Information*, voir [Smith et al., 2011]), contenant plus de 700 titres, et 1400 annotations, et on sépare ces données en jeu d'entrainement, de test et de validation. Dans [Ullrich et al., 2014], le détails des titres utilisés pour l'apprentissage n'est pas donné. On n'utilise donc pas la même base d'entrainement dans le cadre de ce travail, la reproduction des résultats de l'article n'est donc pas assurée. Cependant, l'idée est ici de comprendre comment fonctionnent les méthodes proposées et de ré-implémenter ces méthodes pour avoir un point de comparaison avec de nouvelles propositions.

Pré-traitement Les entrées de ce réseau sont des extraits de **mélogramme** (spectrogramme filtrées par un banc de filtres de Mel, pour coller à la perception humaine) auxquels on applique un *max-pooling* temporel de 3, 6 ou 12 trames. Si l'on désigne par x_i une colonne du mélogramme avant *max-pooling*, $p = \{3, 6, 12\}$, celui-ci s'exprime comme

$$x'_i = max_{j=1..p} \left(x_{(i-1)p+j} \right)$$

Les taux $p = \{3, 6, 12\}$ sont désignés respectivement par `high`, `std` et `low`. De plus, pour pouvoir détecter les frontières en début et en fin de piste, on concatène au mélogramme des trames de bruit rose à -70 dB, en fonction de la taille de l'extrait donné en entrée au ConvNet (la figure 13 présente un exemple de spectrogramme ainsi traité).

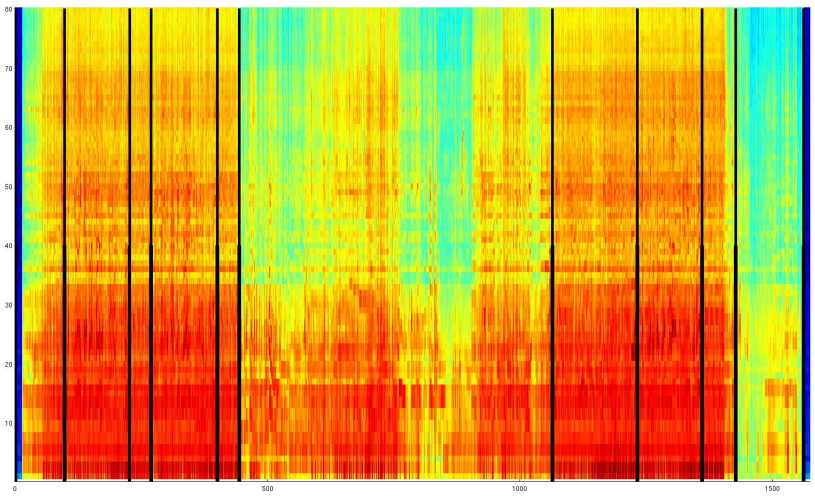

FIGURE 13 – Exemple de mélogramme avec annotations (abscisse : temps en secondes, ordonnée : numéro de bande de Mel.

Les annotations de l'ensemble de données peuvent être assez variables d'un annotateur à l'autre, on utilise donc la technique du *target smearing*, qui consiste à, pour un exemple donné, étiqueter les E exemples précédents et suivants comme étant positifs. Ceux-ci auront néanmoins un poids réduit lors de l'apprentissage. Autour de l'exemple réellement positifs, on assigne une valeur aux exemples notés comme positifs selon une fonction gaussienne, comme illustré en figure 14.

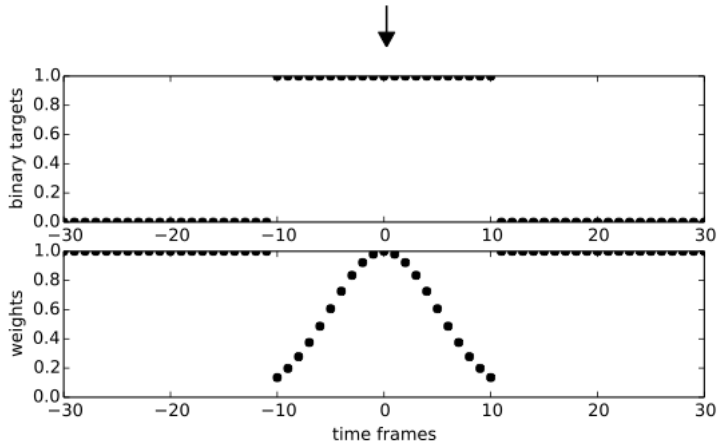

FIGURE 14 – Valuation des exemples pseudo-positifs. Tiré de [Ullrich et al., 2014]

D'autre part, il y a relativement peu d'exemples marqués positifs pour chaque piste. Pour palier à ce problème, les exemples positifs ont une probabilité trois fois plus élevée d'être choisi dans la phase d'apprentissage.

