FOSMOR

Fooo Optical Sheet Music Recognition

Rapport de Soutenance

Seconde soutenance, le 10 Avril 2009

Félix *Flx* **Abecassis** (*abecas_e*)
Christopher *Vjeux* **Chedeau** (*chedea_c*)
Vladimir *Vizigrou* **Nachbaur** (*nachba_v*)
Alban *Banban* **Perillat-Merceroz** (*perill_a*)

Table des matières

Chapitre 1

Introduction

La première phase du projet jalonnée par la première soutenance a été l'occasion pour chacun des membres du groupe d'expérimenter et de mettre en place l'activité dont il était responsable.

Aujourd'hui le travail de chacun converge vers un même bloc : le prétraitement permet l'extraction des symboles de l'image avant de les acheminer vers le réseau de neurones responsables de la reconnaissance. Le résultat est mis en forme dans un format reconnaissable notamment par Lilypond pour être finalement retransformé en partition ou directement lu.

Chapitre 2

Prétraitement de l'image

2.1 Gommage du bruit

Les filtres utilisés lors de la première soutenance étaient linéaires, et celui retenu était un flou gaussien, qui pour chaque pixel effectuait une moyenne entre lui-même et ses huit voisins avec un coefficient différent en fonction de la distance du pixel concerné.

Cependant la méthode avait le défaut de modifier l'image source en créant, comme son nom l'indique, un flou. L'objectif était alors de mettre en place des filtres qui palliaient ce défaut tout en gommant les imperfections de l'image.

Le filtre médian, non linéaire, permet d'éliminer les valeurs extrêmes tout gardant une meilleure fidélité à l'image originale. Il est possible également de mélanger le filtre médian avec le filtre moyen : il suffit de faire la moyenne des trois ou des cinq valeurs médianes (parmi les neuf pixels).

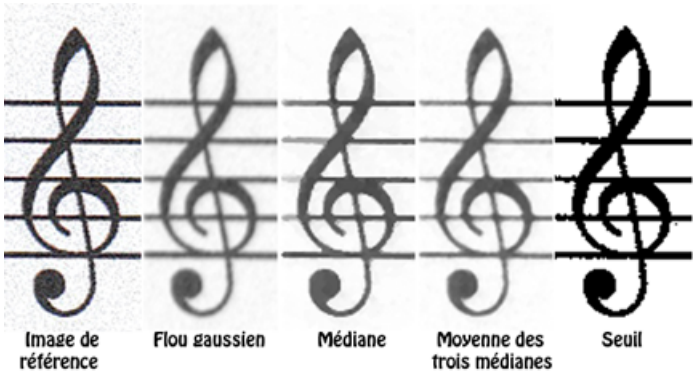

FIGURE 2.1 – Comparaison des différents filtres

2.2 Choix du biais

Pour le traitement de l'image, il était nécessaire de réduire le nombre de couleurs. Nous avions évoqué la possibilité d'utiliser de huit à deux niveaux de gris, et nous avons choisi d'en utiliser deux pour des raisons de simplicité et d'efficacité.

Chaque pixel de l'image doit donc être transformé en un pixel blanc ou un pixel noir une fois le prétraitement de l'image terminé. La difficulté devient donc de choisir un seuil à partir duquel on considère qu'un pixel est noir.

Il est facile de trancher lorsqu'il s'agit d'une seule image, il suffit en effet de tester tous les seuils (dans le cas présent cela consiste à afficher les 256 images générées correspondant à chaque seuil de l'image contenant 256 niveaux de gris à l'origine), de les comparer et de décider de manière quasiment objective quelle image est la meilleure.

On a ensuite le malheur de découvrir que quelques heures de travail supplémentaires seront nécessaires quand on découvre qu'avec une autre image de test le meilleur seuil est totalement différent de celui de la première image.

