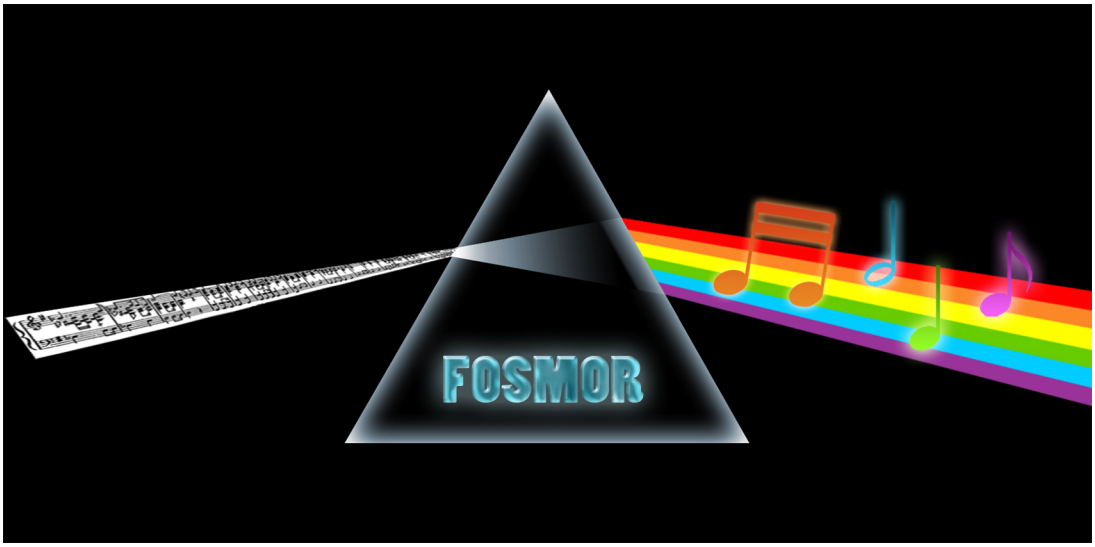
FOSMOR

Fooo Optical Sheet Music Recognition

Rapport de Soutenance

le 13 Février 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

Dans le cahier des charges nous avons défini la nature du projet : un logiciel de reconnaissance de partitions de musique. Nous avions aussi défini la répartition des activités et pour chacune d'elle nous avons effectué des recherches afin de trouver les méthodes usuelles de résolution dans la communauté scientifique.

Dans le présent document nous allons montrer notre avancement dans chaque partie.

L'état de l'art constitué pour le cahier des charges avait exhibé plusieurs méthodologies possibles pour résoudre les problèmes classiques rencontrés dans les applications de reconnaissance de symboles nécessitant un prétraitement.

Nous allons donc préciser les choix théoriques effectués parmi le panel de solutions qui s'offrait à nous.

Nous allons mentionner les difficultés rencontrées lors de la mise en oeuvre de la méthodologie choisie et les solutions que nous avons apportées, ou que nous essayons d'appliquer si le problème n'est pas encore résolu.

Pour montrer le travail accompli, des exemples d'exécution de nos algorithmes seront proposés au jury lors de la soutenance.

Chapitre 2

Prétraitement de l'image

2.1 Chargement de l'image

Malgré la prise en charge n'importe quel format d'image (jpg, png, gif, etc.), il est nécessaire de travailler constamment sur le même format d'image. Nous avons choisi le format *Portable Bit Map* pour sa simplicité. Il est en effet composé d'une courte entête suivie d'une matrice de pixels, évidemment non compressée.

Pour ne pas perdre de temps à gérer les cas particuliers de chaque format standard il est indispensable de faire appel à une bibliothèque de fonctions. Pour son exhaustivité nous avons choisi la bibliothèque *ImageMagic*[1].

Il était tout d'abord question d'utiliser l'interface C de la bibliothèque pour intégrer nativement la conversion d'image dans notre code source. Cependant après de nombreux essais nous avons décidé d'abandonner et d'utiliser plutôt la même bibliothèque déjà installée sur le PIE[2] notamment en raison de nombreux problèmes de quotas sur nos comptes de l'école.

2.2 Gommage du bruit

Afin d'obtenir une image de la meilleure qualité possible, il est nécessaire de la *nettoyer* de ses impuretés, notamment du bruit. Pour cela nous avons dans un premier temps utilisé des matrices de convolutions pour transformer l'image. La technique consiste simplement à multiplier la valeur de chaque pixel de l'image par celles de ses 9 pixels adjacents (le pixel compris), avec des coefficients déterminés par la matrice de convolution.

1. http ://www.imagemagick.org
2. Parc Informatique de l'EPITA

L'application du filtre sur les zones d'impuretés permet de les gommer : ces pixels *impurs* sont en effet beaucoup influencés par leurs voisins eux beaucoup plus proche du blanc. L'application de ces filtres a cependant des effets secondaires qui sont un effet de flou sur les motifs de l'image. L'application de filtres non linéaires sera la prochaine étape et devrait réduire le phénomène de flou.