Architecture On utilise l'architecture présentée en figure 15. Les couches convolutionnelles ont des activation en tangente hyperbolique. Le *stride* est unitaire (pas d'avancement de la convolution de 1) et l'on n'effectue pas de *zero-padding*. On a donc 1.200.512 poids et 177 biais à apprendre pour ce réseau.
La sortie du réseau est celle d'un neurone avec une activation sigmoïde, donnant donc une sortie entre 0 et 1. Celle-ci représente la probabilité pour la trame centrale de l'extrait donné en entrée d'être un instant de césure dans le signal.

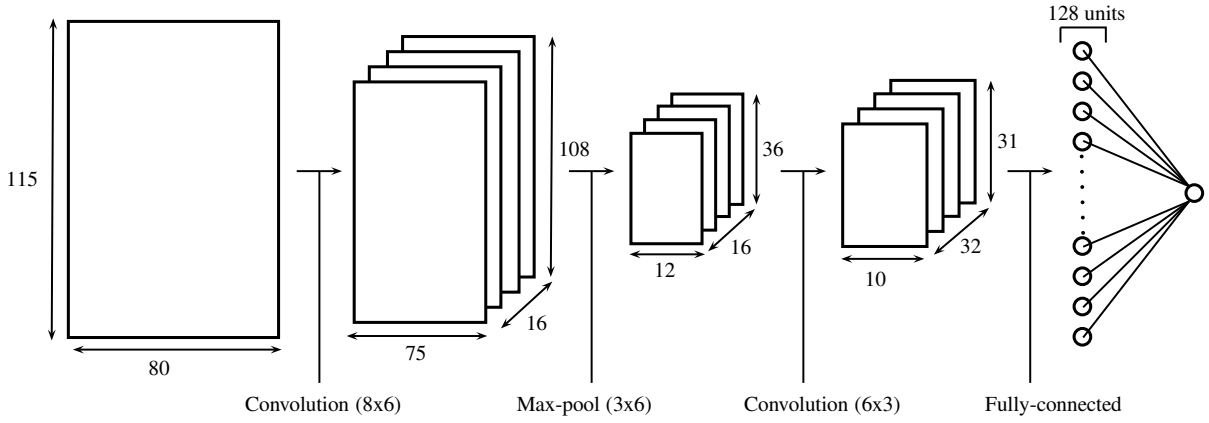

FIGURE 15 – Architecture utilisée

Entrainement du réseau On utilise la descente de gradient avec AdaDelta, en posant $\varepsilon = 10^{-6}$ et $\rho = 0.99$ qui sont les valeurs préconisées dans [Zeiler, 2012]. Nous avons également implémenté les algorithmes Momentum et AdaGrad pour l'apprentissage de nos réseaux (contrairement à [Ullrich et al., 2014] qui n'utilise que le Momentum). Il apparait que pour un nombre d'époque [1] équivalent, l'apprentissage avec AdaDelta est plus performant (généralisation des performances du réseau améliorée). La taille des mini-batches est fixée à 64 dans [Ullrich et al., 2014]. L'erreur sur l'ensemble de validation est contrôlée de façon à éviter le sur-apprentissage. Une régularisation avec le *dropout* à 50% des entrées des deux dernières couches fortement connectées est utilisée.

Post-traitement et test Après apprentissage du réseau, on procède à une détection de pics (*peak-picking*) de la sortie du neurone de la dernière couche, en considérant un instant comme césure si celui-ci est le maximum d'activation sur une durée de 6 secondes autour de lui. A chaque césure est associé un poids (*strength*) correspondant à son activation moyennée sur 18 secondes.
Pour améliorer les résultats, nous utilisons également une méthode appelée *bagging*. Celle-ci consiste à entraîner le même réseau avec plusieurs initialisations différentes. On utilise ensuite la moyenne des sorties de chaque réseau.

Évaluation Pour évaluer notre réseau, on utilise la F-mesure des résultats (voir partie 5). Comme une frontière peut être considérée comme variable (d'un annotateur à l'autre par exemple), on considère deux métriques dérivées de la F-mesure et spécifique à l'estimation de structure : la $F_{0.5}$ et la F_3. Ces deux mesures induisent une tolérance dans le résultat trouvé. Pour la $F_{0.5}$, on considère une frontière estimée cPour F_3, la tolérance est fixée à 3 secondes.

Résultats MP2(2013) est la méthode de la campagne MIREX 2012 ayant les meilleurs résultats de segmentation (voir [McFee and Ellis, 2013]), et sert de comparatif dans l'article [Ullrich et al., 2014].

