FOSMOR

Fooo Optical Sheet Music Recognition

Rapport de Projet

Soutenance finale, le 26 Mai 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

L'OMR, ou Optical Music Recognition, consiste en la reconnaissance optique d'une partition de musique dans le but de la rendre compréhensible de la machine. A partir de là différentes optiques sont envisageables, les plus communes étant la lecture de la musique, ou simplement la retranscription en une nouvelle partition.

C'est dans ce projet ambitieux que nous nous sommes lancés, et ce n'est pas sans effort que nous sommes arrivés au moindre résultat. De nombreuses étapes sont nécessaires pour reconnaitre une partition, du prétraitement de l'image à la reconnaissance des formes, en passant par la localisation des différents symboles, et pour finir, l'exportation dans le format désiré.

Toutes ces étapes interconnectées entre elles et chacun des membres de la Fooo Team a travaillé dur pour arriver au résultat que nous avons aujourd'hui.

Chapitre 2

Prétraitement de l'image

2.1 Chargement de l'image

Afin de prendre en charge le plus possible de formats d'image, et pour ne pas perdre de temps inutilement à gérer les problèmes spécifiques à tous ces formats, nous nous sommes aidés d'une bibliothèque de fonctions, ImageMagick, qui nous a permis de convertir toutes les images entrées par l'utilisateur dans un format commun sur lequel nos algorithmes se basent.

2.2 Gommage du bruit

Pour faciliter la tâche des opérations ultérieures il est nécessaire de nettoyer l'image de toutes les impuretés qui ne font pas partie de l'image, comme le bruit par exemple.

Après plusieurs expérimentations de filtres linéaires et non linéaires appliquées à chaque pixel de l'image, le filtre retenu a été un mélange de filtre médian et de filtre de flou gaussien : chaque pixel est comparé à ses huit pixels environnants, les neufs pixels sont classés par valeur croissante, et le pixel est remplacé par la moyenne des pixels classés en position 4, 5 et 6, ce qui correspond au pixel médian et aux deux qui l'entoure.

2.3 Rotation de l'image

Une étape principale du prétraitement de l'image est la rotation de celle-ci pour qu'elle soit parfaitement droite. La détection de l'inclinaison de l'image se fait au moment de la détection des lignes, il suffit donc d'appliquer une rotation en fonction d'un angle donné.

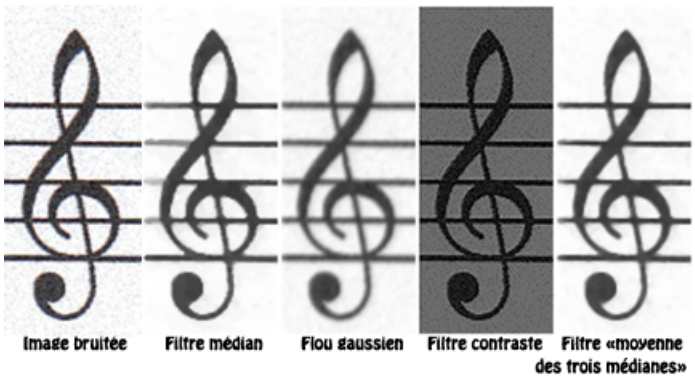

FIGURE 2.1 – Résultats des différents algorithmes sur une image fortement bruitée

La rotation d'une image induit forcément La rotation d'une image induit forcément une approximation dans l'image de destination. Pour calculer au mieux l'image cette approximation, deux méthodes principales existent : les interpolations bilinéaire et bicubiques.On retiendra l'interpolation bilinéaire : en effet, la qualité est légèrement inférieure à l'interpolation bicubique mais son temps d'exécution est largement inférieur, ce qui nous pousse à choisir cette solution.

2.4 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.

Pour sélectionner le meilleur biais possible il était nécessaire de comparer plusieurs images issues de la même source à laquelle on a appliqué différents biais.C'est seulement après de nombreux tests sur plusieurs images qu'il a été possible de définir un biais générique convenant à toutes les images.