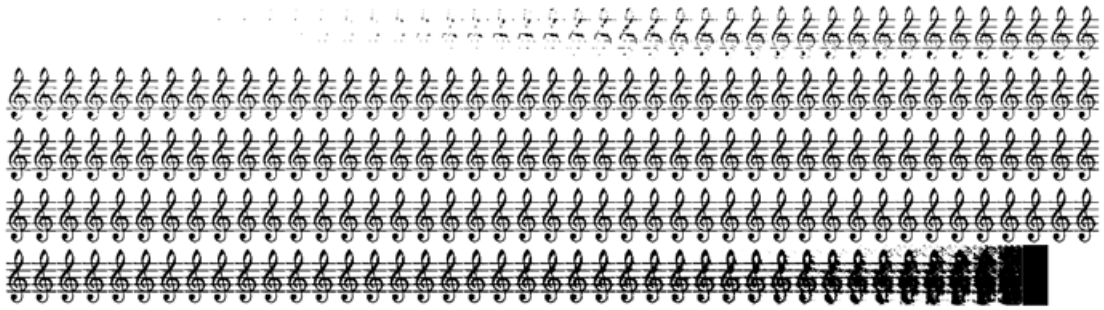

FIGURE 2.2 – Le choix du biais se fait par l'expérience

Chapitre 3

Analyse de l'image

3.1 Suppression des lignes

Lors de la première soutenance nous avons montré que nous étions capables de retrouver les lignes de portée sur une image d'une partition. Afin de pouvoir isoler les éléments plus simplement, nous allons tenter de supprimer ces lignes.

Cette phase est très délicate. Supprimer les lignes revient à perdre de l'information, il faut donc soigneusement réaliser cette tâche pour ensuite être capable de lire les notes correctement. Aussi, il faut que l'intégralité de la ligne soit supprimée, pour ne pas que les résidus soient mal interprétés par la suite.

On connait la largeur de la ligne ainsi que sa position. On va regarder les pixels au dessus de la supposée ligne, si ceux-ci sont blancs, alors on supprime la moitié haute de la ligne, ici représenté par les couleurs pâles. On effectue le même procédé pour la partie basse de la ligne ensuite.

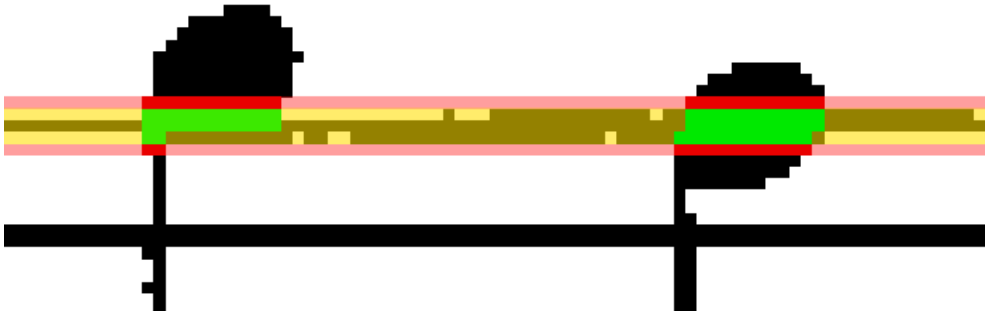

FIGURE 3.1 – Procédure de suppression des lignes

Cette procédure est très efficace à condition de connaître précisément la position de la ligne et sa largeur. Dans l'ensemble la détection et suppression des lignes est efficace, 90% des lignes sont bien retirées à l'approche des notes. Afin de pouvoir finaliser notre projet, nous avons décidé de ne pas passer plus de temps à essayer de résoudre ces petits problèmes mais de passer à l'extraction des notes.

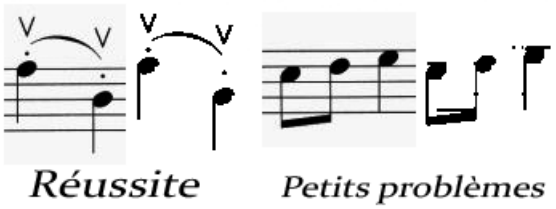

FIGURE 3.2 – Résultats avec des hauts et des bas

3.2 Détection des notes

Maintenant que nous ne sommes plus dérangés par les lignes, il est possible de passer à la détection des notes. Pour se faire, nous allons nous baser sur les verticales. En effet, la majorité des notes possèdent une verticale, ainsi que les barres de mesure qui vont nous aider par la suite.