1. Une époque est une passe d'apprentissage de l'ensemble des mini-lots (ou *mini-batches*).

Algorithme (taille des extraits / taux de max pooling / taille de target smearing)	À ±0.5 s.			À ±3 s.		
	F-mesure	Précision	Rappel	F-mesure	Précision	Rappel
[Ullrich et al., 2014] (16s / std / 1.5s)	0.4646	0.5553	0.4583	0.5724	0.5648	0.6675
Cohen-Hadria : 16s / std / 1.5s avec bagging	**0.4136**	**0.7169**	**0.3112**	**0.5636**	**0.5392**	**0.6572**
Cohen-Hadria : 16s / std / 1.5s sans bagging	0.3743	0.4775	0.3498	0.5389	0.6	0.5319
[McFee and Ellis, 2013]	0.3280	0.3001	0.4108	0.5213	0.4793	0.6443

TABLE 2 – Résultats de l'implémentation avec spectrogramme

Dans la table, nous voyons que nos résultats sont meilleurs avec bagging (F-mesure de 0.41 et 0.56) que sans (0.37 et 0.53). Ils sont bien que meilleurs que les résultats de l'état de l'art hors ConvNet (0.32 et 0.52), mais reste inférieurs à ceux de [Ullrich et al., 2014] (0.46 et 0.57). Deux points peuvent expliquer cela. Premièrement, la différence d'ensemble de données est un obstacle important à la reproductibilité d'apprentissage de réseaux de neurones. D'autre part, il est à noter que l'apprentissage de réseaux convolutifs (appris sur plusieurs centaines de pistes, donc plusieurs millions d'extraits de spectrogrammes) est couteux en temps et en puissance de calcul. N'ayant pas à disposition de module de calcul sur GPU, nous n'avons pu entrainer un nombre important de modèle, ce qui pénalise également nos résultats [1].

Analyses des résultats Il est cependant possible d'analyser les résultats obtenus. Premièrement, on s'aperçoit que la précision de nos modèles est importante, mais que le rappel est faible. Les ConvNets sont des modèles adaptés aux images. En les appliquant sur des spectrogrammes (représentation temps-fréquence), on induit une composante temporelle dans l'analyse de nos images, conditionnée par la taille de nos filtres. La structure musicale est basée à la fois sur des césures temporelles à horizon court (un court silence peut séparer des parties, le contenu spectral peut fortement varier entre deux sections) et sur des répétitions à échéances longues (répétition du refrain par exemple à plusieurs minutes d'intervalle). Le ConvNet semble ne repérer que les césures à horizon temporel court (ce qui est cohérent avec la taille des filtres). Ceci explique les résultats quand aux observations concernant la précision et le rappel.

En observant les filtres appris par nos réseaux, on retrouve également l'idée que les frontières détectées dans le spectrogramme sont des frontières de variation fortes du contenu spectral.

1. Les réseaux de neurones sont des modèles très sensibles à l'initialisation de leurs paramètres. Entrainer de nombreux modèles est donc essentiel afin d'obtenir les meilleurs résultats possibles.

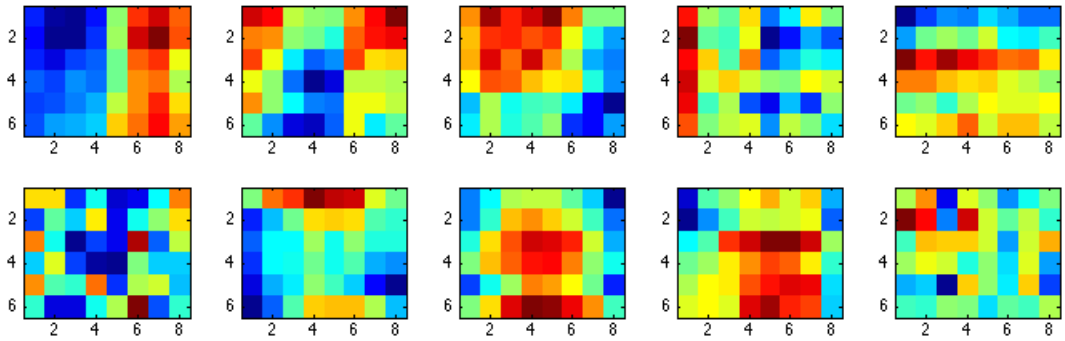

FIGURE 16 – Exemple de filtres appris (première couche du réseau). Abscisses : temps, Ordonnées : Mel.

Malgré des résultats encourageant, il apparaît qu'il serait intéressant de choisir une autre représentation des pistes musicales, faisant par exemple apparaître des liens de répétitions entre extraits de la piste. On choisit alors de suivre [Grill and Schlüter, 2015] et d'utiliser des matrice d'auto-similarité (SSM).

6.2 Avec des matrices de lag

Génération des matrices de lag On part d'un mélogramme $X = x_{i=1...n}$ de N trames, identique à celui de [Ullrich et al., 2014] sur lequel on applique un *"max-pooling"* d'un facteur p. Afin d'extraire des MFCC (voir partie 2), on applique la DCTII sur chaque trame de ce mélogramme, soit $\widetilde{x_i} = \text{DCTII}(x'_i)$, que l'on empile par m trames :

$$\widehat{x_i} = \left[\widetilde{x_i}^T, ..., \widetilde{x}_{i+m}^T\right]^T$$

Une fois ces vecteurs calculés, il reste alors à construire une matrice de lag sur ces MFCC (voir partie 2). Pour rappel dans une matrice de lag, les similarités sont exprimées non pas en relation temps-temps, mais en relation temps-décalage.