La matrice suivante donne un effet de flou standard : $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

Cette matrice dite de *flou gaussien* donne un résultat un peu plus satisfaisant : $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

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.

Cependant les méthodes naïves de rotation induisent une détérioration de la qualité de l'image. Il faut donc appliquer différentes méthodes pour que l'image résultante soit la plus fidèle possible à l'originale.

En appliquant une formule de rotation inversée, on peut obtenir l'antécédent de chaque pixel de la nouvelle image. Cet antécédent est cependant une valeur théorique et ne tombe pas forcément sur un pixel précis, mais est comprise entre la valeur de quatre pixels. La méthode naïve consiste à prendre en compte la valeur du pixel le plus proche de ce point théorique. Cette méthode implique de fortes approximations et réduit donc fortement la qualité de l'image.

Une deuxième méthode, dite d'*interpolation bilinéaire* permet de prendre en compte les quatre pixels au lieu d'un seul, pondérés par l'inverse de leur distance à la valeur théorique de l'antécédent du pixel que l'on souhaite calculer. Cette méthode est un peu plus coûteuse en calculs mais qui donne un résultat beaucoup plus lisse.

Par la suite nous implémenterons une troisième méthode, dite *bicubique*, et nous comparerons les résultats pour en juger de l'intérêt face à la différence de complexité des deux méthodes.

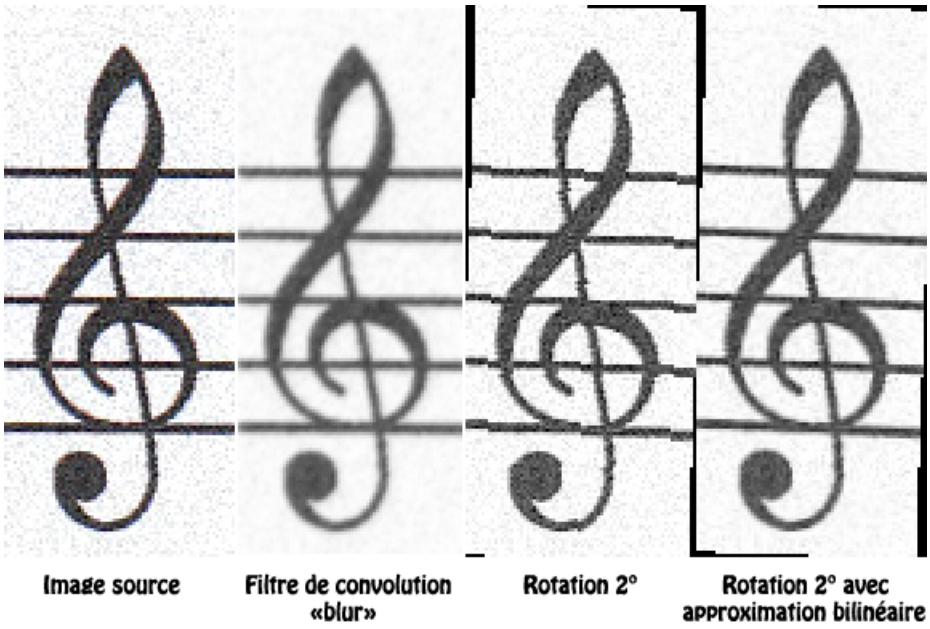


FIGURE 2.1 – Résultats des différents algorithmes sur une image peu bruitée

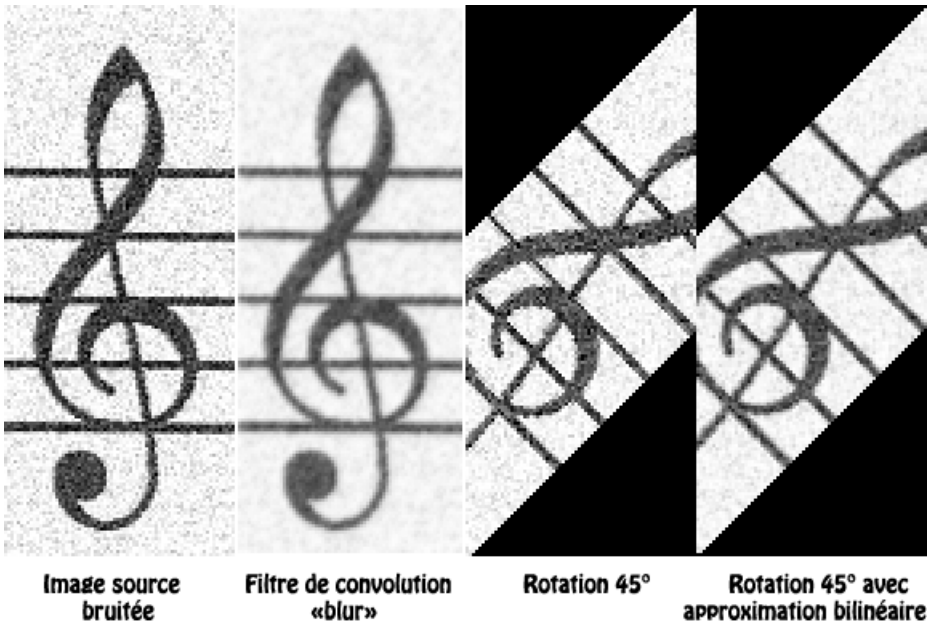


FIGURE 2.2 – Résultats des différents algorithmes sur une image fortement bruitée

