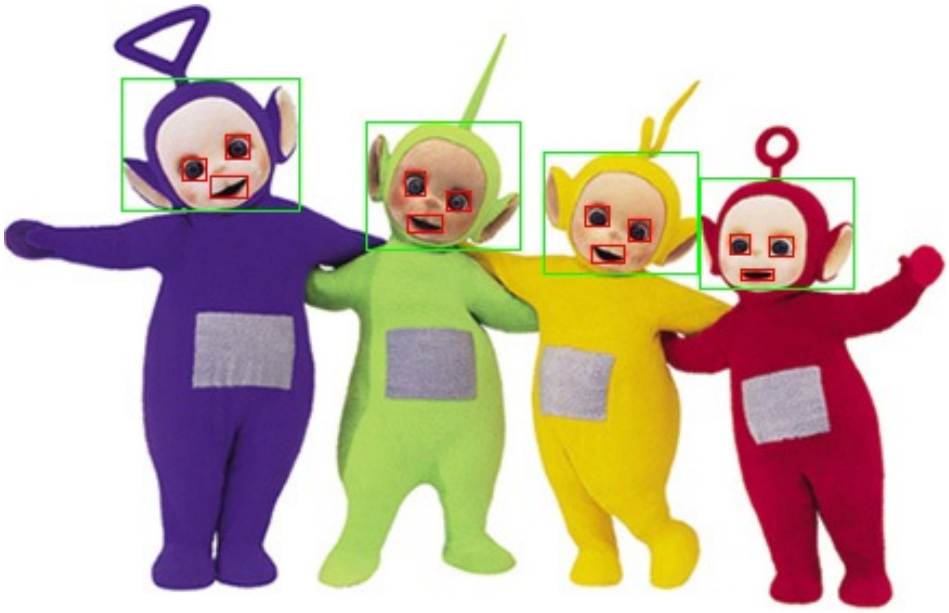

Rapport de projet « MediOCR »

Logiciel de reconnaissance optique de caractères

The Tubbies

Epita

Antoine « Pluggi » Bardoux — bardou_a
Thibaut « tbarroyer » Barroyer — barroy_t
Pierre « BLUESCREEN » Boulay — boulay_p
Valentin « B-add » Lamatte — lamatt_v

Projet de Spé 2014 – 2015

Sommaire

1 Introduction

Le logiciel de reconnaissance optique de caractères, autrement appelé OCR, représente le coeur de la partie informatique pratique de ce premier semestre de notre seconde année à l'EPITA. C'est donc à quatre que nous nous sommes lancés dans ce projet et avons donné le meilleur de nous même pour mener à bien cet objectif. Notre projet a beau s'appeler mediOCR, nous nous sommes impliqués et appliqués au maximum pour qu'il ne le soit pas. Nous sommes donc les Tubbies, en référence aux héros de notre enfance, et ce rapport que vous tenez entre vos mains en ce moment même constitue la pièce finale de l'édifice qu'a représenté ce projet pour nous.

2 Présentation des membres

2.1 Valentin « B-add » Lamatte

Mon arrivée à l'EPITA, en début d'année dernière, a constitué pour moi un grand saut dans le monde de l'informatique. C'était un monde qui m'avais toujours perdu quelque part entre l'excitation et la peur de tout ce que l'on pouvait faire avec. C'est pourquoi j'avais choisi d'intégrer cette école qui proposait à ses élèves une incursion dans l'informatique en passant par les bases de celle ci. Désormais familiarisé avec les bases et la théorie liés à la programmation, je m'apprête à me lancer dans cette nouvelle aventure. C'est donc fort de ma première année d'expérience en programmation que je me lançais dans le projet de cette année, l'OCR. Ce n'est cette fois plus l'accessible C# que l'on utilisera, mais bien du C.

2.2 Thibaut « tbarroyer » Barroyer

Ayant commencé le langage C à l'âge de 15 par simple curiosité, je me suis vite rendu compte que je souhaitais en faire mon métier. J'ai donc commencé à étudier plusieurs langages de programmation, sans jamais rien faire de poussé. Je suis donc rentré à l'EPITA l'an dernier en espérant de tout mon coeur pouvoir faire des choses plus intéressantes. C'est en première année que j'ai découvert la joie des projet de longue durée en équipe grâce au jeu vidéo que j'ai du réaliser l'an dernier. Ce premier projet m'a fait découvrir le jeu vidéo de manière amusante, mais j'avais vraiment hâte de faire quelque chose de vraiment complexe. C'est pourquoi j'ai décidé de m'occuper du réseau de neurones durant ce projet. J'ai alors vite découvert les joies des "segfaults" ainsi que des "core dumped". Ce fut un véritable défi de réaliser ce réseau de neurones, vu que je ne m'y connaissais pas du tout en intelligence artificiel, et je suis très content du travail que j'ai réalisé.

2.3 Antoine « Pluggi » Bardoux

J'avais commencé à coder il y a un petit moment, aux alentours de la 3e, en Python. Ce langage m'a vraiment passionné parce que grâce à lui je pouvais réaliser ce que je voulais, du site web à l'émulateur Chip-8 en passant par le scripting dans le but de faire de l'édition de vidéo. Sa simplicité au premier abord a été pour moi d'un réel intérêt car je n'aurais sûrement pas eu la volonté de commencer en apprenant le C, qui demande trop de connaissances dès le début. Cependant, maintenant que j'ai vu

différents langages tels que le Python, l'OCaml, le C#, etc. je suis bien plus à même de l'utiliser avec efficacité. Ce projet aura donc été pour moi l'occasion, en plus de travailler sur des algorithmes complexes, de me familiariser avec un langage qui à première vue ne m'attirait pas trop.

2.4 Pierre « Bluescreen » Boulay

Passionné par l'informatique dès mon plus jeune âge, j'ai commencé la programmation à l'âge de 14 ans en attaquant directement les choses sérieuses, c'est-à-dire le C. Le C est un langage que je connais assez bien et que j'aime utiliser. C'est donc un plaisir de réaliser ce projet dans ce langage. J'aime beaucoup la programmation en langage natif et pour cette raison, je compte m'investir corps et âme pour que ce projet soit un succès.

J'aime beaucoup travailler en équipe, chaque membre apporte ses connaissances et son expérience au service du projet. On apprend les uns des autres et des liens se créent. C'est pour moi la meilleure méthode pour apprendre, d'une part à programmer, et de l'autre à travailler en équipe, gérer son temps et répartir les tâches.

3 La compilation et le débogage

Pierre Boulay

Avant de commencer à travailler sur le code, il faut d'abord définir comment le programme sera compilé. Nous avons fait le choix d'utiliser GCC, car la plupart d'entre nous le connaissent mieux que Clang. Une fois le compilateur choisi, il faut créer le makefile. Pour être efficaces, nous avons choisi de créer deux cibles :

— release :, la cible par défaut, compile avec –O3 (optimisation maximale) et définit la constante « BUILD_RELEASE ».

— debug : compile avec l'option –g, ce qui facilite l'utilisation d'un débogueur, et définit la constante « BUILD_DEBUG ».

Ce système permet de ne pas avoir à choisir entre les performances et la facilité de débogage. Une fois le makefile mis en place, nous avons choisis d'utiliser un header principal. Ce fichier s'appelle « ocrdefs.h ». Ce fichier est inclus par tous les autres fichiers du projet. Il définit par exemple les types tels que « Bin_Image »(une image binarisée) ou encore « Pixel », les fonctions usuelles telles que min, max, abs, et certaines constantes (comme des couleurs par défaut, des tailles de buffer, etc.) en fonction de la cible.