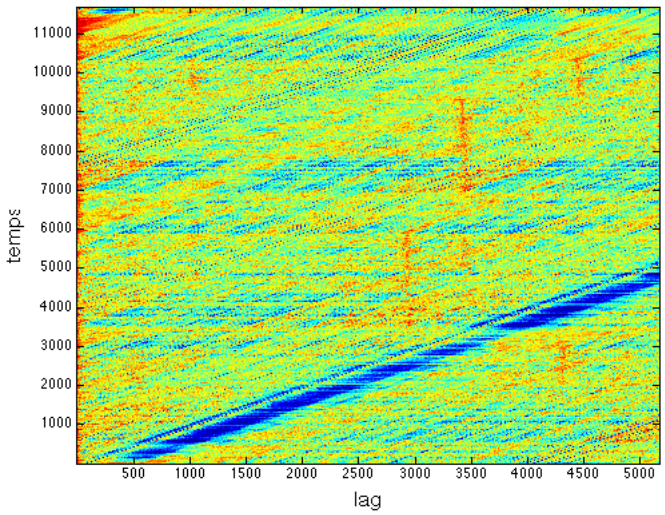

FIGURE 17 – Exemple de matrice de lag

La figure 17 présente un exemple de matrice de lag. Les lignes verticales sont les séquences répétées au cours du signal. Ce sont en fait des lignes parallèles à la diagonales principales d'une matrice d'auto-similarité qui, avec le changement d'axe pour faire apparaitre le décalage, deviennent parallèle à l'un des axes de la matrice. Ce type de représentation met en lumière les répétitions de séquences musicales au sein d'une pièce, il est donc intéressant de les utiliser dans le cadre de l'apprentissage de structure par CNN.
Pour calculer la matrice de lag D, on utilise la distance cosinusoïdale $\delta_{cos}(x, y) = 1 - \langle \frac{x}{||x||}, \frac{y}{||y||} \rangle$:

$$D_{i,l} = \delta_{cos}(\widehat{x}_i, \widehat{x}_{i-l}), \text{ pour } l = 1... \left\lfloor \frac{L}{p} \right\rfloor$$

où L est la valeur maximale du lag (exprimé en nombre de trames de décalage). Une permutation circulaire sir l'indice i est ensuite appliquée $i' = i + \left\lfloor \frac{N}{p} \right\rfloor$. Ne reste qu'à seuiller cette matrice de lag, à l'aide d'une fonction sigmoïde $\sigma(x) = 1/(1 + e^{-x})$ et d'un seuil ε_i :

$$R_{i,l} = \sigma(1 - \frac{D_{i,l}}{\varepsilon_i})$$
$$\varepsilon_i = Q_\kappa(D_{i,1}, ..., D_{i,\lfloor \frac{L}{p} \rfloor})$$

avec Q_κ le quantile κ, fixé ici à 0,1.

Dans [Grill and Schlüter, 2015], L est fixé à 5168, soit 240 secondes. Ceci résulte en une matrice de lag de 5168 colonnes. Cette taille est très importante pour constituer l'entrée d'un ConvNet (occupation mémoire très importante due à la convolution). De ce fait, [Grill and Schlüter, 2015] tronque la matrice aux 300 premières colonnes. La conséquence est que seules les répétitions inférieures à 14 secondes environ seront représentées. Dans [Grill and Schlüter, 2015], cette matrice est encore réduite par *max-pooling* d'un facteur 3 pour arriver à 100 colonnes. Afin de conserver le même taux de FPS (*frame per seconds*) que le spectrogramme du premier ConvNet, on applique un *max-pooling* également sur les temps d'un facteur 3.

Architecture utilisée Ainsi, on a désormais des matrices de lags de dimension acceptable pour l'entrée d'un ConvNet. Celles-ci servent donc à entraîner un nouveau réseau dont l'architecture est semblable à celle présentée à la figure 15 (le nombre de filtres des couches convolutionnelles est multiplié par deux uniquement, on a donc 3.339.904 poids à apprendre et 225 biais). Les fonctions d'activations sont toujours des tangetnes hyperboliques sauf pour les couches fortement connectées qui possèdent des activations sigmoïdes.

Entrainement et post-traitement Au vu des bonnes performances d'Adelta sur les mélogrammes, on choisit de n'utiliser que cet algorithme de descente de gradient. La taille des mini-batches est toujours fixée à 64 exemples. Le *dropout* est gardé sur les couches fortement connectées comme précédemment. On garde à l'identique les post-traitement présentés pour la méthodes avec mélogramme.

Résultats Nous présentons ici les résultats obtenus.