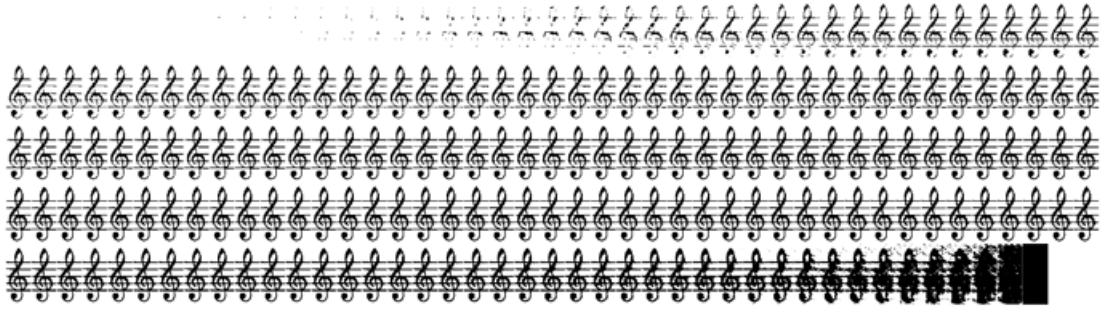

FIGURE 2.2 – Le choix du biais se fait par l'expérience

Chapitre 3

Analyse de l'image

Pour finaliser le projet, nous nous sommes attardés sur la robustesse du programme et son adaptabilité. En effet, un certain nombre de constantes qui étaient autrefois déterminées par l'expérience prennent maintenant en compte les données de la partition.

3.1 Sélection des lignes

Lors des précédentes soutenances nous avons réussi à déterminer avec fiabilité où étaient les lignes. Malheureusement, des artefacts étaient présent dans des lignes comportant du texte par exemple. Pour garder uniquement les lignes de portées, il a fallu se rappeler leurs caractéristiques.

Chaque portée est un ensemble de 5 lignes horizontales, chacun à égale distance de ses voisins. Nous avons à disposition un ensemble de lignes chacune définie par son ordonnée. Notre technique pour sélectionner les bonnes lignes consiste en deux étapes.

Tout d'abord, nous calculons l'écart médian entre tous les couples de lignes voisines. En faisant l'hypothèse que le nombre d'artefact est moins important que le nombre de lignes licites, on arrive grâce à cette technique à savoir l'intervalle entre deux lignes de portée.

Ensuite, on ne va garder chaque ligne que si elle possède une ligne voisine (au sens large) qui se situe aux environs de l'intervalle trouvée précédemment. Ainsi, on arrive à isoler des groupes de lignes. On peut sans trop de soucis supprimer les groupes qui ne comportent pas 5 éléments.

3.2 Détection des notes

Afin de trouver détecter les notes, on procède en deux étapes. Tout d'abord, on recherche les barres verticales, puis on regarde en haut à droite ainsi qu'en bas à gauche de la barre en quête des potentiels ronds.

Il faut faire attention au fait que les barres font plusieurs pixels de large. Pour ne pas rechercher plusieurs fois chaque rond, il faut trouver un moyen de fusionner ces pixels. Nous avons choisi une solution simple mais efficace. Lorsque nous détectons une ronde, nous allons marquer les pixels en question indiquant qu'ils ont déjà été analysés. Ainsi, ils ne vont pas pouvoir être réutilisés pour détecter une autre ronde.

Une fois que nous avons détecté la position d'une ronde, nous l'envoyons au réseau de neurone qui va déterminer sa valeur. Une fois celle-ci acquise, nous allons la comparer à sa position par rapport aux lignes pour obtenir la hauteur de la note.

Armé de toutes ces données, il ne reste plus qu'à les transmettre à l'API lilypond afin d'avoir un rendu utilisable.

Chapitre 4

Stockage des données

4.1 Evolution du projet

A mesure que nous avancions, certaines parties ont changé de principe de fonctionnement, et d'autres, comme le réseau de Neurones, ont complètement été refondues. Le code étant de plus en plus tourné en OCaml pour la partie précédant la mémorisation des données, nous avons rajouté une API afin de créer une partition et de l'exporter en format lilypond directement en OCaml, alors que toute cette partie était codée en C auparavant. Il a donc fallu bien comprendre l'interaction OCaml-C, afin de réaliser cette surcouche à l'API déjà existante.