En effet, à l'aide du langage préprocesseur, on peut savoir sous quelle cible on est en train de compiler. Ce qui permet par exemple de contrôler les fonctions inlines. On définit ce type de constante comme montré ci dessous.

```
#ifdef BUILD_RELEASE
  #define OCRAPI inline
#else
  #define OCRAPI
#endif
```

Lorsque l'on définit une fonction et que l'on marque avec le mot clé « inline », le compilateur va, plutôt que de faire un réel appel sur cette fonction, recopier le code de la fonction à l'endroit où elle est appelée, ce qui permet d'améliorer les performances, mais rend le débogage plus compliqué. En faisant précéder une fonction de la constante « OCRAPI », celle-ci sera définie « inline » seulement en mode « release ». Il faut néanmoins noter qu'il vaut mieux appliquer ce mot clé seulement aux fonctions de petites tailles. De plus, ce système de macro peut servir à rediriger les écritures de la sortie standard vers un fichier (par défaut logs.txt).

Pour le débogage à proprement parler, nous utilisons GDB. Mais il n'est pas suffisant. En effet, le C est un langage natif, dans lequel l'accès à la mémoire est direct. La plupart des erreurs des programmes écrits en C proviennent d'une mauvaise manipulation de pointeurs. Le problème, c'est que lorsque l'on commet une erreur et que l'on utilise un pointeur qui est invalide, par exemple si on ne l'a pas initialisé, sa valeur est indéfinie, c'est-à-dire qu'il peut pointer vers n'importe quelle adresse. Par chance, il peut prendre la valeur de 0, et lorsqu'on va essayer de l'utiliser, le programme va immédiatement s'arrêter, l'erreur est donc bien visible. Mais il est possible que, par malchance, l'adresse vers laquelle pointe le pointeur appartienne bien au processus. Dans ce cas, le programme va lire ou écrire ailleurs que là où il croit le faire, dans la plupart des cas, le programme plantera plus loin. Lorsque ce moment arrive, il est très difficile de remonter à la source du problème.

Pour réduire au maximum ce genre de scénario, nous avons pris la décision d'utiliser Valgrind, qui, en plus de détecter la plupart des erreurs de segmentation, permet aussi de détecter les fuites de mémoires.

4 Les bibliothèques

Pour cette dernière soutenance, nous utilisons deux bibliothèques externes : libharu et GTK. La première nous permet de créer des PDF contenant le texte reconnu par notre réseau de neurone. Cette bibliothèque est cependant rendue optionnelle si le projet est compilé en utilisant `make nopdf`. En effet, certains utilisateurs pourraient ne pas avoir envie d'installer cette dépendance et elle n'est pas forcément nécessaire à l'utilisation de notre programme car l'on peut également exporter en format texte. On a donc également pris soin d'afficher une fenêtre d'avertissement dans l'interface graphique pour indiquer à l'utilisateur que le projet a été compilé sans ce support.

Nous utilisons donc également la bibliothèque GTK. Celle-ci sert à créer une interface graphique digne de ce nom afin de présenter toutes les fonctionnalités du langage sous la forme d'un programme simple à utiliser pour un novice.

5 Les structures de données

Pierre Boulay

Ce projet gère beaucoup de types de données différents. En effet, on peut citer les images (en couleurs, en nuances de gris ou binarisées), les rectangles, les pixels, les structures du réseau de neurones. La plupart de ces structures sont définies dans le header principal, `ocrdefs.h`. La structure représentant un pixel est définit de cette façon :

```
typedef struct
{
    BYTE red;
    BYTE green;
    BYTE blue;
} Pixel;
```

Pour ce qui est d'une image, on la définit de la façon suivante :

```
typedef struct
{
    USHORT w;//La largeur de l'image
    USHORT h;//La hauteur de l'image
    Pixel *data;
}Image ;
```

Pour une image en niveau de gris, on remplacera seulement le membre « data »par une pointeur sur un tableau de BYTE (niveau de gris compris entre 0 et 255). Pour ce qui est de l'image binarisée, on utilisera un tableau de booléens. Il faut noter que l'on pourrait utiliser des opérations bit à bit pour stocker 8 pixels par booléen (en effet, un booléen pèse en réalité 8 bits), mais ce système impliquerait qu'à chaque fois que l'on veut accéder à un pixel, on devrait effectuer ses opérations bit à bit, ce qui demanderait plus de travail au processeur. Pour cette raison, nous avons fait le choix de stocker un seul pixel par booléen, ce qui prend plus de mémoire, mais qui est plus rapide au niveau lecture/écriture.

Logiquement, une image est un tableau de pixels à deux dimensions. Mais, lorsque l'on utilise un système d'allocation dynamique de mémoire, un tableau à deux dimensions

est en réalité un tableau de pointeurs pointant eux-mêmes sur des tableaux(les données ne sont pas contiguë), ce qui signifie que l'on doit suivre deux pointeurs pour accéder à un élément (contre un seul dans un tableau à une dimension). Pour stocker les pixels de l'image, on va donc allouer un tableau à une dimension de taille w*h*(taille d'un pixel) et utiliser la formule suivante : $place = y * w + x$ pour lire ou écrire un pixel dans ce tableau.

6 Le pré-traitement

6.1 Niveau de gris

Antoine Bardoux

Afin de réaliser la binarisation nécessaire à une bonne détection des caractères, il est fondamental de passer l'image en niveaux de gris. Le principe pour cette transformation est très simple. Il suffit d'additionner les différentes composantes du pixel et de les diviser par 3 afin de rester sur un nombre compris entre 0 et 255. L'accès aux composantes du pixel se fait grâce à des fonctions que nous avons écrites nous-mêmes car cette fonction prend en paramètre une `struct Image`. Nous n'utilisons pas de *magic numbers* comme on peut en trouver sur internet car ils sont là avant tout pour améliorer la perception par un œil humain.

L'algorithme est donc très simple. Il suffit d'itérer sur chaque pixel de l'image et d'appliquer la fonction suivante :

```
Grey(R, G, B) = (R + G + B) / 3
```

Cependant, étant donné que nous enregistrons ces nouveaux pixels dans un tableau uni-dimensionnel de type `unsigned char`, il nous a fallu faire attention à le transformer en `int` proprement. Nous utilisons nos propres structures car l'accès à une `SDL_Surface` est très lent et cette structure comporte énormément de données dont nous ne voulons pas. En effet, après un passage en niveau de gris, nous n'avons pas besoin des trois composantes RGB présentes dans une `SDL_Surface` car elles seront égales.

FIGURE 1 – Version en nuance de gris à droite

6.2 Binarisation

Antoine Bardoux

La binarisation est une étape incontournable avant la détection des lignes et caractères. En effet, sans elle, l'image est remplie de différentes nuances de gris et les algorithmes seraient bien plus complexes sans cette binarisation. Nous avons donc choisi d'implémenter la méthode d'Otsu qui détermine quel est le meilleur seuil pour binariser l'image. En plus de celle-ci, grâce à notre interface en ligne de commande, il est possible de faire une binarisation avec un seuil fixé par l'utilisateur.

6.2.1 Binarisation rapide

La binarisation rapide est donc une fonction avec un seuil fixe. Nous avons décidé d'ajouter ce choix car c'est une binarisation extrêmement rapide qui peut donc être utilisée sur des images de très grande taille. De plus, elle est utilisée dans notre implémentation de l'algorithme d'Otsu.