La détection est relativement simple, nous partons à la recherche de segments verticaux d'une taille minimum, qui peuvent contenir des trous de faible dimension. Notre image est supposée redressée donc il n'est pas très important d'utiliser des algorithmes complexes pour cette étape.

Une fois trouvés, nous allons regarder pour chaque segment vertical ce qui se trouve en haut à droite de sa position. Cette partie de l'image va être envoyée au réseau de neurones afin de savoir s'il s'agit d'une boule pleine, d'une boule vide ou pas de boule du tout.

Suivant les conclusions du réseau de neurones, nous allons pouvoir agir en conséquence. S'il s'agit d'une boule, alors nous allons récupérer sa position par rapport aux lignes et la sauvegarder. Sinon, nous allons répéter la même étape pour la partie en bas à gauche. Si les deux parties ne sont pas des boules, alors nous ne sommes pas en présence d'une note. Il s'agit très certainement d'une barre de mesure, que nous ne traitons pas encore.

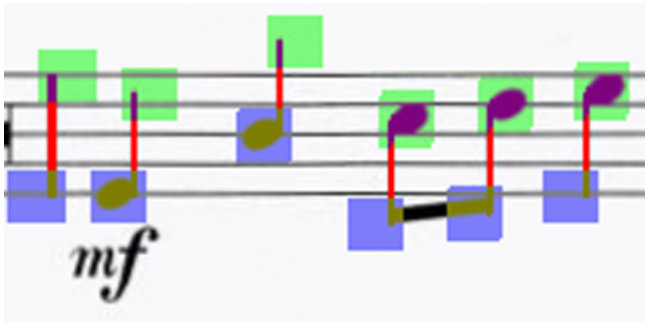

FIGURE 3.3 – Détection des verticales (en rouge) et mise en évidences des zones critiques

Malheureusement, le développement de la nouvelle version du réseau de neurone n'est pas encore terminé, donc nous utilisons celui de la première soutenance. Celui-ci n'est pas capable de faire la distinction entre des boules pleines ou vides, mais arrive tout de même à dire s'il y a une boule. De plus, étant donné la faible quantité de données, les résultats pour la partie basse des notes ne sont pas fiables du tout, donc nous avons ignoré ceux-ci pour le moment.