Méthodes	À ±0.5 s
	F-mesure
[Ullrich et al., 2014] (mélogramme)	0.4646
[Grill and Schlüter, 2015] (matrice de lag)	0.43
Cohen-Hadria (matrice de lag)	**0.412**
MP2(2013)	0.3280

TABLE 3 – Résultats de l'implémentation avec matrice de lag

On remarque que l'utilisation des matrices de lag n'améliore pas les performances de détection. Dans le cas de [Grill and Schlüter, 2015], les performances chutent même entre les deux représentations.
Comme mentionné ci-dessus, ceci peut s'expliquer par la prise en compte d'un nombre réduit de décalage (14s).

Analyse de filtres Néanmoins, lors de l'analyse des filtres appris (voir figure 18), on remarque que les motifs capturés par le ConvNet entrainé sur des matrices de lag sont bien différents de ceux appris sur des mélogrammes. Les motifs diagonaux présents sur ces filtres s'expliquent par les frontières des blocs homogènes (devenant des parallélogramme dans une représentation de type temps-lag). On aurait souhaiter pouvoir représenter par ces filtres les lignes verticales (voir Figure 17) caractéristique des répétitions. Les informations mises en valeur sont donc de nature différentes, et certainement utiles pour détecter d'autres types de frontières.

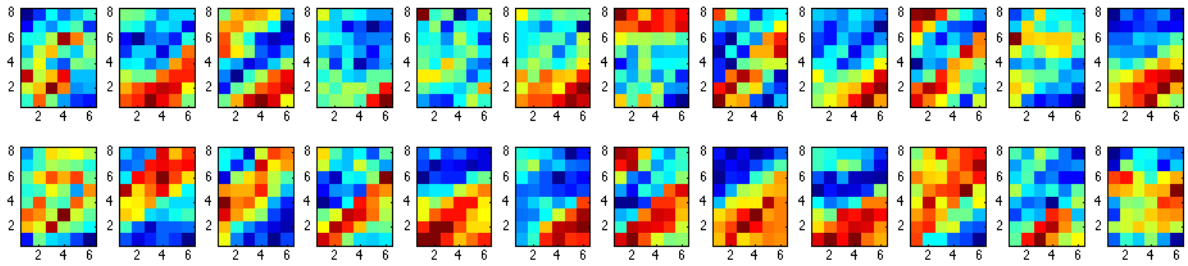

FIGURE 18 – Filtres de la première couche appris sur des lag-matrices. Abscisses : temps, Ordonnées : lag.

L'idée suivante est d'utiliser les informations apprises par les deux représentations, melogramme et matrice d'auto-similarité, et ainsi combiner les différentes informations présentes dans les deux représentations.

6.3 Combiner les deux représentations

Plusieurs choix s'offrent à nous pour combiner les *features* apprises par ConvNet avec spectrogrammes et matrices d'auto-similarité (voir figure 19).

Moyenne de deux réseaux distincts (MoyFusion) L'idée la plus simple est d'entraîner deux réseaux distincts avec en entrée nos deux représentations. Une fois l'apprentissage terminé, il ne reste qu'à faire la moyenne des sorties de ces deux réseaux.

Cependant, cette méthode ne permet pas réellement de combiner les informations contenues dans les deux représentations.

Combinaison durant l'apprentissage par *Fully Connected* (Fusion FC) Une première possibilité pour combiner les deux représentations est d'entrainer deux sous réseaux composés de deux couches convolutionnelles chacun et de les réunir dans une couche fortement connectée (*fully-connected*, FC). Les nouvelles représentations apprises par les différentes couches des sous réseaux sont ainsi fusionnées en une seule qui sert d'entrée à la couche fortement connectée.

Combinaison durant l'apprentissage par couche convolutionnelle (FusionConv) Comme un extrait de spectrogramme et de matrice d'auto-similarité couvrent le même contexte temporel et ont les mêmes dimensions, on peut également en place une fusion des *features maps* du ConvNet en une couche convolutionnelle.

Ainsi, avec ce deux méthodes de fusion, les représentations apprises par le réseaux sont véritablement combinées, dès la phase d'apprentissage du réseau, et non plus à la phase de test.