Le principe est assez simple, on parcours l'image et si le niveau de gris du pixel

est supérieur à un seuil `s`, alors le pixel est noir, sinon il est blanc :

$$Bin(grey) = \begin{cases} \texttt{0x00} & \text{si } grey > s \\ \texttt{0xFF} & \text{sinon} \end{cases} \quad (1)$$

e whisky se trouve sans doute à un tournant de son histoire. On assiste en effet depuis quelques années à de nombreux bouleversements : le rachat en juillet 2005 du N°2 mondial des vins et spiritueux Allied Domecq par le français Pernod Ricard (ex N°3) qui en contrepartie a du céder sa distillerie irlandaise Bushmills au N°1 mondial Diageo, sans oublier le rachat inattendu fin 2004 de Glenmorangie Plc par le groupe LVMH, leader mondial des produits de luxe. Ces cessions et acquisitions qui portent sur plusieurs dizaines de milliards d'euros ont animé les marchés financiers et mis en émoi le microcosme des professionnels du whisky.

Alors que ces différents acteurs imposent leurs marques sur la scène internationale, on assiste dans le même temps au rachat, par des négociants visionnaires et des investisseurs audacieux, de distilleries plus confidentielles. D'autres ont récemment vu le jour ou sont en passe d'être créées. Or jamais au cours de ces 50 dernières années autant de changements n'avaient été observés. Serait-ce le début d'un nouvel âge d'or pour l'industrie du whisky ? Seule une réglementation plus contraignante et le politiquement correct en matière de consommation d'alcool semblent pouvoir inverser cette tendance.

Ainsi comme vous le découvrirez au fil de ces pages, nous ne nous contentons pas de commercialiser des marques existantes. Nous sélectionnons et créons nos propres gammes de whiskies. Nous communiquons notre passion à travers notre site internet www.whisky.fr, Nous publions des ouvrages et sommes également à l'initiative de Whisky Magazine, seule revue en français consacrée au whisky. Nous animons un club de dégustation, organisons des voyages dans les différents pays producteurs et réunissons amateurs et distilleries à l'occasion de notre festival annuel, le Whisky Live Paris. Nous nous intéressons également depuis quelques années à d'autres univers olfactifs et gustatifs.

A leurs débuts, les plus grandes maisons d'assemblages, Chivas ou Johnnie Walker (dont on fête cette année le 200ème anniversaire de la naissance), étaient également des négociants en thé, café, épices ... Notre intérêt pour ces produits de nez et de bouche, est une sorte de retour aux sources. Mais c'est surtout une démarche personnelle, fruit de la rencontre avec de grands professionnels comme Maître Tseng, spécialiste du thé mondialement connue ou Pierre Hermé, pâtissier génial et maître chocolatier inspiré.

Mais vous vous demandez peut-être comment tout cela a commencé ? Quelle était la place du whisky en France à nos débuts et quels ont été les grands changements de ces 50 dernières années ? Alors...

Thierry Bénitah
Directeur Général

FIGURE 2 – Binarisation avec seuil de 127

6.2.2 Méthode d'Otsu

La méthode d'Otsu sert à déterminer la meilleure valeur de seuil pour un pixel. Le calcul est effectué en supposant que l'image ne contient qu'un arrière plan et un premier plan, ce qui est parfait dans notre cas, car les documents donnés à un OCR sont généralement des feuilles scannées. L'arrière plan est donc le fond blanc et le premier plan est le texte. Le principe de l'algorithme est donc de calculer le seuil optimal entre ces deux plans en réduisant le plus possible la variance entre les deux classes (arrière plan et premier plan).

Nous avons implémenté deux algorithmes légèrement différents dans leur implémentation mais revenant au même. Dans un cas, on cherche le seuil qui minimise la variance intra-classe avec la formule suivante :

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \tag{2}$$

Alors que dans l'autre nous calculons le seuil pour lequel la variance inter-classe est maximale :

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_1(t)\omega_2(t)\Big[\mu_1(t) - \mu_2(t)\Big]^2 \tag{3}$$

On voit bien ici que l'algorithme d'Otsu est bien meilleur car avec le seuil fixe nous avions des caractères très dégradés alors qu'ici ils restent suffisamment épais.

6.2.3 Méthode de Sauvola

Même si l'algorithme d'Otsu nous donne de bons résultats, parfois la qualité du document est tout de même dégradée. Ainsi, j'ai recherché un meilleur algorithme de binarisation qui nous permettrait de perdre le moins de données possible tout en étant très performant. J'ai donc fini par trouver un article à propos de la méthode de Sauvola, qui est une dérivée de celle de Niblack. Ces deux méthodes ont été spécialement conçues pour la binarisation de texte. Sauvola considère le document qui lui est donné comme étant une collection de sous-composants, ces sous-composants étant soit du texte soit le fond du document. C'est donc sans surprise que les résultats sont bien meilleurs que ceux d'Otsu.

$$T_{sauvola} = m \times \left(1 - k \times \left(1 - \frac{s}{R}\right)\right) \tag{4}$$

Le whisky se trouve sans doute à un tournant de son histoire. On assiste en effet depuis quelques années à de nombreux bouleversements : le rachat en juillet 2005 du N°2 mondial des vins et spiritueux Allied Domecq par le français Pernod Ricard (ex N°3) qui en contrepartie a du céder sa distillerie irlandaise Bushmills au N°1 mondial Diageo, sans oublier le rachat inattendu fin 2004 de Glenmorangie Plc par le groupe LVMH, leader mondial des produits de luxe. Ces cessions et acquisitions qui portent sur plusieurs dizaines de milliards d'euros ont animé les marchés financiers et mis en émoi le microcosme des professionnels du whisky.

Alors que ces différents acteurs imposent leurs marques sur la scène internationale, on assiste dans le même temps au rachat, par des négociants visionnaires et des investisseurs audacieux, de distilleries plus confidentielles. D'autres ont récemment vu le jour ou sont en passe d'être créées. Or jamais au cours de ces 50 dernières années autant de changements n'avaient été observés. Serait-ce le début d'un nouvel âge d'or pour l'industrie du whisky ? Seule une réglementation plus contraignante et le politiquement correct en matière de consommation d'alcool semblent pouvoir inverser cette tendance.

Ainsi comme vous le découvrirez au fil de ces pages, nous ne nous contentons pas de commercialiser des marques existantes. Nous sélectionnons et créons nos propres gammes de whiskies. Nous communiquons notre passion à travers notre site internet www.whisky.fr, Nous publions des ouvrages et sommes également à l'initiative de Whisky Magazine, seule revue en français consacrée au whisky. Nous animons un club de dégustation, organisons des voyages dans les différents pays producteurs et réunissons amateurs et distilleries à l'occasion de notre festival annuel, le Whisky Live Paris. Nous nous intéressons également depuis quelques années à d'autres univers olfactifs et gustatifs.

A leurs débuts, les plus grandes maisons d'assemblages, Chivas ou Johnnie Walker (dont on fête cette année le 200ème anniversaire de la naissance), étaient également des négociants en thé, café, épices ... Notre intérêt pour ces produits de nez et de bouche, est une sorte de retour aux sources. Mais c'est surtout une démarche personnelle, fruit de la rencontre avec de grands professionnels comme Maître Tseng, spécialiste du thé mondialement connue ou Pierre Hermé, pâtissier génial et maître chocolatier inspiré.