Nous avons principalement été bloqué par certains problèmes de compatibilité entre le PIE et nos machines (MacOs, ArchLinux, Ubuntu..) dans cette partie. Désormais, il est possible aussi simplement pour un codeur en C que pour un codeur OCaml de créer une partition, de lui donner un titre, un tempo, d'y ajouter des notes, silences, et enfin de l'exporter au format Lilypond.

4.2 Evolution de l'API

Avec l'avancement du projet, les capacités de reconnaissance de signes ont évolué. Il a donc fallu améliorer le système de mémorisation et d'export de partition afin qu'il puisse gérer de plus en plus de données. En effet, afin de ne pas ralentir le projet par cette partie, il fallait en permanence prévoir de gérer des signes en phase d'être ajoutés à la liste de tous ceux que nous reconnaissions. Typiquement, notre OMR est capable de reconnaitre les noires et les blanches, et de plus en plus de hauteurs différentes de notes. Pour être en avance sur la reconnaissance, l'API de mémorisation et exportation de

partitions est capable de gérer les altérations (dièses, bémols, bécarres), mais aussi les notes pointées ou doublement pointées, les silences de différentes durées, etc.

Voici un exemple de code pour exporter une version simplifiée de la musique "Au Clair de la Lune" sur la sortie standard en C :

```c
#include "tsheet.h"

int main(void)
{
  char notes[11] = {DO, DO, DO, RE, MI, RE, DO, MI, RE, RE, DO};
  tSheet *p = sheetMake("Au Clair de la Lune", QUARTER, 80);

  for (int i = 0; i < 11; i++)
    sheetNoteAdd(p, QUARTER, notes[i], 0, 0);
  sheetToLily(stdout, p);
  sheetBurn(p);
  return 0;
}
```

Le code OCaml suivant quant à lui exporte une version simplifiée de "Au Clair de la Lune" dans le fichier 'clair.ly' :

```ocaml
open ToLily

let _ =
  (
    let sheet = new tSheet "Au Clair de la Lune" 4 80 in
      sheet#addNotes [(Quarter, Do); (Quarter, Do); (Quarter, Do);
                      (Quarter, Re); (Half, Mi); (Half, Re);
                      (Quarter, Do); (Quarter, Mi); (Quarter, Re);
                      (Quarter, Re); (Full, Do)];
      (sheet#export "clair.ly");
      sheet#delete ();
      exit 0
  )
```

Il est intéressant de voir que cette API est manipulable aussi facilement en C qu'en OCaml. Il est donc envisageable de la réutiliser pour un projet tout autre.

Chapitre 5

Problématique de la caractérisation d'une image

5.1 Nécessité d'une caractérisation

Nous allons ici analyser les problèmes que pose la caractérisation d'une image avant de pouvoir la soumettre à notre réseau de neurones.

Un réseau de neurones prend sur son entrée des données abstraites, cela n'est donc pas forcément la représentation telle quelle de l'image mais cela peut être le résultat de méthodes mathématiques pour extraire des caractéristiques significatives de ces images.

Il est d'ailleurs indispensable de traiter l'image avant toute tentative de classification statistique, par exemple il est nécessaire dans tous les cas d'éliminer le bruit, de passer en niveaux de gris, de centrer le motif, d'effectuer une rotation si nécessaire, ou tout autre traitement qui peut être indispensable avant l'application d'un opérateur mathématique. Par exemple la caractérisation d'une image par moments géométriques nécessite une normalisation des données.

Même si nous avons à notre disposition des images de taille identiques et parfaitement centrées, nous ne pouvons concevoir de soumettre l'image telle quelle à un perceptron multi-couches, en effet pour constituer notre base d'apprentissage nous avons besoin d'un grand nombre de motifs de chaque classe, et ils doivent présenter une morphologie différente pour espérer augmenter la capacité de généralisation du réseau de neurones. Dans le cas du perceptron multi-couches, que nous avions implanté lors de la première soutenance, c'est l'intensité des pixels selon leurs positions qui est prise en compte pour la reconnaissance, sans même tenir compte de l'agencement géométrique de ces pixels.