Chapitre 3

Détection des lignes de portée

3.1 Introduction

La première étape de l'analyse de la partition musicale est la localisation des lignes de portée. En effet, elles apportent une grande quantité d'information et ont des caractéristiques qui permettent de les repérer avec une relative facilité. Voici leurs caractéristiques fondamentales :

- Elles sont au nombre de 5.
- Elles sont toutes parallèles les unes par rapport aux autres.
- L'espace entre chaque ligne est constant.
- Elles représentent une grande partie de la largeur de la feuille.
- Elles démarrent et s'arrêtent toutes à la même position verticale.

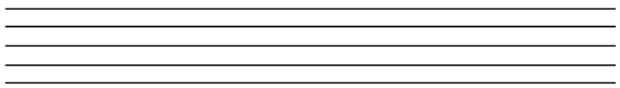


FIGURE 3.1 – Exemple de lignes de portée

3.2 Méthode de détection

Afin de détecter les lignes, nous allons utiliser une méthode déviée du projeté horizontal. D'après les caractéristiques, une ligne est horizontale et prend une grande partie de la largeur de l'image. Grâce à ce constat, on peut compter le nombre de pixels noirs de chaque ligne de l'image et ainsi réaliser un histogramme.

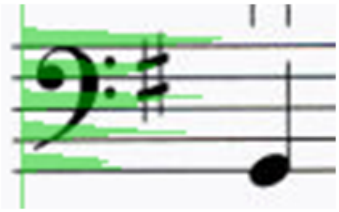


FIGURE 3.2 – Histogramme de densité horizontale de pixels

On voit qu'il y a des pics au dessus de chaque ligne. En effet, l'image n'a pas été parfaitement scannée, ainsi il existe une petite rotation. On observe ainsi des pics relativement larges. Il faut noter que nous pondérons le nombre de pixels noirs avec la taille de la plus grand chaîne continue de pixels noirs. Ainsi, les lignes denses en caractères ont un poids moins fort comparées aux lignes de portée.

Maintenant, il faut réussir à extraire des informations de cette courbe. Nous utilisons un moyen certes simple, mais très efficace : en fixant une valeur seuil, pour l'instant définie empiriquement, nous allons trouver tous les segments de l'intersection de cette droite avec l'histogramme.

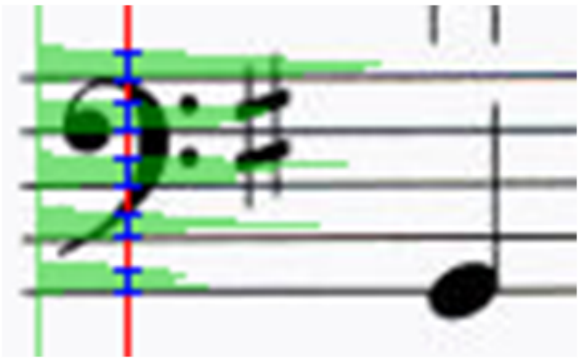


FIGURE 3.3 – Faible inclinaison des lignes

Ainsi, en connaissant la taille h du segment et la largeur w de la partition on arrive à calculer l'angle alpha que fait la ligne avec l'horizontale.

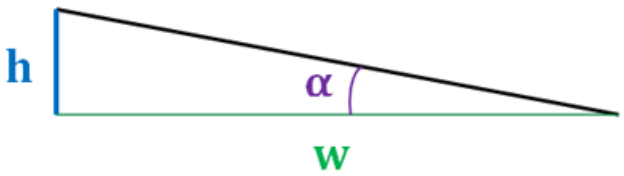


FIGURE 3.4 – Calcul de l'angle

Malheureusement, nous obtenons la valeur absolue de l'angle. En effet, il nous est impossible à ce moment de déterminer si la ligne est penchée vers le bas ou vers le haut. Un moyen simple de comparer les valeurs obtenues grâce au calcul pondéré précédent sur les deux diagonales. On affiche en rouge les lignes ainsi calculées. Le résultat est très proche de celui attendu.