Mais vous vous demandez peut-être comment tout cela a commencé ? Quelle était la place du whisky en France à nos débuts et quels ont été les grands changements de ces 50 dernières années ? Alors...

Il était une fois ...

Thierry Bénitah
Directeur Général

FIGURE 3 – Binarisation par la méthode d'Otsu

Sur la même image que précédemment, l'application de l'algorithme de Sauvola nous donne ceci :

On peut encore une fois voir une très nette amélioration par rapport aux autres méthodes.

Le whisky se trouve sans doute à un tournant de son histoire. On assiste en effet depuis quelques années à de nombreux bouleversements : le rachat en juillet 2005 du N°2 mondial des vins et spiritueux Allied Domecq par le français Pernod Ricard (ex N°3) qui en contrepartie a du céder sa distillerie irlandaise Bushmills au N°1 mondial Diageo, sans oublier le rachat inattendu fin 2004 de Glenmorangie Plc par le groupe LVMH, leader mondial des produits de luxe. Ces cessions et acquisitions qui portent sur plusieurs dizaines de milliards d'euros ont animé les marchés financiers et mis en émoi le microcosme des professionnels du whisky.

Alors que ces différents acteurs imposent leurs marques sur la scène internationale, on assiste dans le même temps au rachat, par des négociants visionnaires et des investisseurs audacieux, de distilleries plus confidentielles. D'autres ont récemment vu le jour ou sont en passe d'être créées. Or jamais au cours de ces 50 dernières années autant de changements n'avaient été observés. Serait-ce le début d'un nouvel âge d'or pour l'industrie du whisky ? Seule une réglementation plus contraignante et le politiquement correct en matière de consommation d'alcool semblent pouvoir inverser cette tendance.

Toutes ces évolutions que nous observons avec intérêt sont finalement très éloignées de ce qui nous anime au sein de La Maison du Whisky : la sélection et la commercialisation de whiskies de qualité. Cette qualité est au cœur même du débat qui oppose les anciens et les modernes. On entend dire régulièrement que les whiskies étaient meilleurs dans les années 1960-70, que les blends à cette époque avaient du caractère. Or malgré notre expérience, nous n'avons pas d'avis tranché sur le sujet. Ce que nous pouvons en revanche affirmer, c'est que les whiskies d'antan étaient élaborés avec les technologies et les contraintes de l'époque afin de produire la meilleure eau-de-vie possible.

A l'aube du 50ème anniversaire de La Maison du Whisky, nous avions envie de vous faire partager notre vision du marché du whisky et de vous dévoiler nos différents métiers. Nous souhaitions également vous décrire notre quotidien et vous confier nos incertitudes qui nous poussent à nous remettre en question et à nous dépasser.

Ainsi comme vous le découvrirez au fil de ces pages, nous ne nous contentons pas de commercialiser des marques existantes. Nous sélectionnons et créons nos propres gammes de whiskies. Nous communiquons notre passion à travers notre site internet www.whisky.fr, Nous publions des ouvrages et sommes également à l'initiative de Whisky Magazine, seule revue en français consacrée au whisky. Nous animons un club de dégustation, organisons des voyages dans les différents pays producteurs et réunissons amateurs et distilleries à l'occasion de notre festival annuel, le Whisky Live Paris. Nous nous intéressons également depuis quelques années à d'autres univers olfactifs et gustatifs.

A leurs débuts, les plus grandes maisons d'assemblages, Chivas ou Johnnie Walker (dont on fête cette année le 200ème anniversaire de la naissance), étaient également des négociants en thé, café, épices ... Notre intérêt pour ces produits de nez et de bouche, est une sorte de retour aux sources. Mais c'est surtout une démarche personnelle, fruit de la rencontre avec de grands professionnels comme Maître Tseng, spécialiste du thé mondialement connue ou Pierre Hermé, pâtissier génial et maître chocolatier inspiré.

Mais vous vous demandez peut-être comment tout cela a commencé ? Quelle était la place du whisky en France à nos débuts et quels ont été les grands changements de ces 50 dernières années ? Alors...

Il était une fois...

Thierry Bénitah
Directeur Général

FIGURE 4 – Binarisation par la méthode de Sauvola

6.3 Filtre médian

Antoine Bardoux

Pour cette deuxième soutenance, j'ai également implémenté une réduction du bruit en utilisant l'algorithme du filtre médian. Le filtre médian est bien adapté à notre usage

car il permet de réduire le bruit tout en conservant les contours de l'image. Ainsi, la forme des caractères est très bien conservée.

Le principe de l'algorithme est de remplacer chaque pixel par la valeur médiane des pixels de son voisinage. Pour notre projet nous avons décidé d'utiliser une matrice `3x3` afin de ne pas trop flouter l'image. Ainsi, la sélection de la valeur médiane ne se fait que dans son voisinage direct.

Prenons l'image suivante :

124	126	127
120	150	125
115	119	123

Il faut tout d'abord trier ces valeurs, afin d'avoir la valeur médiane au milieu du tableau. Les valeurs voisines sont donc 115, 119, 120, 123, 124, 125, 126, 127 et 150. On peut donc en déduire facilement que la valeur médiane est de 124.

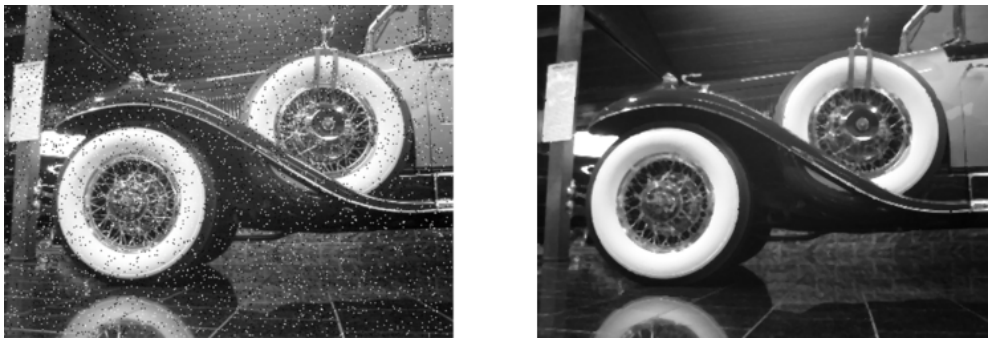

FIGURE 5 – Exemple de transformation par l'usage d'un filtre médian

6.4 La rotation

Pierre Boulay

Pour appliquer une rotation à une image en fonction d'un angle et d'un point d'origine de rotation, nous avons utilisé les formules suivantes (avec α l'angle, x et y les coordonnées du point dans l'image source, px et py celles du centre de rotation et w et h respectivement la largeur et la hauteur de l'image) :

— $fx(x, y) = cos(\alpha) \times (x - px) - sin(\alpha) \times (y - py) + px$

— $fy(x, y) = sin(\alpha) \times (x - px) + cos(\alpha) \times (y - py) + py$

Néanmoins, si l'on applique simplement cette formule, on obtient un résultat similaire à l'image ci-dessous :