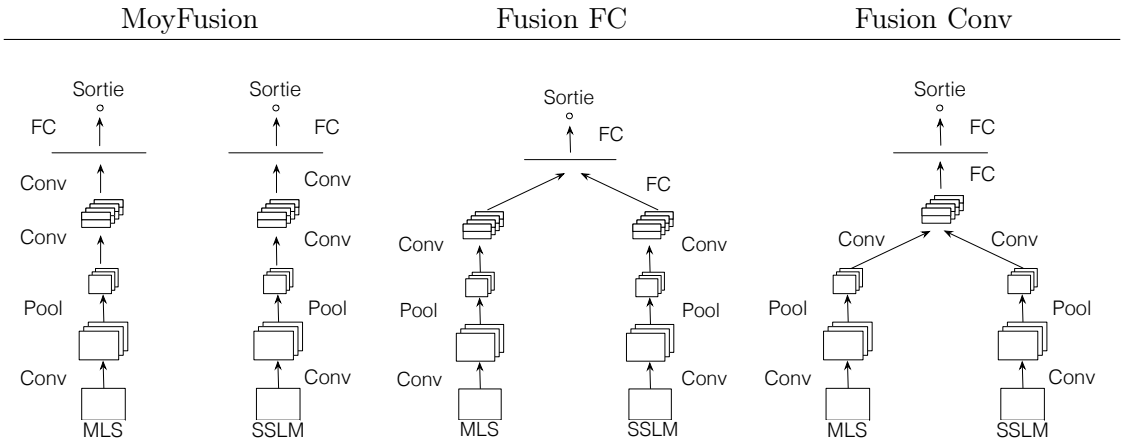

FIGURE 19 – Différentes architectures pour combiner les deux représentations

Entrainement et post-traitement L'entrainement de ces réseaux nécessite de mettre en œuvre une implémentation différente afin de réaliser la fusion de deux représentations. La rétro-propagation dans ce cadre s'effectue comme dans un réseau avec une seule "branche", en respectant l'ordre inverse de la propagation. On utilise toujours l'algorithme ADADLETA pour la descente de gradient. La taille des mini batches est fixée à 128 exemples. Le *dropout* à 50% des entrées des couches fortement connectées est utilisé. On garde les post-traitement appliqués pour les mélogrammes.

Résultats La fusion de deux représentations améliore effectivement les résultats. Le rappel est grandement augmenté, ce qui confirme que les mélogrammes et les matrices de lag aident à capturer différents types de frontières.

Méthodes	À ±0.5 s.		
	F-mesure	Précision	Rappel
[Grill and Schlüter, 2015]	0.523	0.646	0.484
Fusion Conv	**0.4526**	0.6221	0.3655
Fusion FC	**0.4503**	0.6620	0.3731
MoyFusion	**0.4429**	0.5612	0.3567

TABLE 4 – Résultats de l'implémentation avec fusion des représentations.

Nous avons donc étudié en détail les propositions existantes pour l'estimation de structures musicales avec des réseaux de neurones convolutionnels. Avec l'étude des résultats et des filtres, nous pouvons désormais comprendre en quoi les ConvNets aident à la détection de frontières. En combinant différentes représentations à différents niveaux, nous avons vu que les résultats étaient grandement améliorés.
Nous décidons donc de mettre en oeuvre une méthode pour combiner d'autres représentations. Nous présentons cette méthode dans la partie suivante.

7 Utilisation des matrices d'auto-similarité

Matrice d'auto-similarité On décide alors d'utiliser deux matrices d'auto-similarité (*self-similarity matrix*, SSM) sur deux descripteurs différents, les MFCCs et les chromas (voir partie 2.1).
Ces deux descripteurs véhiculent en effet deux types d'informations différentes. Les MFCCs donnent une information sur le timbre d'une piste musicale. Les chromas, quand à eux, donnent des informations sur le contenu harmonique du signal. Ces deux informations couplées semblent décrire précisément différentes propriétés de notre signal.

Combiner ces représentations Pour combiner ces représentations, on décide de les concaténer pour former un tenseur de profondeur 2. Les ConvNets sont des modèles qui prennent en entrée des images. Si il s'agit d'une image en couleur, alors celle-ci est représentée par un tenseur de profondeur 3 (une matrice pour chaque couleur R, G et B). En concaténant ces deux matrices d'auto-similarité, on obtient deux "couleurs" différentes, représentant notre signal.
Pour rendre la dimension des matrices acceptable pour l'entrée d'un réseau de neurones, on donne les sous-matrice formées le long de la diagonale principale (voir figure 20). On effectue un *max-pooling* également les SSMs pour correspondre au même taux de trames par seconde que précédemment.

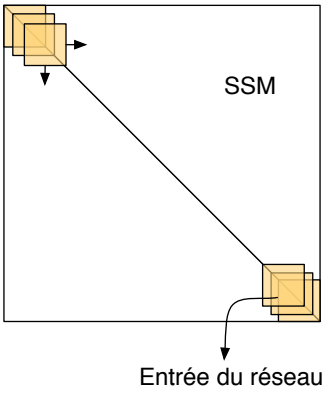

FIGURE 20 – Entrée de notre modèle

Architecture utilisée L'architecture utilisée est semblable à celle utilisée précédemment. Les ajustements nécessaires sont réalisés pour prendre en entrée des SSMs de 115×115. On garde des activations tangente hyperbolique (sauf pour le dernier neurone qui garde une activation sigmoïde figurant une probabilité).

Apprentissage AdaDelta est utilisé. La taille des mini-lots est de 128. La régularisation avec *dropout* à 50% des entrées des couches fortement connectées est conservée. Comme précédemment, l'erreur sur l'ensemble de validation est contrôlée de façon à éviter le sur-apprentissage.

Résultats Nous présentons ici les résultats de nos modèles. La concaténation de deux "couleurs" de matrice d'auto-similarité (FusionSSM) améliore grandement les résultats obtenus comparativement à l'utilisation des matrices SSM individuelles (SSM Chroma et SSM MFCC). On passe d'une F-mesure de 0.29 à 0.39.