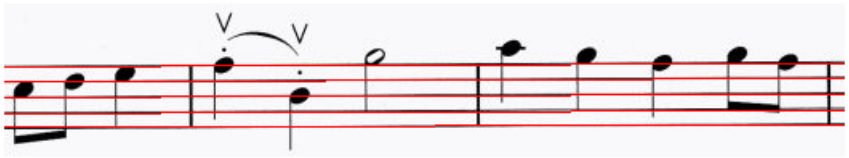


FIGURE 3.5 – Superposition des lignes calculées

Si nous voulions seulement trouver les lignes, notre tâche serait terminée. Nous allons cependant effectuer une rotation de l'image de l'angle trouvé afin de permettre une détection plus simple des verticales ainsi que pour optimiser la reconnaissance des symboles.

Nous procédons à une nouvelle passe de l'algorithme. Les résultats ainsi obtenus sont proches de la perfection sur nos partitions de test. L'amélioration des résultats d'une troisième passe est négligeable et impose une seconde rotation qui est couteuse en calcul et détériore l'image.

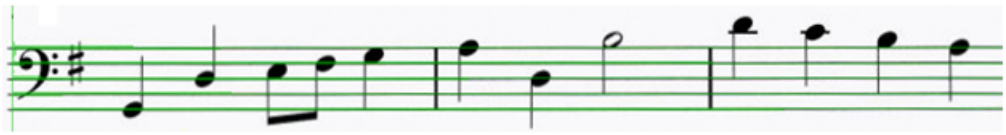


FIGURE 3.6 – Histogramme confondu avec les lignes après rotation

Comme vous pouvez le voir, l'histogramme est confondu avec les lignes et sa largeur ne dépasse pas les 2 pixels, taille de la ligne. Une dernière optimisation qui a été apportée se situe au niveau des marges gauche et droite. En effet, il y a souvent une zone blanche inutilisée qui fait perdre en précision les calculs d'angles. Une simple vérification de la densité en pixel noir de chaque colonne permet de détecter et d'ignorer cette zone.

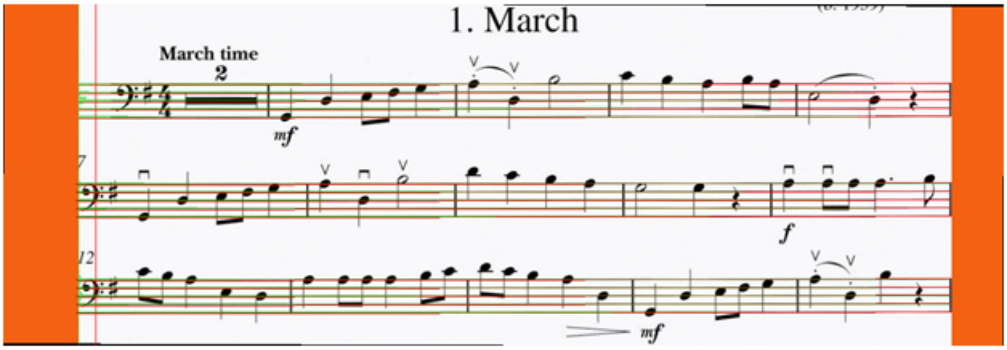


FIGURE 3.7 – Résultat final après rotation et détection des lignes

3.3 Objectifs

Pour faire la détection des lignes, nous avons repris une méthode existante et intuitive tout en l'adaptant pour répondre à nos besoins de la façon la plus simple possible. Cette méthode est efficace pour des lignes droites avec une faible rotation. S'il y a une rotation plus importante, il existe des techniques adaptées que nous prendrons la peine d'implémenter si cela s'avère nécessaire.

Le résultat obtenu avec les diverses optimisations est presque parfait pour les exemples choisis. Les améliorations possibles résident surtout dans la suppression des constantes empiriques utilisées actuellement.

La prochaine étape majeure du développement est la suppression des lignes maintenant trouvées. C'est un travail qui requiert beaucoup de précision. Une mauvaise suppression peut entraîner des fusions de symboles.

Chapitre 4

Perceptron multi-couches

4.1 Choix théoriques

Comme annoncé dans le cahier des charges, nous avons décidé d'utiliser le perceptron multi-couches (PMC) parmi d'autres méthodes comme les réseaux de Kohonen ou de Hopfield.

Le PMC a fait ses preuves dans le domaine scientifique par sa relative facilité de mise en oeuvre et ses bons résultats en tant que classifieur statistique.

Un réseau de neurones est une fonction non linéaire de ses variables et de ses paramètres. Les réseaux de neurones sans neurones cachés ne peuvent discriminer les données non linéairement séparables et donc ne peuvent pas converger en un nombre fini d'itérations. Le XOR est un exemple de fonction non linéairement séparable.

On montre mathématiquement qu'un perceptron simple trouve un hyperplan séparateur en un nombre fini d'itérations si les exemples de l'ensemble d'apprentissage sont linéairement séparables.

Lorsque les exemples ne sont pas linéairement séparables ; on représente la surface discriminante à l'aide de neurones cachés. L'hyperplan ainsi défini par chaque neurone caché doit séparer correctement les exemples des classes différentes au moins dans un voisinage limité de l'hyperplan.