3.3 Avancement

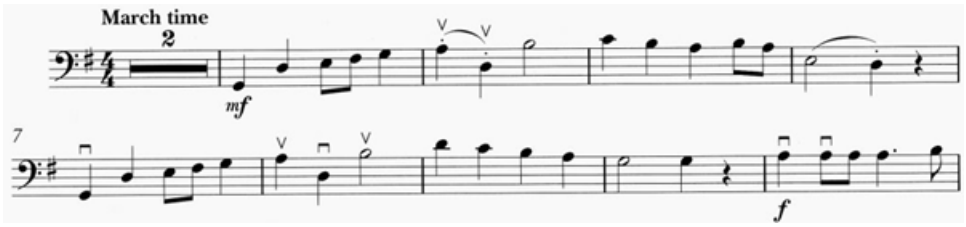

FIGURE 3.4 – Avant suppression

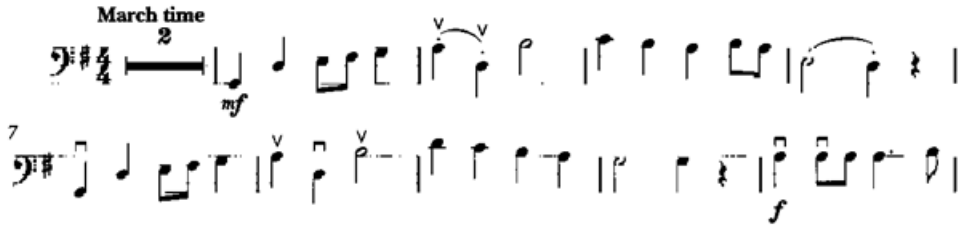

FIGURE 3.5 – Après suppression

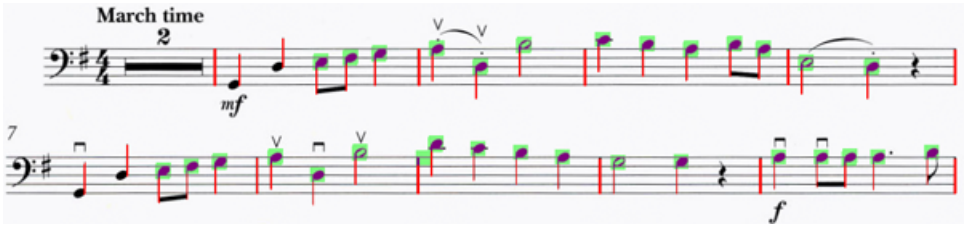

FIGURE 3.6 – Détection des notes hautes

Chapitre 4

Stockage et utilisation des données

4.1 Stockage

Pour pouvoir reconnaître la partition et l'utiliser dans son ensemble, il est important de mémoriser chaque nouvelle découverte au fur et à mesure. Pour cela il a fallu réfléchir à une structure de donnée capable de représenter en mémoire une partition, et notamment la variété de symboles, de modificateurs et plus généralement toutes les informations que nous arriverons à en tirer.

Chaque note est donc déterminée par une durée, une hauteur (qui selon l'armure et les altérations qu'on lui applique, peut varier), cette note peut être un accord, être liée...

Et il existe beaucoup d'autres genres de symboles.

On peut globalement retirer plusieurs catégories :

Le symbole ponctuel simple. C'est un simple signe qui n'a pas de variante et qui s'applique là où il est écrit. Un exemple est le point d'orgue.

Les symboles ponctuels variables, comme les silences ou les notes. Ces symboles sont simples à placer dans la structure de données, mais il faut pouvoir représenter leurs différentes variations :

un soupir peut varier en longueur (pause, demi-soupir...), et être affecté par un (ou plusieurs) points qui rallongent sa longueur

une note a les mêmes caractéristiques (longueur, points), mais a en plus une certaine hauteur, et possède encore d'autres caractéristiques (altération comme le dièse)

Enfin il existe les symboles qui s'appliquent sur une certaine zone, comme les répétitions, ou encore les crescendos.

Pour ceux-ci, la méthone la plus simple est probablement de distinguer 2

variantes pour chaque symbole, indiquant chacune s'il s'agit du symbole de début ou de fin.

Pour le moment la structure que nous avons créée n'est capable que d'accueillir des notes et des silences, avec les caractéristiques décrites plus haut pour ces 2 types de symboles.

Mais elle a été conçue pour pouvoir facilement rajouter de nouveaux types de données, et au fur et à mesure de notre avancement, il ne devrait pas être difficile de prendre en compte les nouveaux symboles reconnus.

4.2 Utilisation

Une fois une partition reconnue, et donc placée en mémoire vive, il pourrait se révéler utile d'en faire quelque chose avant de quitter le programme [1]. C'est pourquoi nous avons réfléchi à plusieurs utilisations de la partition. Il est en effet normalement possible, à partir des informations que nous avons, de générer une partition de musique imprimable, ou encore de la musique directement !

Dans le cas d'une partition imprimable, la génération d'une image serait probablement une des idées les moins appropriées. En effet, toutes les informations numériques concernant la partition seraient perdues dans la transformation en image : Si quelqu'un veut par la suite modifier cette partition scannée, il devra soit utiliser un logiciel de retouche d'image, soit utiliser un autre OMR qui lui permettra d'enregistrer dans un format de partition éditable... Il semble plus logique de directement proposer d'enregistrer dans un format de partition éditable.

Notre choix s'est porté vers le format du logiciel Lilypond, qui ressemble fortement à LaTeX dans la mesure où la syntaxe du fichier (texte) est proche, et qu'il permet d'exporter notamment en format PDF, PostScript, PNG, et même en musique MIDI.

Lilypond est un logiciel libre, disponible sous plusieurs plateformes, et il ne nous semble donc pas aberrant de proposer à un utilisateur de notre programme d'enregistrer dans ce format, qui est reconnu par de plus en plus de logiciels d'édition de musique.

Dans le cas de la musique, nous avons réfléchi à l'éventualité de jouer la musique directement dans notre programme, ou encore d'exporter au format MIDI (le format musical le plus adapté dans notre cas), mais depuis l'adoption du format lilypond pour exporter nos données, nous avons pensé que cette éventualité était désormais inutile, puisque le programme Lilypond permet lui-même de générer un fichier MIDI.