Méthodes	À ±0.5 s.		
	F-mesure	Précision	Rappel
FusionSSM avec bagging	0.3982	0.4036	0.4051
SSM Chroma avec bagging	0.2950	0.2567	0.3403
SSM MFCC avec bagging	0.2989	0.3465	0.3025

TABLE 5 – Résultats de l'implémentation avec SSM

On remarque également un lissage des différences entre les taux de rappel et de précision. L'utilisation de deux descriptions semble aider à détecter différents types de frontières dans le signal audio. Cette observation est également corroborée par les filtres appris (voir figure 21).

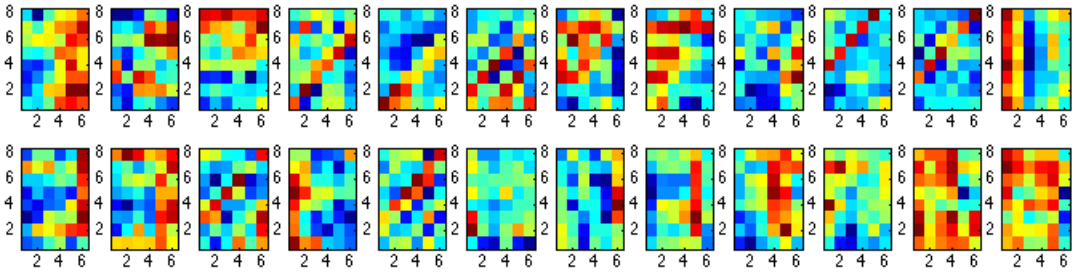

FIGURE 21 – Filtres appris par le ConvNet (première couche), Abscisse : temps, Ordonnées : temps

On constate que différents motifs apparaissent. Des motifs diagonaux semble capturer des répétition le long de la diagonale principale. Des motifs de césures semblables à ceux observés avec les mélogrammes sont également présents.

Dans cette partie, nous avons donc présenté comment l'utilisation des matrices d'auto-similarité peut être intéressante pour l'apprentissage d'un réseau de neurones convolutionnel. Pour poursuivre ces travaux, il pourrait être avantageux de mettre en place une fusion de plusieurs matrices d'auto-similarités avec plus de "couleurs".

8 Conclusion

Ce rapport présente notre travail réalisé dans le cadre d'un stage de Master 2 ATIAM.

Celui-ci porte sur l'utilisation des réseaux de neurones convolutionnels et les problématiques liées à leur apprentissage. Nous avons donc étudié les différents algorithmes d'optimisation de ces réseaux (différentes implémentations de descente de gradient : Momentum, AdaGrad et AdaDelta).

Nous avons ensuite appliqué ces modèles au problème de l'estimation de structure musicale.

Nous avons montré que ces modèles, développés à la base pour l'analyse d'image, sont efficaces pour mettre en évidence les propriétés temporelles nécessaires à la détection de frontière de structure temporelle. Les représentations d'entrées de ces réseaux, bien que sous forme d'images, possèdent en effet un axe temporel permettant l'analyse (sous forme de motifs dans l'image) d'une certaine temporalité du signal.

Lors d'une ré-implémentation de l'état de l'art du domaine ([Ullrich et al., 2014]), nous avons montré le bénéfice d'une optimisation adaptative (AdaDelta). Nous avons également étudié différents types de représentation en entrée du réseau : spectre mélogramme et matrice de lag ainsi que leur utilisation conjointe par *Late-Fusion.*

Nous avons également proposé une nouvelle représentation d'entrée : la matrice d'auto-similarité. Pour celle-ci, nous avons montré la pertinence d'utiliser la profondeur de la couche d'entrée des ConvNets. Celle-ci attribuée aux trois couleurs RGB en reconnaissance d'image est ici assignée à différentes matrices d'auto-similarité correspondants à différentes observations (MFCC et Chroma). Bien que les résultats obtenus restent inférieures à ceux obtenus par ([Ullrich et al., 2014]) (notons cependant que nous n'avons pas accès à l'entièreté de la base d'entrainement de [Ullrich et al., 2014]), nous pensons que cette manière de combiner différents points-de-vue sur le contenu audio est plus prometteuse que les méthodes de *Late-Fusion* utilisée jusqu'à présent (voir partie 6.3). De plus, les motifs des filtres appris selon cette méthode permettent une interprétation sémantique correspondant de manière plus étroite à celle de ruptures et répétitions temporelles (diagonales secondaires). Ces filtres présentent également une grandes variété, laissant ainsi penser que cette méthode pourrait mettre en lumière des frontières plus diverses qu'avec une seule représentation.