Un PMC, comme son nom l'indique, comporte plusieurs couches d'un ou plusieurs neurones. Chaque neurone est relié à tous les neurones de la couche suivante et cette transition est pondérée selon l'importante de la transition. Nous avons choisi une initialisation aléatoire de ces poids plutôt qu'une initialisation tabula rasa pour vérifier la convergence de l'algorithme dans tous les cas de figure.

Une transition n'est activée que lorsque le potentiel d'activation dépasse un certain seuil (en accord avec le paradigme biologique neuronal dont il est issu). Ceci est modélisé par une fonction d'activation, généralement une fonction à seuil comme la sigmoïde ou la tangente hyperbolique.

Une « couche » d'entrée reçoit l'information sans effectuer de traitement.

Chaque entrée symbolise une caractéristique particulière de l'image selon la méthode de discrimination utilisée : ce peut être la valeur d'un pixel, un moment géométrique d'ordre n, le succès ou non du placement d'un bâtonnet, etc. Au moins une couche cachée qui réalise des traitements intermédiaires par le biais de ses neurones, c'est à dire en réalisant des combinaisons non linéaires de ses entrées.

Enfin une couche de sortie équivalente à un vecteur de flottants, dans notre cas nous utilisons le code 1-parmi-C, c'est à dire qu'a l'événement « l'image en entrée appartient à la classe Ci » est associé le vecteur de la couche de sortie dont seule la composante i est égale à 1. Par exemple si l'on souhaite reconnaître les 10 chiffres on pourra par exemple décider que le 2ème neurone de la couche de sortie devra être à 1 lorsque le chiffre 2 est reconnu.

4.2 Apprentissage supervisé

Les réseaux de neurones non bouclés à apprentissage supervisé sont utilisés pour la classification statique et la discrimination. On dispose d'un ensemble de variables mesurées et d'un ensemble de mesures d'une grandeur relative à un phénomène quelconque. On suppose qu'il existe bel et bien une relation entre les variables et la grandeur que l'on cherche à modéliser et l'on cherche à déterminer une forme mathématique de cette relation valable dans notre domaine d'apprentissage.

Nous ne disposons que de mesures en nombre finis, et ces mesures sont probablement parasitées par du bruit. Il est également possible que toutes les variables qui déterminent notre grandeur à modéliser ne soient pas mesurées.

On cherche à établir un modèle satisfaisant du processus ne se basant que sur les données disponibles. L'apprentissage est supervisé lorsque l'on on présente un motif en entrée à notre réseau et que on le force à converger vers une sortie bien précise. C'est à dire que l'apprentissage n'est pas terminé tant que sa réponse ne se rapproche pas suffisamment d'un seuil que nous avons fixé. Une simple analogie pourrait être de comparer l'apprentissage du réseau de neurones aux expériences vécues par un enfant en bas âge à qui l'ont montrerait une série d'objets verts pour lui faire assimiler la représentation de cette couleur.

4.3 Rétropropagation du gradient

Durant l'apprentissage, puisque les poids sont initialisés aléatoirement, la sortie sera forcément erronée au départ et donc très éloignée du résultat attendu. Le principe de l'algorithme de rétropropagation est de calculer l'erreur commise par le PMC en utilisant l'erreur quadratique moyenne : $\sum_{i=1}^{k}(S[i] - Y[i])^2$

S étant le vecteur attendu de sorties.

Y étant le vecteur de sortie du réseau de neurones après propagation.

Nous cherchons ici à justement minimiser cette fonction de coût des moindres carrés.

L'algorithme consiste à remonter progressivement cette erreur depuis les sorties jusqu'à l'entrée en modifiant les poids au passage.

Soit p la taille de la couche cachée. La sortie i a alors reçu les p sorties des neurones de la couche cachée modifiées par les poids caractéristiques. Ce sont des valeurs mal ajustées de ces poids qui ont induit ces erreurs, il faut donc les ajuster légèrement selon l'incidence qu'il ont eu sur l'erreur en utilisant le gradient.

Prenons un exemple avec un réseau de neurones à 3 entrées, 1 sortie et deux couches cachées.