1. Moi j'dis ça...

Pour le moment, notre programme est capable d'exporter les partitions au format Lilypond. Comme le nombre de symboles susceptibles de se trouver dans une partition est relativement élevé, notre programme ne peut actuellement en mémoriser qu'une petite partie (ceux que nous trouvons les plus importants pour le moment). A mesure que notre programme sera capable de reconnaitre plus de symboles, il faudra donc également faire évoluer cette partie responsable de l'export au format Lilypond.

En attendant, voici l'exportation en PDF des notes lues sur la partition.

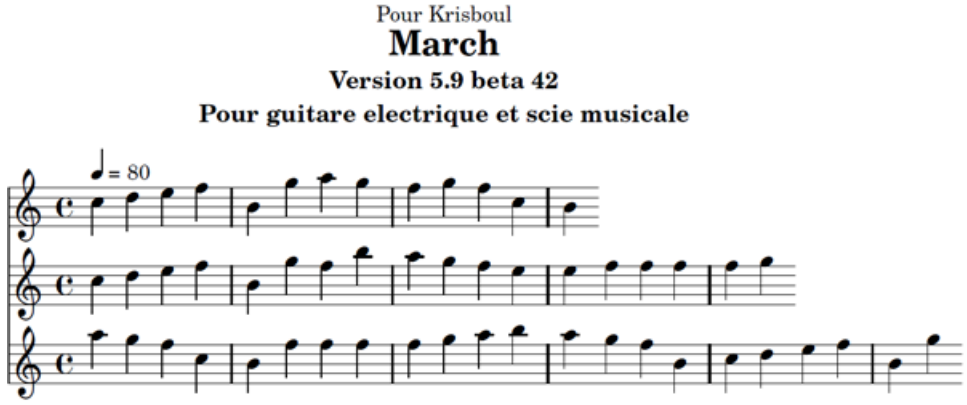

FIGURE 4.1 – Sortie en PDF

Chapitre 5

Caractérisation d'une image

5.1 Rappel de la problématique

Un réseau de neurone classique comme un perceptron multi-couches (MLP) n'est qu'un simple classifieur statistique. En se basant sur les résultats obtenus lors de l'apprentissage, il est capable, lorsque qu'on lui soumet une entrée, de donner son inférence sur sa classe d'appartenance.

Un simple perceptron multi-couches avec connectivité totale entre la couche d'entrée et la couche cachée possède le désavantage de présenter beaucoup trop de poids différents, et une faible capacité de généralisation car les caractéristiques de l'image ne sont pas extraites. La convergence est donc chaotique et les résultats ne sont pas suffisants puisque la moindre modification de la matrice peut entrainer un résultat erroné.

5.2 Exploration de certaines méthodes

Lors du rapport de première soutenance, nous avions donné des pistes de réflexion pour trouver une méthode efficace afin de caractériser une image avant de la soumettre à notre MLP.

Tout d'abord nous avons envisagé puis nous avons essayé de mettre au point une caractérisation par calcul des moments géométriques invariants de l'image en utilisant les polynômes proposés par Fritz Zernike, qui sont des polynômes complexes orthogonaux sur le cercle unitaire.

Cette tentative n'a pas été concluante car les informations sur cette méthode ne sont pas abondantes, les explications sont parfois contradictoires et peu d'exemples solides existent hormis en Matlab, qui est le langage le plus incompréhensible que je puisse concevoir.

Nous avions aussi émis la possibilité d'utiliser une approche par placement de bâtonnets sur l'image. L'idée est de placer des bâtonnets sur l'image et de voir si certains bâtonnets sont effectivement en intersection avec une partie du symbole. Le placement des bâtonnets étant déterminé durant la phase d'apprentissage en utilisant une heuristique avancée.

Nous avons effectué des recherches sur le sujet, mais nous n'avons pas été convaincus sur la pertinence de cette méthode pour résoudre efficacement notre problème de caractérisation d'une image. C'est une méthode qui peut se relever efficace mais qui est difficile a mettre en oeuvre, le processus heuristique demande des algorithmes génétiques ce qui aurait demandé des recherches avancées dans ce domaine également, alors que les réseaux de neurones eux-mêmes sont déjà infiniment complexes et toujours améliorables.