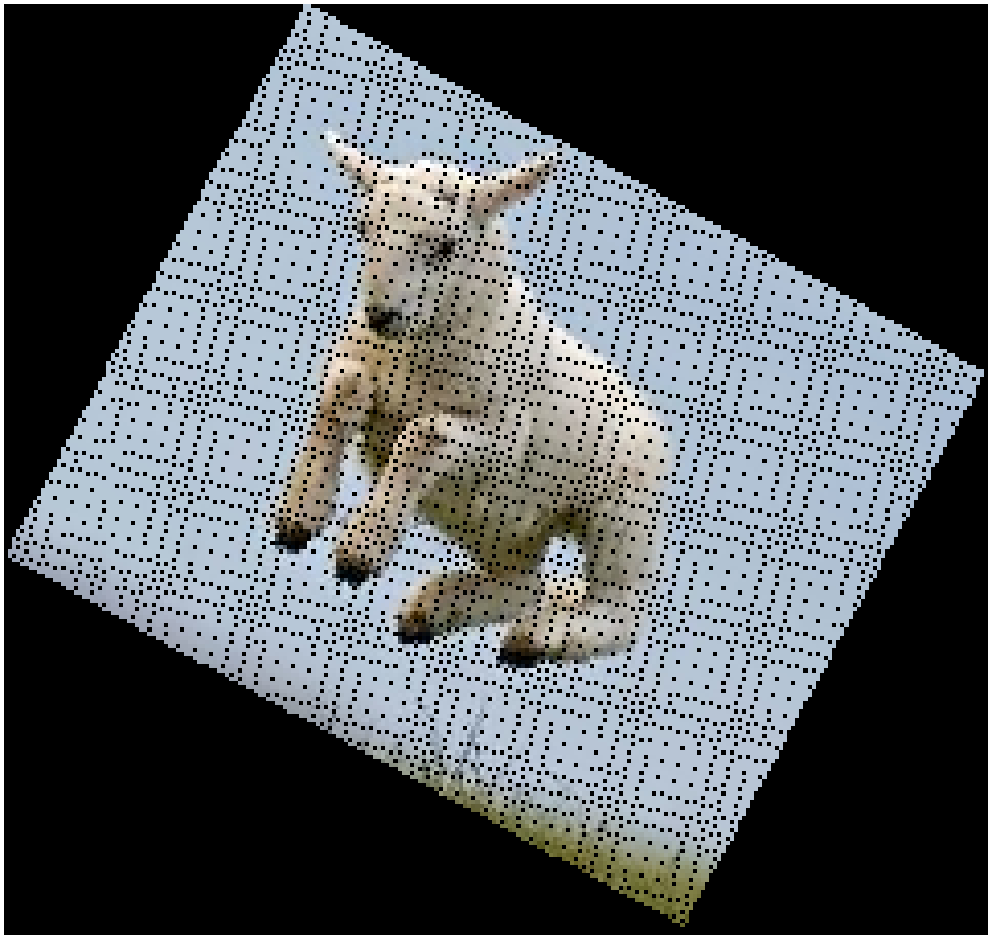

FIGURE 6 – Résultat de l'application naïve de la formule

On observe que certains pixels sont noirs. Ce problème provient du fait qu'à cause des arrondis, pour le pixel p, $|fx(x, y) - fx(x + 1, y)|$ peut être strictement supérieur à 1. Ce qui signifie que lors du remplissage d'une ligne, un des pixels de cette ligne peut, à cause des arrondis, rester à sa valoir d'origine. Si les trous que l'on a observés plus haut sont noirs, c'est parce que la mémoire a été mise à zéro avant d'être utilisée.

Pour résoudre ce problème, il va falloir compter les écarts en x et en y avec le pixel précédant, si l'on constate que l'un de ces écarts est supérieur ou égal à 2, on va arrondir les coordonnées du pixel de l'image de sortie et en fonction de cet arrondi puis combler le trou avec le pixel actuel ou le pixel précédent.

Mais ce n'est pas le seul problème posé par la rotation, en effet, lorsque l'on applique un angle différent de pi et de 0, une certaine partie de l'image va se retrouver dans une zone avec des coordonnées négatives (on considère que le point en haut à gauche de l'image est l'origine du repère). Mathématiquement, ça ne pose pas de problème, par contre, lorsque les pixels sont représentés par un tableau, il n'est pas possible d'avoir des indexes négatifs. Pour cette raison il faut donc calculer l'abscisse du point le plus à gauche ainsi que l'ordonnée du point le plus en haut. Pour ce faire, on va calculer appliquer la formule aux quatre coins de l'image et conserver la valeur d'abscisse minimale ainsi que la valeur d'ordonnée minimale. On va ensuite ajouter ces coefficients de décalage à tous les pixels avant de les écrire sur l'image de sortie.

Notons tout de même que la fonction de rotation produit des nombres qui ne sont pas forcement entiers, et que, à cause des arrondis, deux pixels différents peuvent se retrouver au même endroit sur l'image de sortie. La rotation d'une image avec cet algorithme peut donc provoquer une légère perte de données. Néanmoins, cette faible perte de données, et, à moins de d'avoir une image avec une très faible résolution, ne devrait pas empêcher le réseau de neurones de reconnaitre les lettres.

7 La détection des caractères

Pierre Boulay

7.1 La détection des blocs de texte

Une fois que l'on dispose d'une image binarisée, on peut commencer la détection de blocs de texte, néanmoins, la détection des blocs de texte est une partie complexe, il faut en effet prévoir le cas où le texte est écrit sur plusieurs colonnes. Pour résoudre ce problème, nous avons appliqué un algorithme rlsa (run length smoothing algorithm) sur l'image binarisée. Cet algorithme a pour but d'extraire des rectangles à partir d'un nuage de point (ici l'image binarisée). Pour ce faire, on parcourt l'image dans n'importe quel sens, lorsque l'on tombe sur un pixel égal à 0, on regarde à quel distance est le pixel à 1 le plus proche, si cette distance est inférieure à une constante, on fait passer la valeur du pixel à 1.Cette solution peut sembler parfaite, mais le problème reste la définition de cette constante, surtout que plus on l'augmente, plus la complexité de l'algorithme augmente, l'image ci-dessous illustre le passage de cet algorithme sur une image binarisée avec plusieurs colonnes.

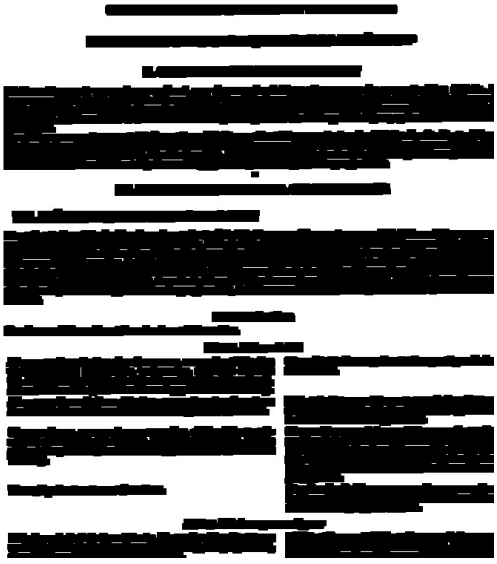

FIGURE 7 – Image générée par un algorithme rlsa

7.2 La détection des lignes

Une fois que l'on a détecté les blocs de texte d'un document, on va pouvoir en extraire les lignes. On ne peut pas simplement parcourir le bloc de haut en bas et couper

dès que l'on tombe sur une ligne entièrement blanche. En effet, si on applique naïvement cette méthode, certaines polices peuvent provoquer des erreurs. Par exemple, si la barre descendante du « p » est assez basse et que la barre montante du « d » est assez haute, deux lignes différentes risquent d'être considérées comme une seule, ce qui posera des problèmes plus loin.