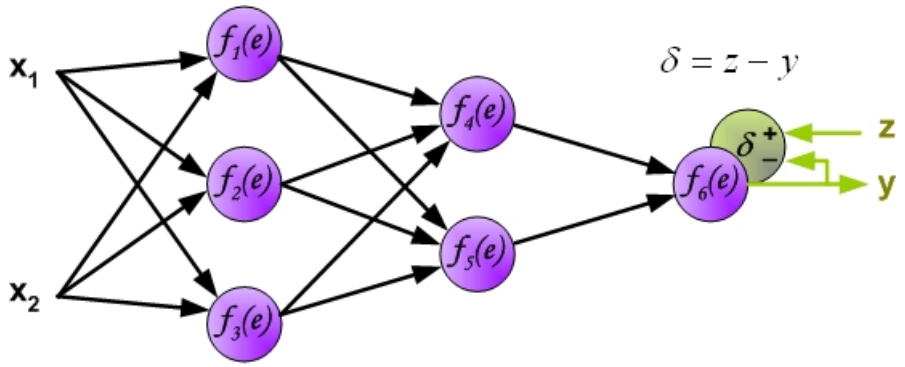


FIGURE 4.1 – Erreur en sortie

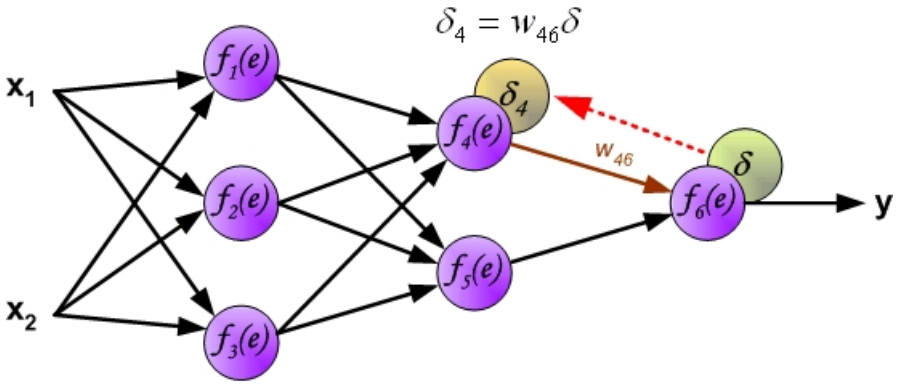


FIGURE 4.2 – Erreur de la deuxième couches cachée sur la sortie

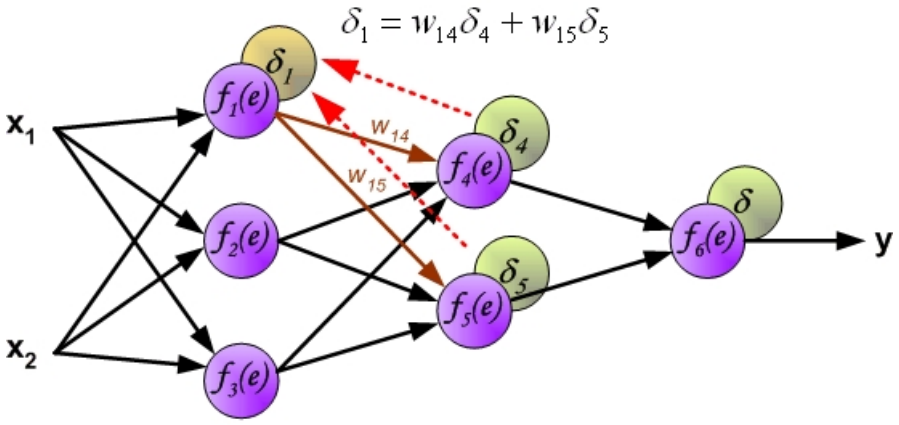


FIGURE 4.3 – Erreur entre première et deuxième couches cachées

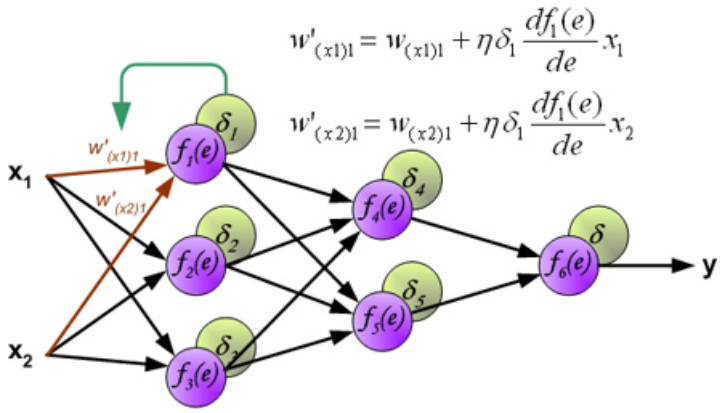


FIGURE 4.4 – Modification des poids entre entrée et première couche cachée

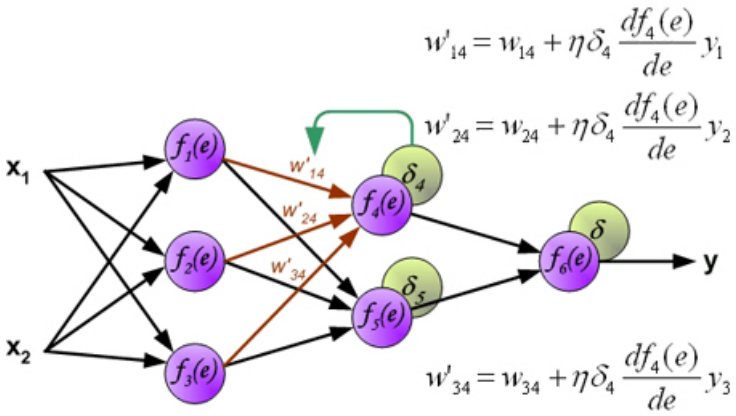


FIGURE 4.5 – Modif. des poids entre première et deuxième couches cachées

4.4 Mise en application

En préparation de cette soutenance nous avons mis en place un perceptron multi-couches à une seule couche cachée avec rétropropagation du gradient comme paradigme d'apprentissage.