En se basant sur le constat précédent, nous avons décidé finalement que la caractérisation et la classification seraient effectuées par le réseau de neurones lui-même en utilisant une méthode à la pointe de la recherche : les réseaux de neurones à convolution dont la structure sera expliquée dans le prochain chapitre.

Chapitre 6

Réseau de neurones

6.1 Convolution

Lors de la précédente soutenance nous avons présenté un simple MLP à une seule couche cachée capable d'apprendre le XOR et de reconnaitre des chiffres de 0 à 9 afin de prouver qu'il était capable de résoudre des problèmes non linéairement séparables.

Comme mentionné au dessus, nous avons décidé d'inclure la caractérisation dans le réseau de neurone en utilisant un réseau de neurone à convolution (CNN).

La convolution est l'application d'un opérateur, c'est une méthode souvent utilisée en traitement d'image. Ici l'originalité et la force de cette méthode c'est que nous n'avons pas à définir la valeur de l'opérateur à réaliser localement, elle est déterminée automatiquement lors de l'apprentissage. Nous assimilons la valeur de cet operateur aux poids des connexions entre neurones. Afin que l'operateur soit appliqué à plusieurs pixels, nous allons tout simplement partager les poids pour plusieurs connexions. Un neurone va par exemple être l'application du même operateur à un champ réceptif de 5x5 pixels de la couche précédente. Ainsi les CNN ne considèrent pas comme les MLP que les pixels sont indépendants et que leur agencement n'est pas significatif. Les CNN s'intéressent, grâce à l'application de ces opérateurs locaux sur un ensemble de pixels, à leur organisation spatiale.

6.2 Structure

Seule la structure a changée dans notre réseau de neurones, c'est ici l'avantage de cette méthode. L a propagation, la fonction d'activation et la rétropropagation se calculent toujours de la manière que nous avions décrite dans le rapport de la soutenance 1. Nous allons présenter plus en détail la structure de notre réseau de neurones utilisée pour nos tests, sachant que pour l'instant nous reconnaissons toujours les chiffres de 0 à 9.
La couche d'entrée est simplement l'image à reconnaitre ou à apprendre en taille normalisée 29x29.

La première couche est une couche de convolution composée de 6 cartes de caractéristique. Nous ne prenons que 1 pixel sur 2 pour notre carte de caractéristique afin d'obtenir de meilleurs résultats. Chaque neurone de cette couche est le résultat de l'application de la convolution à une zone de 5x5 pixels l'image de base. Il y a donc 13 possibilités pour les colonnes et 13 pour les lignes. Ces cartes sont donc de dimension 13x13.

Et il y a (5x5 + 1)x6 = 156 poids (en comptant un poids pour le biais) et 13x13x6 = 1014 neurones.

La deuxième couche est basée sur le même principe en utilisant 50 cartes de caractéristiques de taille 5x5 dont chaque neurone est l'application d'une convolution de 5x5 depuis la couche précédente.

On a donc ici (5x5+1)x6x50 = 7800 poids et 5x5x50 = 1250 neurones.

La troisième couche est une couche cachée classique, avec 100 neurones entièrement connectés à la couche précédente soit 100x(1250+1) = 125100 poids.

Enfin, la couche de sortie est composée de 10 neurones (on utilise une méthode de 1 parmi N pour N classes d'appartenances possibles), eux aussi entièrement connectés aux neurones de la couche précédente.

Nous obtenons donc l'impressionnant total de 3215 neurones, 134066 poids, et 184974 connexions pour relier nos neurones.

Nous montrons à la page suivante une schéma général de la structure de notre CNN.