Pour éviter cette erreur, il existe une solution assez simple. Pour commencer, on se place en haut du bloc, et on va descendre jusqu'à arriver sur une ligne qui contient au moins un pixel noir. Une fois que c'est fait, on va chercher à séparer cette ligne de la suivante (ou éventuellement atteindre la fin du bloc). On va continuer à descendre jusqu'à tomber sur une ligne qui remplit au moins un des deux critères suivants :

- La ligne est totalement blanche (ce qui arrive dans la majorité des cas)
- La ligne n'est pas totalement blanche, dans ce cas on la parcourt de gauche à droite (ou de droite à gauche, cela n'a pas d'importance) en appliquant l'algorithme suivant sur chaque point noir :
  - On va parcourir la ligne comme un graphe non orienté, en considérant qu'il existe une arrête entre tous points noirs contiguë.
  - Pendant ce parcours, on va ajuster un rectangle pour qu'il contienne tous les sommets de ce graphe.
  - Une fois le parcours du graphe terminé, on regarde si le point le plus haut du graphe est plus haut que le point sur lequel on a commencé le parcours. Si oui, cela signifie que l'on est sur une lettre descendante (comme un « p »), ce n'est donc pas la fin de la ligne. Dans le cas contraire, on est sur une lettre montante (comme un « l »), on eut donc considérer que l'on a atteint la fin de la ligne.

On note que pour optimiser les performances, les parcours de graphes sont, dans leur implémentation, des parcours largeurs. Cela permet, de plus, d'éviter les erreurs de type « stack overflow » lorsque l'on parcourt des graphes de grandes tailles.

7.3 L'extraction des lettres

7.3.1 Le parcours de graphe

Après avoir détectés les blocs puis les lignes, la partie la plus intéressante commence : la détection des caractères. On va donc analyser chaque ligne extraite précédemment en stockant les lettres extraites dans des structures du type suivant :

```
typedef struct
{
    Rect rect;//Le rectangle dans lequel se trouve la lettre
    Pixel color;//La couleur de la lettre
    Bin_image *data;//Une image binarisee representant la lettre
                Rect bloc;//Le bloc de texte contenant la lettre
                char value;//La valeur de lettre
}Letter;
```

On va donc parcourir chaque ligne, mais avant de commencer la parcours, on va copier le rectangle représentant la ligne à partir de l'image binarisée dans une image binarisée temporaire. Ensuite, on va démarrer le parcours, on commence en haut à droite et on parcourt les colonnes de haut en bas, puis les lignes de gauche à droite.

Lorsque l'on arrive sur un pixel noir, on va, comme pour les lignes, parcourir l'image binarisée comme un graphe non orienté en enregistrant les points par lesquels on est passés. Une fois le parcours terminé, on va créer une image binarisée qui sera l'image de la lettre que l'on passera au réseau de neurones plus tard. On va ensuite rendre blanc tous les pixels noirs que l'on a enregistrés lorsque l'on a fait le parcours de graphe.

Ce système propose plusieurs avantages lorsque l'on le compare à un simple parcours de la ligne de gauche à droite en regardant si les colonnes sont totalement vides.

— Il fonctionne sans problème lorsque le texte est souligné ou en italique
— Il ne risque pas de traiter deux lettres comme une seule

7.3.2 Les points sur les "i"

La méthode décrite plus haut n'est pas parfaite. En effet, un problème majeur subsiste : les lettres dont tous les pixels ne sont pas contigus (comme les lettres à accents, les « i », etc.) comme deux entités différentes. Pour pallier à ce problème, on va effectuer un traitement après la reconnaissance des lettres.

On va parcourir les lettres de chaque ligne en regardant les positions en x des lettres. Si on remarque qu'une lettre est incluse dans une autre sur l'axe x, on va la combiner avec la lettre dans laquelle est-elle incluse. Par exemple, pour le « i ». La bas et le autre de la lettre seront d'abord reconnus séparément, puis, le post-traitement va

remarquer que ces deux lettres ont des positions en x qui sont liées et va transformer ces deux lettres en une seule.

7.3.3 Les espaces

C'est un problème qui ne saute pas aux yeux, mais dont il faut tout de même s'occuper. En effet, il n'est pas aisé de repérer les espaces dans un document. Dans un premier temps, on peut penser à faire une moyenne des espaces entre les lettres et considérer que lorsqu'un de ces espaces est supérieur à la moyenne, c'est un espace entre deux mots. Cette méthode ne peut pas fonctionner dans tous les cas, par exemple quand un document contient des blocs de textes de polices de tailles différentes, la moyenne est faussée.

Une fois que toutes les lettres ont été détectées, on dispose d'une pile contenant les rectangles des lettres détectées. On va donc pouvoir créer une image illustrant la détection en dessinant des rectangles de couleurs aux positions définies dans cette pile. L'image ci-dessous illustre ce processus.

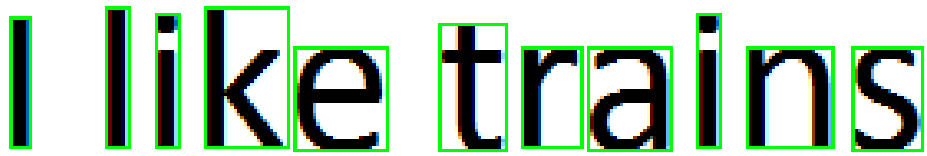

FIGURE 8 – Résultat de la détection de caractères

7.4 La détection des couleurs

Une fois que l'on dispose du rectangle contenant la lettre, il est facile de détecter sa couleur. En se basant sur l'image binarisée, on va faire la moyenne des couleurs des pixels contenus dans le rectangle (on calcule donc cette moyenne à partir des pixels qui sont à 1 dans l'image binarisée). On peut ensuite dessiner le rectangle autour de la lettre de la même couleur que celle ci comme l'illustre l'image ci dessous.

On va donc appliquer cet algorithme en pondérant la moyenne par bloc. On va donc calculer une moyenne pour chaque bloc, ensuite, on va, en fonction du nombre de lettres

FIGURE 9 – Résultat de la détection de caractères

par ligne, pondérer la moyenne. Cela permet d'éviter qu'une ligne ne contenant qu'un mot fausse la moyenne. Par la suite, on va définir tous les espaces qui sont supérieurs à la moyenne comme des espaces entre deux mots.

Dans la structure Letter, un espace est défini lorsque les champs « value » vaut : ' '. En ce qui concerne les espaces entre les blocs, il est assez facile de les détecter, lorsqu'une lettre est un espace entre deux blocs, le champ 'value' prend la valeur '
n'.

8 Le calcul parallèle

Les algorithmes utilisés dans MediOCR sont assez complexes et peuvent prendre un temps élevé pour retourner leur résultat. Pour diminuer le temps que prennent les calculs, nous avons fait appel à un système de calcul parallèle (multithreading). Ce système utilise la bibliothèque pthread.

Nous avons utilisé la fonction « timediff » que nous avions créée dans debugapi.c pour mesurer quelles parties de l'extraction de texte prenait le plus de temps et nous nous sommes rendu compte que la détection de bloc prenait une majeure partie du temps de calcul.

Comme expliqué plus haut, la détection de blocs se base sur un système de r.l.s.a. CE système implique que l'on voit devoir lire tous les voisins de chaque pixel à une distance n. Cette distance « n » est généralement comprise entre 5 et 15. On imagine bien qu'un tel algorithme est très couteux lorsque l'on souhaite l'appliquer à une grande image. Nous avons décidé d'utiliser pthread pour minimiser le temps de retour de cette fonction.