Les limites d'un tel modèle apparaissent très rapidement lors des tests, la capacité de généralisation est quasi-nulle, nous ne pouvons espérer que reconnaitre des éléments de la base d'apprentissage.

Même sur la base d'apprentissage, le perceptron multi-couches montre ses faiblesses, par exemple sur une base d'apprentissage conséquente de chiffres manuscrits de 0 à 9, il existe de nombreuses manières de faire par exemple un 9 selon la personne, même si l'image est centrée, certaines personnes le font plus ou moins penché, la boucle plus ou moins grande, etc. On peut alors parfois obtenir des résultats catastrophiques sur la base d'apprentissage ! Le perceptron s'affolant à chaque nouvelle représentation un minimum exotique d'un motif qu'il a déjà appris, il ne réussira jamais à converger.

5.2 Différentes solutions

Nous nous sommes donc rapidement dirigés vers la recherche de solutions plus viables. Nous allons d'abord mentionner les solutions que nous avons envisagées et même expérimentées pour certaines, puis nous détaillerons enfin la solution finale que nous avons trouvée et appliquée.

Représentation vectorielle : une idée intuitive pour dépasser la simple représentation matricielle serait d'analyser et de ne garder que les caractéristiques géométriques de l'image, par exemple la localisation et la taille des segments, des cercles, des ovoïdes... C'est toutefois une méthode complexe et difficile que nous n'avons finalement pas retenue, car elle s'adapte mal avec un réseau de neurones classique.

Analyse par segments : une méthode intéressante sur le plan algorithmique, il s'agit de placer sur notre image de nombreux bâtonnets selon des positions et des tailles déterminées par un algorithme génétique et de voir quels segments coïncident avec notre motif. C'est cependant un processus très délicat à mettre en oeuvre, et nous n'avons pas été convaincus par sa pertinence à résoudre notre problème.

Moments de Zernike : ce sont des moments géométriques invariants par translation et homothétie, ils possèdent une bonne résilience au bruit et sont réputés pour obtenir de bons résultats. Cependant nous n'avons trouvé que très peu d'explications pertinentes sur ce sujet. Le calcul de ces moments implique des connaissances mathématiques avancées, cette méthode utilise notamment des polynômes complexes orthogonaux sur le cercle unitaire. Les divers articles scientifiques que nous avons pu trouver rabâchaient les mêmes ébauches théoriques avec la même formule avant de partir dans des considérations pour expliquer comment ils ont amélioré les performances du modèle déjà existant, sans expliquer réellement l'essence de son fonctionnement. La tentative de mise en place de cette méthode pourtant prometteuse s'est donc interrompue pour laisser la place à notre solution définitive.

Le salut est venu du Dr. LeCun [LeCun 1989] avec sa proposition de réseau de neurones à convolution développé au AT&T Bell Labs. Cette méthode présentait l'avantage d'être une évolution du perceptron multi-couches, en gardant une partie de la structure originale. Nous allons donc présenter sa structure et les résultats que nous avons obtenus dans la partie suivante dédiée aux réseaux de neurones.

Chapitre 6

Problématique de la classification statistique

6.1 Convolution

Comme précisé plus haut, notre choix s'est finalement porté sur une structure de réseau de neurones à convolution (CNN).

La propagation se fait de manière classique depuis l'entrée vers la sortie, l'information est véhiculée avec une intensité plus ou moins grande selon un calcul prenant en compte la somme des poids de la couche précédente et une fonction d'activation pour lisser cette somme, dans notre cas nous avons choisi la tangente hyperbolique dont la dérivée est simple à calculer.