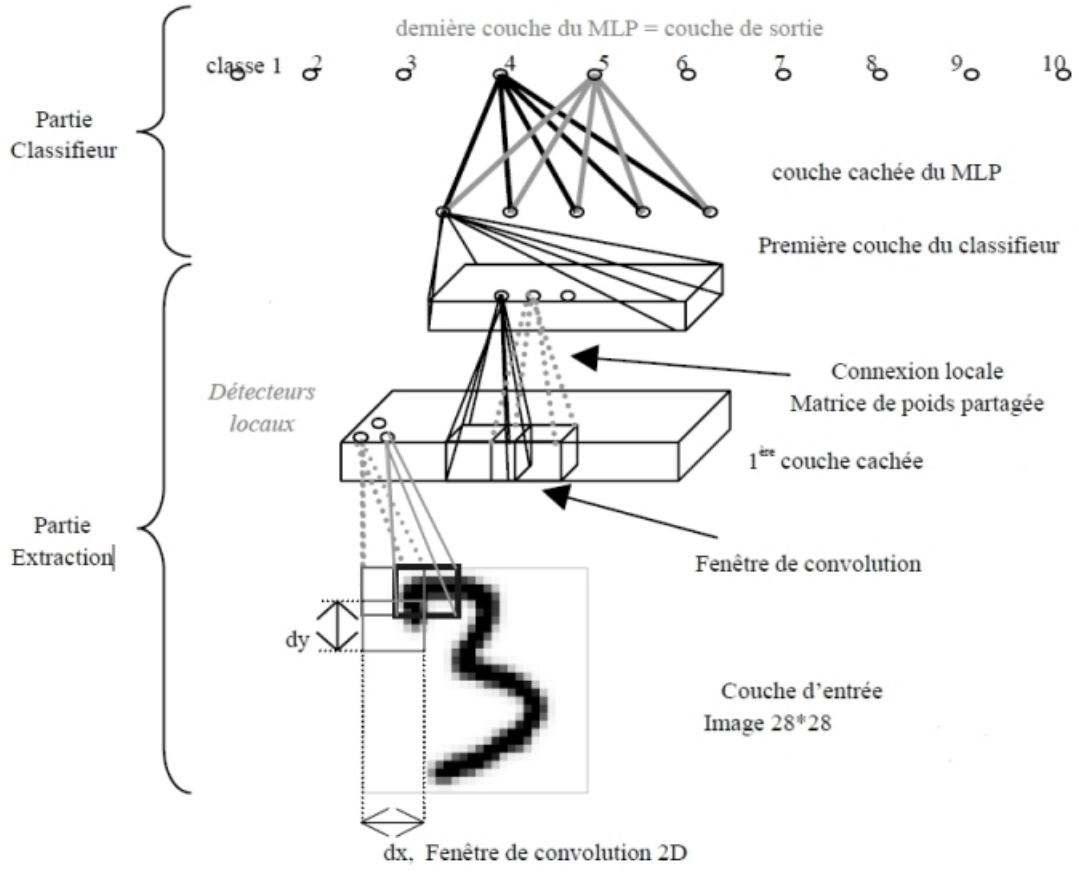

FIGURE 6.1 – Réseau de neurones à convolution

6.3 Objectifs

Des problèmes persistent encore. Tout d'abord l'apprentissage d'un nombre si important de poids rend le processus très couteux et très long. Un apprentissage de plusieurs milliers d'éléments (comme un extrait de la base MNIST) dure de nombreuses minutes.

L'étude d'articles scientifiques nous a orientés vers certaines méthodes mathématiques pour rendre l'apprentissage plus rapide. Par exemple en utilisant l'algorithme de Levenberg-Marquardt pour minimiser l'erreur. Ou alors en effectuant une rétropropagation de second ordre en utilisant la matrice hessienne. Une autre technique exploitable et plus intuitive serait de ne pas rétropropager certaines erreurs jugées suffisamment faibles. Cependant le « suffisamment » doit encore être défini et bien calibré, si la valeur choisie est trop haute, le réseau de neurones serait trop permissif et réaliserait probablement trop d'erreurs sur la base de test.

La base de données à notre disposition peut être importante, mais elle n'est pas toujours suffisante, afin d'augmenter la capacité de généralisation du réseau de neurones. Pour cela on pourrait appliquer des distorsions élastiques à certaines images de la base d'apprentissage pour modifier certaines caractéristiques spatiales du symbole, tout en gardant la même allure générale. Notre CNN devrait donc être capable de reconnaitre efficacement des symboles assez éloignés de notre base de test initiale. Nous donnons un exemple de distorsion que nous pourrions réaliser sur le symbole 3.