L'utilisation de plusieurs threads dans un programme peut poser des problèmes de concurrence. Il faut s'assurer qu'il n'est pas possible que la mémoire lue par un thread ne

puisse être écrite par un autre simultanément. Il faut également vérifier que deux threads ne peuvent pas écrire au même endroit au même moment. Il est possible d'obtenir des « verrous » sur des ressources pour bien vérifier que seul un thread y a accès, mais l'obtention d'un « verrou » est couteuse. Si l'on devait verrouiller chaque pixel avant d'y avoir accès, cela aurait un effet désastreux sur les performances.

Nous avons donc réfléchi à une solution fonctionnant sans « verrou » mais stable. Nous avons donc mis au point l'algorithme suivant :

Sur une image de w*h pixels, on démarre n threads. Chaque thread comme à la ligne i (0<=i<n), une fois la ligne traitée, il passe à la ligne n+i. Ce système de calcul parallèle fonctionne donc aucun système de verrouillage, ce qui lui permet de s'exécuter rapidement.

9 Réseau de neurones

Thibaut Barroyer, Valentin Lamatte

9.1 Introduction

La détection des caractères est une partie très importante d'un OCR, mais une fois ces caractères détectés il nous faut l'analyser pour pouvoir savoir de quels caractères il s'agit. Et ce tout en tenant compte des éventuelles imperfections de l'image et de la police d'écriture. Pour ce faire, nous auront donc recours à un réseau de neurones.

9.2 Qu'est ce qu'un réseau de neurones ?

Un réseau de neurones est donc un modèle de calcul inspiré du fonctionnement du cerveau humain. Comme ce dernier, ils peuvent avoir la capacité d'apprendre. Un réseau de neurones est donc muni d'entrées et d'une sortie. Ainsi il est assimilable à une fonction mathématique. Un réseau de neurones est donc composé, comme son nom l'indique, de neurones, ces derniers étant constitués de la manière suivante :

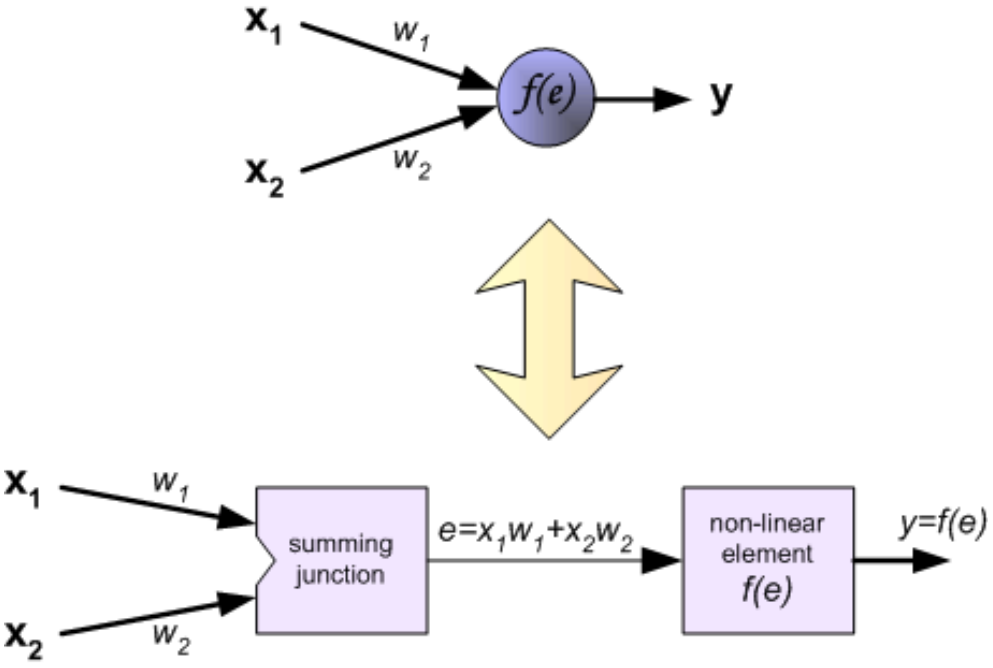

FIGURE 10 – Un neurone

Ces neurones reçoivent un nombre X d'entrées, ces entrées, représentant souvent la sortie d'un neurone précédent, sont multipliées par un coefficient propre à chaque synapse liant chaque neurone. Une somme est ensuite faite à partir de ces entrées. Cette somme sera par la suite traitée par une fonction inhérente au neurone. Cette fonction, dans un réseau de neurone, peut être de tout type mais représente souvent un seuil. Suivant que la somme des entrées est supérieure au seuil ou inférieure à celui-ci, le résultat en sortie sera différent.

Une fois compris ce qu'était un neurone et comment il marchait, il nous a fallu étudier la façon dont ils allaient communiquer entre eux, et donc étudier l'architecture d'un réseau neuronal. Après quelques recherches, nous avons vite remarqué qu'un réseau de neurones était un modèle très structuré. Mais cette structure peut être généralisée, car un réseau de neurones n'est au final qu'un enchaînement fini de couches de neurones. C'est donc ainsi que tous les neurones de la première couche vont prendre en entrée toutes les entrées du réseau dans son ensemble, puis les sorties de chacun de ces neurones vont de nouveau être en entrée de chacun des neurones de la couche. Et ce pour chaque couche du réseau jusqu'à la sortie.

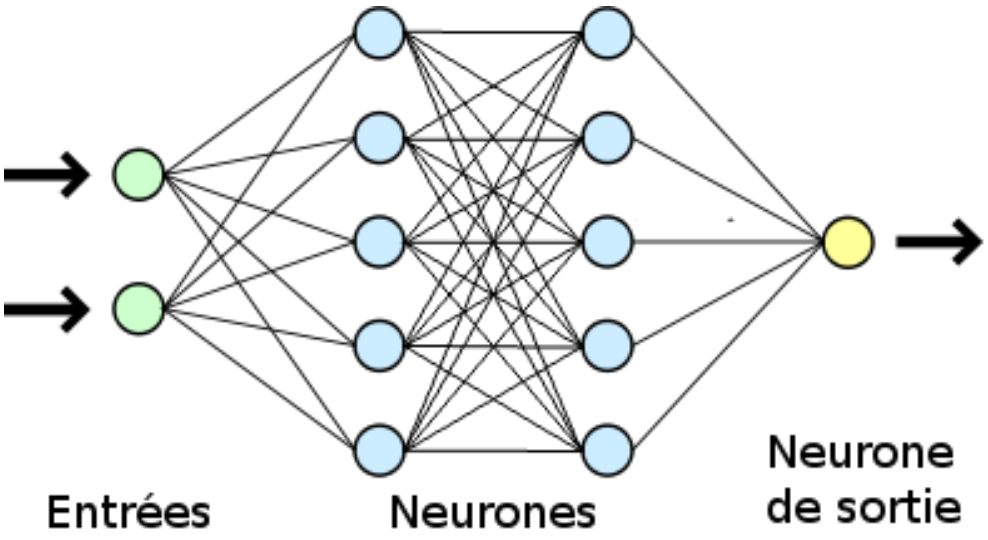

FIGURE 11 – Exemple de réseau de neurones utilisant des couches

9.3 Sigmoïde

Pour le besoin de notre réseau de neurones, nous avons choisi non pas de faire que nos neurones marchent avec une fonction à seuil, dans laquelle on compare uniquement la somme des entrées à une valeur de seuil renvoyant 1 si cette somme est supérieure au seuil, 0 sinon, mais d'utiliser une fonction sigmoïde. Cette dernière va prendre la somme, en entrée, et y appliquer la fonction sigmoïde qui va renvoyer une valeur entre 0 et 1 permettant un échelonnement par rapport au modèle a seuil. Cette fonction est la suivante : $f(x) = \frac{1}{1+\exp(-x)}$ Celle ci va nous permettre de mettre en place l'apprentissage des caractères.