Il a été réalisé sous Objective Caml en utilisant justement la couche objet pour en faciliter la manipulation mais aussi pour masquer l'implémentation au reste du programme, il se trouve dans un fichier séparé, se comportant donc à la fois comme un module simple non imbriqué et un objet.

On utilise 2 matrices pour représenter les poids entre les neurones : 1 matrice pour les relations entre entrée et couche cachée, et 1 matrice pour les relations entre couche cachée et sortie.

Nous avons choisi d'utiliser pour le moment la tangente hyperbolique comme fonction d'activation. Les résultats obtenus semblent légèrement meilleurs qu'avec un sigmoïde classique.

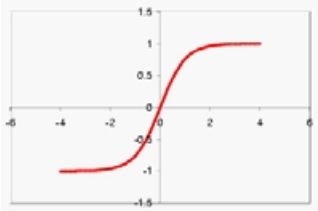


FIGURE 4.6 – La tangente hyperbolique

Les algorithmes de propagation et de rétropropagation sont intrinsèquement impératifs.

Le mode apprentissage prend en entrée toutes les mesures et les sorties désirées en même temps. Cela permet d'apprendre au perceptron tous ces exemples successivement tant que l'erreur moyenne sur chacun des exemples n'est pas satisfaisante (c'est à dire inférieur à un seuil spécifié). Ainsi nous pouvons être sur de minimiser les risques de surapprentissage (puisqu'un réseau de neurones à tendance à se conformer au dernier exemple présenté) comme nous itérons un grand nombre de fois avec tous les exemples de la base d'apprentissage.

Une implémentation objet (tout comme une implémentation modulaire avec type abstrait) permet strictement d'empêcher la modification des variables internes et empêche aussi d'utiliser des fonctions non désirées par utilisation d'une interface de type.

Le réseau de neurones doit se comporter à terme comme une "boite noire" par rapport au reste du programme, il ne doit être possible que de le mettre en mode apprentissage, de lui présenter une entrée, et d'obtenir l'inférence sur sa classe d'appartenance.

Cependant la verbosité a été temporairement accrue dans une optique de démonstration devant le jury en la personne de Christophe Boullay. Par exemple lors de l'apprentissage nous affichons l'erreur moyenne intermédiaire toutes les 100 itérations pour montrer la progression. Enfin nous vérifions les résultats en propageant chaque exemple de la base d'apprentissage et en observant le résultat. Bien sur si l'apprentissage s'est correctement déroulé, le taux de reconnaissance est évidemment de 100%, ce qui est notre cas.

Il y a également une option pour observer les performances selon le nombre de neurones dans la couche cachée. C'est une opération très lourde puisqu'il faut évidemment observer le nombre d'itérations moyenne avant convergence un très grand nombre de fois. Ce nombre d'itérations varie selon les données à traiter (au détriment de la précision) afin d'obtenir un résultat approximatif dans un temps raisonnable.

Pour l'instant, on peut tester l'apprentissage et l'optimisation de la couche cachée sur un XOR, qui est une fonction non linéairement séparable et qui donc ne fonctionnerait pas pour un perceptron simple. Mais on peut aussi faire la même chose sur les chiffres 0..9 en représentation matricielle 16x16 en prenant une entrée par pixel soit 256 entrées.

Pour les chiffres, la convergence demande plus d'itérations et donc plus de temps machine, ce qui rend le test d'optimisation extrêmement couteux sur le PIE, ce qui n'est cependant pas dramatique puisque ce calcul est destiné à être effectué une seule fois.

Pour le XOR, voici le nombre moyen d'itération nécessaire pour faire converger l'algorithme entre 4 et 20 neurones cachés, on remarque qu'un minimum semble atteint pour 15 neurones cachés.