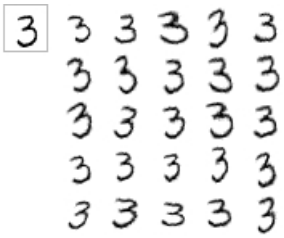

FIGURE 6.2 – Distorsions élastiques sur le symbole 3

Chapitre 7

Conclusion

Après plusieurs mois passés sur ce projet, nous en comprenons de mieux en mieux les enjeux autant que les problèmes liés à la reconnaissance de partitions musicales. Nous appréhendons notamment désormais plus facilement les types de symboles qui risquent de nous poser problème, à cause de leur forme, ou plus souvent à cause de la manière selon laquelle ils sont inscrits au sein d'une partition.

Cette seconde soutenance est très prometteuse, nous avons réussi à réunir nos divers travaux afin d'obtenir une détection de quelques notes d'une partition. La base du projet est construite, les futurs développements vont s'appuyer sur celle-ci pour lire une partition complète.

En attendant, il est possible de suivre notre avancement sur notre site web [1]

1. http://perso.epita.fr/~nachba_v/fosmor - Hébergement temporaire

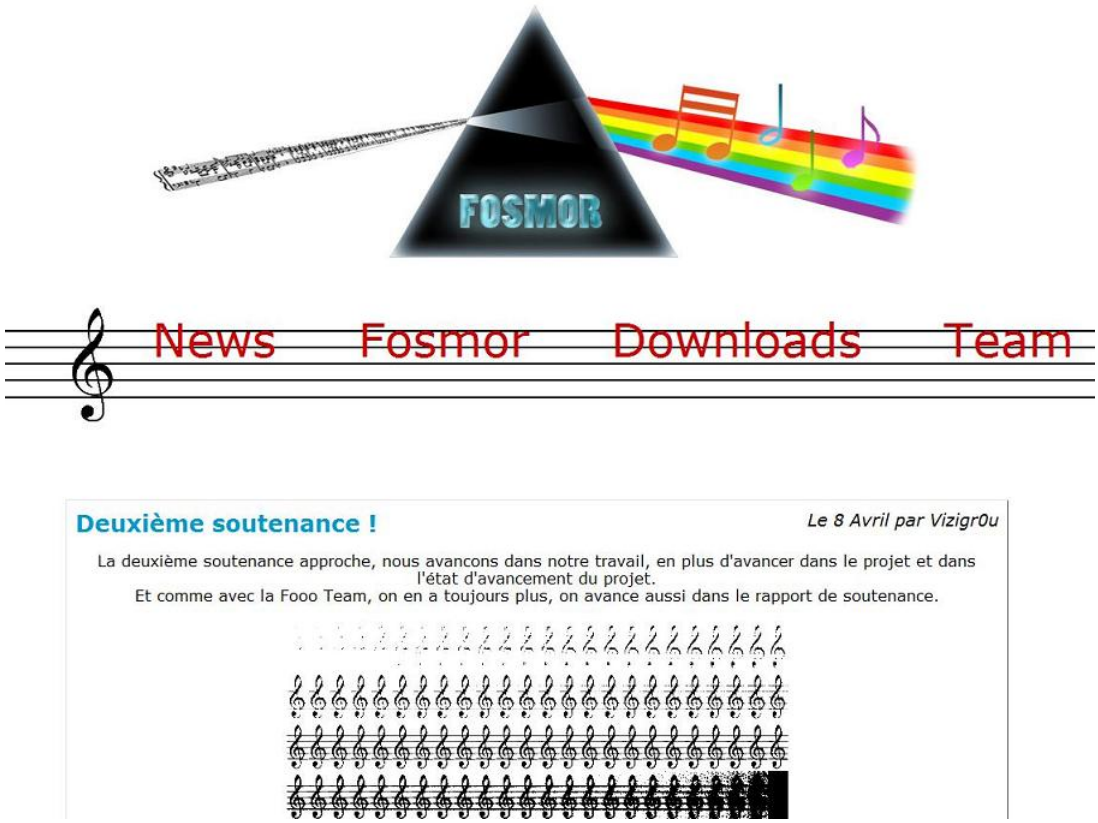

FIGURE 7.1 – Le site web du projet FOSMOR