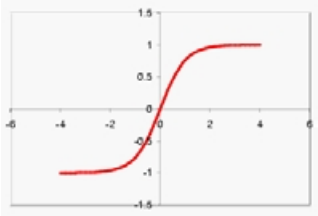

FIGURE 6.1 – La tangente hyperbolique

L'apprentissage est supervisé en utilisant l'algorithme classique de rétropropagation de l'erreur quadratique aussi appelé rétropropagation du gradient. Cet algorithme consiste à déterminer l'erreur commise par chaque neurone puis à modifier la valeur des poids pour minimiser cette erreur. Tout d'abord on effectue une propagation de l'entrée à travers le réseau par la méthode de calcul détaillée plus haut afin de déterminer l'erreur commise par chaque neurone de sortie. Ensuite l'algorithme consiste à remonter progressivement cette erreur depuis les sorties jusqu'à l'entrée en modifiant les poids à la remontée, en utilisant notamment la dérivée de la fonction d'activation pour minimiser l'erreur.

Les points qui viennent d'être précisés ne sont pas originaux par rapport à un perceptron multi-couches classique. C'est dans la gestion des poids que réside la puissance de notre nouveau modèle.

La convolution est un opérateur mathématique souvent utilisé en traitement d'images, mais c'est souvent au programmeur de le définir.

Ici, nous appliquons un opérateur local, mais la valeur est définie automatiquement par l'apprentissage ! Nous assimilons la valeur de cet operateur aux poids des connexions entre neurones. Afin que l'opérateur 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 perceptrons multi-couches que les pixels sont indépendants et que leur agencement n'est pas significatif. Les réseaux de neurones à convolution s'intéressent, grâce à l'application de ces opérateurs locaux sur un ensemble de pixels, à leur organisation spatiale.

6.2 Structure

Détaillons maintenant la structure de notre nouveau réseau de neurones, il comporte 4 couches en plus de l'entrée : 2 couches de convolution qui forment la partie extraction, et une couche cachée habituelle suivie d'une couche de sortie qui forment la partie classification. Pour la sortie nous utilisons la méthode 1-parmi N, c'est à dire qu'une seule sortie sera activée (proche de 1) correspondant à l'inférence du réseau de neurones sur la classe d'appartenance, les autres seront proches de -1.

La couche d'entrée est simplement l'image à reconnaitre ou à apprendre en format 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 de 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, 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.

Ceci peut paraitre impressionnant, et la place en mémoire et le temps d'apprentissage sont en effet conséquents, mais c'est la norme pour de tels réseaux de neurones.