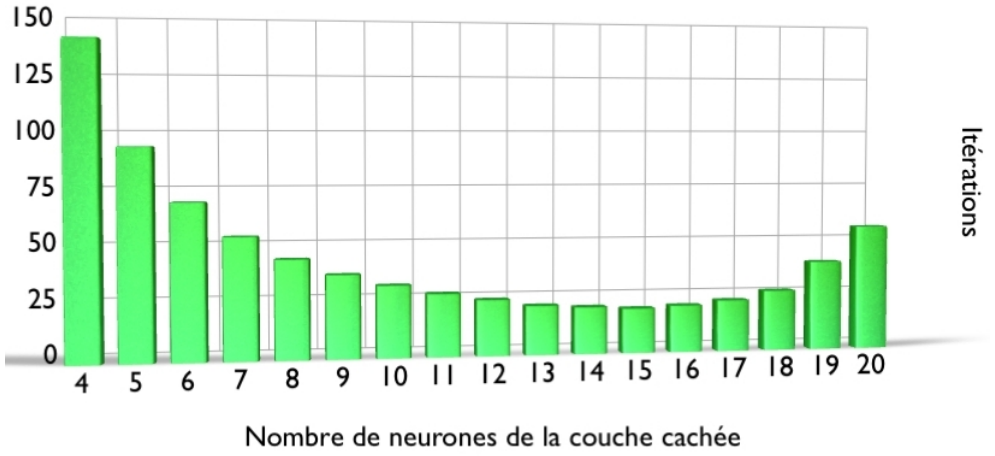


FIGURE 4.7 – Itérations avant convergence

4.5 Objectifs

Pour la prochaine soutenance, nos objectifs sont de tester de nouvelles méthodes d'apprentissage, de tenter de perfectionner l'algorithme et éventuellement de l'optimiser pour le rendre plus rapide.

De nouvelles couches cachées pourraient être ajoutées comme c'est par exemple le cas pour le réseau de neurones LeNet dont la première couche cachée est un "champ réceptif" de 5x5 pixels appelé "carte de caractéristique", toutes les variables d'une même carte sont affectées des même paramètres par la technique des poids partagés. Ainsi on fait agir le même opérateur, localement, sur chaque ensemble de 25 pixels.

Chapitre 5

Caractérisation d'une image

5.1 Problématiques et recherches de solutions

Le problème majeur dans notre cas de reconnaissance de partitions de musique est la multitude de formats pour les symboles que ce soit en hauteur ou en largeur. Or un réseau de neurones n'est qu'un classifieur, la caractérisation est une étape décisive afin d'obtenir une bonne discrimination.

Pour résoudre d'abord le problème du format et pour éviter une normalisation en taille hasardeuse et dangereuse en termes de pertes, nous avons décidé de ne pas normaliser en un seul format mais en plusieurs. Chaque symbole sera donc classé dans une grande catégorie et à chaque catégorie correspond un réseau de neurone différent. Cela ne devrait pas être difficile puisque l'appartenance de chaque symbole à une catégorie devrait pouvoir être inférée sans étude de sa forme.

Concernant les fonctions de caractérisation, nous avions émis plusieurs possibilités lors de la rédaction du cahier des charges :

Représentation matricielle : demande un travail important de normalisation car il est nécessaire de manipuler des matrices de grande dimension. D'autres désavantages comme une occupation mémoire importante donc une lenteur d'apprentissage (n*n entrées). Demande aussi de centrer parfaitement les symboles sinon les résultats sont très chaotiques. Toutefois n'est pas à exclure puisque cela reste la méthode la plus facile.

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. Toutefois les recherches effectuées sur ce sujet se sont heurtées au manque d'explications et à la complexité mathématique sous-jacente. Des essais ont été réalisés mais des problèmes persistent, gageons que le recul apporté par la fin du cours sur les espaces préhilbertiens de Mr Rodot permettra de continuer sur cette voie prometteuse.

Méthode des segments : procédé intéressant algorithmiquement, il consiste à placer des segments sur une image et d'observer ceux qui touchent le symbole. Demande une méta-heuristique longue impliquant par exemple des algorithmes génétiques pour normaliser la longueur des segments, savoir où les placer, etc. Ce n'est pas non plus une méthode à écarter mais dont la possibilité de s'adapter efficacement à notre problème reste encore indéfinie.

5.2 Objectifs

La priorité actuelle est de résoudre les problèmes rencontrés actuellement par le calcul des moments de Zernike puisque cela semble être la méthode la plus prometteuse. L'apport du cours de mathématique pourrait être important pour comprendre les fondements des propriétés de ces moments comme l'invariance. Cumulé à une période de recul et de réflexion au calme après la soutenance, nous espérons repérer et régler les problèmes ou au moins passer à autre chose. Des articles scientifiques concernant l'optimisation des calculs des moments de Zernike ont été trouvés, donc si nous réussissons nous devrions pouvoir rapidement améliorer la rapidité d'exécution.

En cas de problème majeur, ou de non adéquation de cette méthode avec les exigences de notre projet, nous devrions rapidement nous rediriger vers soit une représentation vectorielle soit vers la méthode des segments.

Chapitre 6

Conclusion

Ce projet se révèle être très intéressant car il permet de nous donner un aperçu de ce qu'est la recherche. En effet, les principaux documents de travail que nous avons utilisé sont des rapports de thèse. Nous devons pour réaliser notre objectif faire une synthèse des différentes méthodes existantes.

De plus, la dimension visuelle du projet rend sa conception pratique et agréable à la fois. Quoi de mieux que de voir que les lignes calculées et réelles se superposent !

Enfin, cela nous donne la possibilité d'appliquer tout ce que l'on a appris durant cette année et demi de prépa aussi bien au niveau algorithmique qu'au niveau du langage utilisé qu'est le Caml.

FIGURE 6.1 – Fooo Team