Table des figures

Liste des tableaux

Références

[Bimbot et al., 2011] Bimbot, F., Deruty, E., Gabriel, S., and Vincent, E. (2011). Methodology and resources for the structural segmentation of music pieces into autonomous and comparable blocks. 6

[Bimbot et al., 2012] Bimbot, F., Deruty, E., Sargent, G., and Vincent, E. (2012). Semiotic structure labeling of music pieces : Concepts, methods and annotation conventions. In *13th International Society for Music Information Retrieval Conference (ISMIR)*. 5

[Bisot, 2014] Bisot, V. (2014). Estimation de la structure musicale par approche dtw localement contraint et maximum likehood. Master thesis, UPMC, Paris, France. 8

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12 :2121–2159. 19, 20

[Foote, 2000] Foote, J. (2000). Automatic audio segmentation using a measure of audio novelty. In *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, volume 1, pages 452–455 vol.1. 6

[Fukushima, 1980] Fukushima, K. (1980). Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36 :193–202. :neocognitron

[Goto, 2003] Goto, M. (2003). A chorus-section detecting method for musical audio signals. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, volume 5, pages V–437–40 vol.5. 7

[Goto, 2006] Goto, M. (2006). Aist annotation for the rwc music database. pages pp.359–360. 6

[Grill and Schlüter, 2015] Grill, T. and Schlüter, J. (2015). Music Boundary Detection Using Neural Networks on Combined Features and Two-Level Annotations. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR 2015)*, Malaga, Spain. 8, 26, 30, 31, 32, 34

[Kaiser and Peeters, 2013] Kaiser, F. and Peeters, G. (2013). Multiple hypotheses at multiple scales for audio novelty computation within music. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 231–235. 6

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11). 12

[Mauch et al., 2009] Mauch, M., Cannam, C., Davies, M., Dixon, S., Harte, C., Klozali, S., Tidhar, D., and Sandler, M. (2009). Omras2 metadata project 2009. In *Proc. of ISMIR (Late-Breaking News)*. 6

[McFee and Ellis, 2014] McFee, B. and Ellis, D. (2014). Learning to segment songs with ordinal linear discriminant analysis. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 5197–5201. 6, 8

[McFee and Ellis, 2013] McFee, B. and Ellis, D. P. (2013). Dp1, mp1, mp2 entries for mirex2013 structural segmentation and beat tracking. 28, 29

[Mckay, 2006] Mckay, C. (2006). A large publicly accessible prototype audio database for music research. ismir. In *in Proc. of the International Conference on Music Information Retrieval (ISMIR)*. 9

[Muller et al., 2013] Muller, M., Jiang, N., and Grosche, P. (2013). A robust fitness measure for capturing repetitions in music recordings with applications to audio thumbnailing. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(3) :531–543. 7, 8

[Paulus and Klapuri, 2009] Paulus, J. and Klapuri, A. (2009). *Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music : 5th International Symposium, CMMR 2008 Copenhagen, Denmark, May 19-23, 2008 Revised Papers*, chapter Labelling the Structural Parts of a Music Piece with Markov Models, pages 166–176. Springer Berlin Heidelberg, Berlin, Heidelberg. 5

[Peeters, 2004a] Peeters, G. (2004a). A large set of audio features for sound description (similarity and classification) in the CUIDADO project. Tech. rep., IRCAM. 5

[Peeters, 2004b] Peeters, G. (2004b). *Computer Music Modeling and Retrieval : International Symposium, CMMR 2003, Montpellier, France, May 26-27, 2003. Revised Papers*, chapter Deriving Musical Structures from Signal Analysis for Music Audio Summary Generation : "Sequence" and "State" Approach, pages 143–166. Springer Berlin Heidelberg, Berlin, Heidelberg. 6

[Peeters et al., 2002] Peeters, G., Burthe, A. L., and Rodet, X. (2002). Toward automatic music audio summary generation from signal analysis. In *In Proc. International Conference on Music Information Retrieval*, pages 94–100. 6

[Peeters and Deruty, 2009] Peeters, G. and Deruty, E. (2009). Is music structure annotation multi-dimensional ? A proposal for robust local music annotation. In *Proceedings of the International Workshop on Learning the Semantics of Audio Signals (LSAS)*, pages 75–90, Graz, Austria. 5, 9

[Schluter and Bock, 2014] Schluter, J. and Bock, S. (2014). Improved musical onset detection with convolutional neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6979–6983. IEEE. 8

[Serrà et al., 2012] Serrà, J., Müller, M., Grosche, P., and Arcos, J. L. (2012). Unsupervised detection of music boundaries by time series structure features. pages 1613–1619, Toronto, Canada. AAAI Press, AAAI Press. 6, 7, 8

[Smith et al., 2011] Smith, J. B. L., Burgoyne, J. A., Fujinaga, I., Roure, D. D., and Downie, J. S. (2011). Design and creation of a large-scale database of structural annotations. 5, 6, 8, 9, 26, 38

[Ullrich et al., 2014] Ullrich, K., Schlüter, J., and Grill, T. (2014). Boundary detection in music structure analysis using convolutional neural networks. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, Taipei, Taiwan. 4, 6, 8, 17, 26, 27, 28, 29, 30, 32, 37, 38

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA : an adaptive learning rate method. *CoRR*, abs/1212.5701. 19, 21, 28