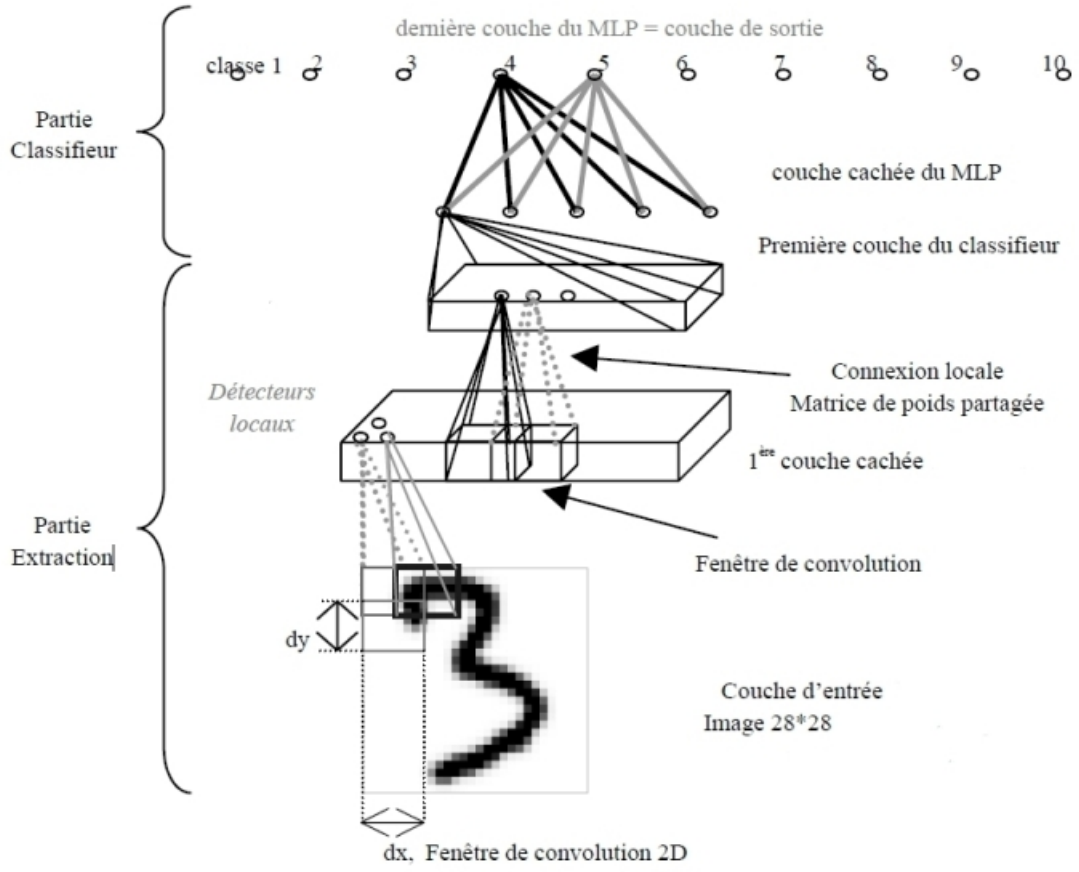

FIGURE 6.2 – Structure du réseau de neurones à convolution

6.3 Méthodes avancées

Nous avons ensuite développé des méthodes supplémentaires pour accélérer la convergence et les performances. Tout d'abord nous avons utilisé une technique de second ordre pour améliorer la vitesse de convergence. Les modèles mathématiques sur la recherche de minimum de fonction nous ont été utiles ici, au lieu d'appliquer le même coefficient d'apprentissage pour chaque poids, nous affectons un coefficient qui va symboliser son importance. Ce coefficient est trouvé en calculant la matrice Hessienne issue de la rétropropagation de second ordre de l'erreur sur un échantillon de la base d'apprentissage.

La deuxième méthode que nous avons utilisé consiste à augmenter la taille de la base d'apprentissage ce qui va augmenter la capacité de généralisation du réseau de neurones. Pour cela, nous appliquons des distorsions élastiques aux images de base avant de les soumettre pour l'apprentissage, ainsi notre CNN voit des motifs différents à chaque fois et il est donc bien forcé de se concentrer sur les caractéristiques essentielles de chaque chiffre ! Pour générer des distorsions, nous créons un noyau gaussien, puis nous appliquons la distorsion à l'image avec un aspect aléatoire.

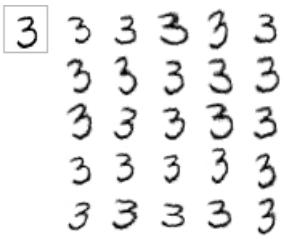

FIGURE 6.3 – Distorsion sur le chiffre 3

6.4 Résultats

Nous avons tout d'abord testé l'efficacité de notre nouveau réseau de neurones sur la base MNIST des chiffres manuscrits de 0 à 9. Cette base comporte 60 000 chiffres pour l'apprentissage. L'apprentissage des 60 000 motifs se fait environ en 15 minutes, et nous obtenons déjà environ 90% de reconnaissance sur cette même base. En appliquant des distorsions, et avec la méthode de la matrice Hessienne tout en effectuant plusieurs itérations sur le flot des 60 000 motifs, nous obtenons un taux de réussite de 98%.

Il serait également possible d'augmenter encore davantage ce taux avec plus d'itérations sur la base d'apprentissage, mais cet apprentissage deviendrait extrêmement lent, et chaque modification demanderait plusieurs heures de test avant de savoir si elle est concluante.

Revenons maintenant à notre sujet, c'est à dire la reconnaissance de partitions de musique, le problème principal qui s'est posé c'est que nous ne disposions pas d'une base de donnée comparable pour faire un apprentissage correct. Il a fallu séparer quelques notes à la main pour commencer à avoir un embryon de base d'apprentissage, mais ce n'était toujours pas suffisant. C'est le problème principal actuellement, le réseau de neurones fonctionne en théorie car il apprend bien la base MNIST, mais notre base de données pour notre problème est vraiment trop petite, les résultats risquent d'être légèrement chaotiques.

Chapitre 7

Interface utilisateur en ligne

La plateforme FreeBSD n'étant pas très répandue sur les postes clients du grand public, une bonne solution pour faire profiter de l'application au plus grand nombre de personnes est de créer une interface graphique en ligne qui permet à quiconque connecté à internet d'utiliser le programme, qui s'exécute sur un serveur dédié.

Or nous avons la chance de posséder ce genre de serveur, qui plus est sous FreeBSD. Php se charge donc d'exécuter le projet fosmor et d'afficher le résultat sur une simple page web.

L'interface utilisateur en ligne est disponible à l'adresse suivante :

http ://omr.fooo.fr

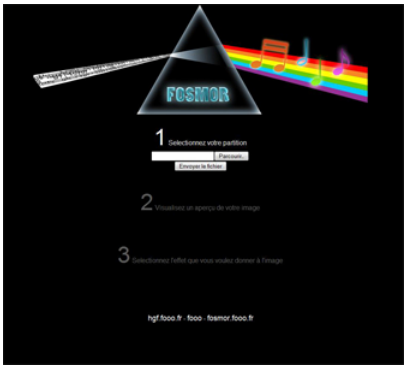

FIGURE 7.1 – L'interface utilisateur en ligne

Chapitre 8

Conclusion

Le projet FOSMOR a permis à notre groupe de faire un pas dans le monde de l'OMR, la reconnaissance optique de partitions musicales.

Lorsque nous avons choisi le projet nous pensions aborder un problème très similaire à celui de la reconnaissance de caractères. Nous nous sommes vite aperçu que la partition de musique, même lorsqu'elle était générée par un logiciel qui l'écrit d'une manière plus ou moins standard, était en fait truffée de différents types de symboles. Contrairement à du texte, ne sont pas tous alignés sur une même ligne et apparaissent parfois superposés, ou à des endroits spécifiques. Il suffit de se représenter un accord avec un do dièse et un là, liée à une blanche pointée à la mesure d'après, le tout joué piano en crescendo vers du fortissimo. Il est donc difficile de cerner tous les endroits où il est possible de rencontrer un type de symbole. Les premiers articles que nous avions lus sur l'OMR nous avaient prévenus : la reconnaissance optique de partitions est très compliquée, et les pistes de recherche sont pour le moment faibles.

En plus de l'aspect technique de la reconnaissance de partitions, nous souhaitions obtenir un projet intéressant de par son utilité. C'est pourquoi nous avons travaillé sur certains domaines comme le fonctionnement en ligne de commande pour permettre l'automatisation. Ceci permet de profiter de la souplesse de la ligne de commande pour par exemple traiter automatiquement de multiples partitions, par exemple pour lancer une numérisation de toute une collection de partitions.

C'est dans cette même optique de facilité d'utilisation que nous avons réalisé l'interface Web. Elle permet en effet à n'importe qui ayant un accès à Internet de numériser sa partition. Un exemple pratique d'utilisation serait de prendre une partition en photo depuis son téléphone, de se connecter au site web et d'obtenir directement la sonnerie au format MIDI.

D'autres aspects, même si nous n'avons pas été en mesure d'aller jusque là, nous paraissaient importants. On peut noter par exemple le fait de ne pas s'arrêter à un type de partition prédéfini (par exemple, généré par le logiciel X ou Y), et même si nous savions que nous n'atteindrions pas le stade de la partition manuscrite, nous avons gardé en tête le concept d'adaptabilité.

Nous sommes très satisfaits de tout ce que nous aura fait apprendre le projet cette année. L'OMR est un domaine passionnant. Alors que l'approche avec ce projet, qui nous était inconnu, était totalement différente de celle que nous avons eu avec le projet de l'an dernier, nous pensons avoir au moins autant appris qu'en Sup.

Il est à noter également que le site web est désormais hébergé à une adresse fixe et plus adéquate que la précédente : http ://fosmor.fooo.fr

Il rejoint donc le même nom de domaine que le projet de l'an dernier, désormais également accessible via l'adresse : http ://hgf.fooo.fr