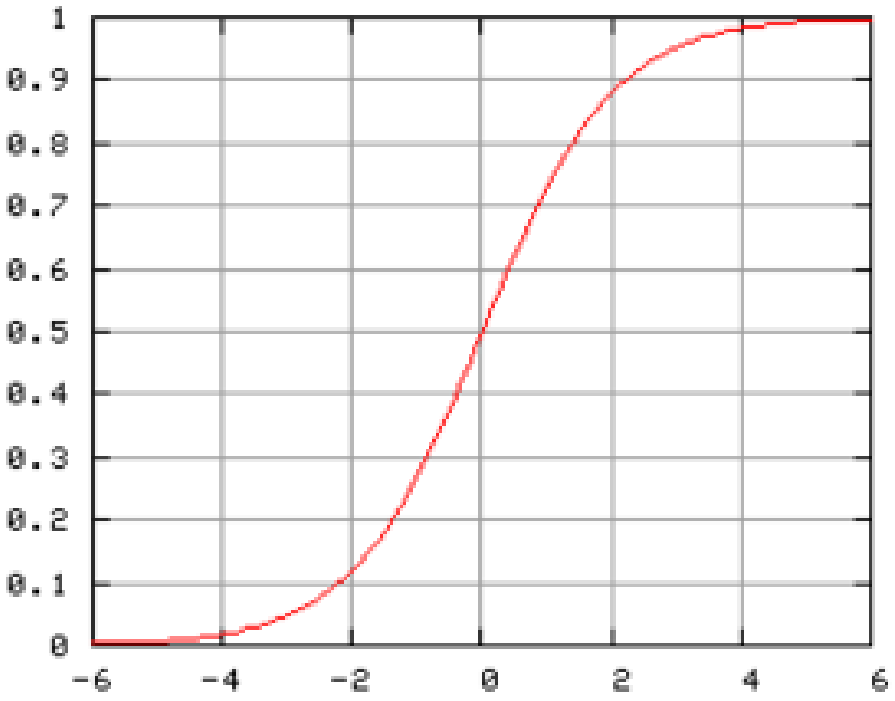

FIGURE 12 – Allure d'une fonction sigmoïde

9.4 Apprentissage

Comme dit plus haut, l'intérêt majeur d'un réseau de neurones réside dans sa capacité à apprendre. Ainsi nous avons dû, pour réaliser cet apprentissage, nous renseigner sur ce dernier. Ainsi, lorsque l'on passera nos entrées au réseau, il lui faudra mettre à jour chacun des coefficients sur les synapses en fonction d'une erreur calculée en soustrayant le résultat obtenu au résultat escompté. Ainsi, si l'on lance l'algorithme d'apprentissage un certain nombre de fois, on diminuera l'erreur. Mais il faut faire attention à ne pas trop faire tourner cet algorithme, car on risque d'obtenir une détection des caractères trop spécifique aux caractères de base et on ne pourra donc pas gérer les éventuelles imperfections.

9.5 Implétentation

9.5.1 Première implémentation

La première implémentation que nous avions choisi était celle-ci :

```
struct network
{
        int nblayer; //Nombre de couches du reseau
        **layer layer; //Tableau de couches de neurones
        **synapse synapse; //Tableau de couches de synapses
};

struct layer
{
        int nbneurone; //Nombre de neurones dans la couche
        *double neurone; //Tableau de valeurs de chaque neurone
                         //de la couche
};

struct synapse
{
        int nbsynapse; //Nombre de synapses dans la couche
        *double synapse; //Tableau de valeurs de chaque synapse
                //de la couche
};
```

Le problème de cette implémentation est qu'elle est trop compliquée à utiliser. Elle nécessite l'utilisation de beaucoup de buffers dans les différentes fonctions de traitement et d'apprentissage. Il est par exemple "lourd" d'écrire "n->layer[n->nblayer - 1]->neurone[i]" afin d'accéder aux neurones de la dernière couche. Le code était trop compliqué à déboguer à cause de cette implémentation trop simple, nous avons donc décidé de passer sur une structure plus complète, rendant notre code plus simple à comprendre et donc plus simple à déboguer.

9.5.2 Deuxième implémentation

```
struct network
{
    double *synapse;
    //tableau de valeurs de toutes les synapses du reseau

    double *neurone;
    //tableau de valeurs de tous les neurones du reseau
    double *neurone_in;
    //pointeur sur le premier neurone de la couche d'entree
    //du tableau de neurones
    double *neurone_out;
    //pointeur sur le premier neurone de la couche de sortie
    //du tableau de neurones

    double *err_neurone;
    //tableau stockant les valeurs d'erreur de chaque neurones
    //du reseau
    double *err_neurone_out;
    //pointeur sur la premiere valeur d'erreur de la couche de
    //sortie du tableau err_neurone

    int *i_layer;
    int *i_synapse;
    //tableau d'index des premiers neurones/synapses de chaque
    //couches dans le tableau de neurones et de synapses
```

```
    int nb_layer;
    //nombre de couches
    int nb_neurone;
    //nombre de neurones dans le reseau
    int nb_synapse;
    //nombre de synapses dans le reseau

    int *size_layer;
    //tableau des tailles de chaque couche
    int size_layer_in;
    //taille de la couche d'entree
    int size_layer_out;
    //taille de la couche de sortie
};
```

Cette nouvelle implémentation a permis de simplifier notre code au maximum et ainsi d'y voir beaucoup plus clair. Pour revenir à notre exemple précédent, pour accéder aux neurones de la couche de sortie, on fait "n->neuroneout[i]". Nous pouvons donc à présent de déboguer plus facilement nos fonctions.

9.5.3 Paramètres de notre réseau pour l'apprentissage de caractère

Après avoir implémenté (et déboguer) les différentes fonctions liées au réseau de neurone, il faut maintenant paramétrer celui-ci afin qu'il effectue un apprentissage correct des différents caractères. Nous avons choisi d'opter pour un réseau de neurones prennant en entrée 16*16 pixels (-1 pour un pixel blanc et 1 pour un pixel noir) et sortant un tableau de 26 flottant, avec dans chaque case la ressemblance au caractère correspondant à la case (entre -1 et 1). Par exemple si l'on obtient 0.7 dans la permière case du tableau et -0.2 dans la deuxième, cela veut dire que le 'a' à plus de chance d'être le bon caractère que le b car sa valeur dans le tableau est plus élevé. Après un nombre considérable de tests, nous avons décidé de créer un réseau comportant une couche cachée de 200 neurones. Les poids des synapse sont fixés aléatoirement grâce cette formule :
$n->syn[i] = 0.5\frac{(double)rand()}{RANDMAX} - \frac{0.5}{2}$
$\frac{(double)rand()}{RANDMAX}$ donnant un flottant aléatoire compris entre 0 et 1, et 0.5 étant donc notre coefficient de génération des synapses.

Nous avons fixé notre apprentissage à 1000 applications de l'algorithme de rétro-propagation du gradient par caractère de chaque police (14), avec un coefficient d'apprentissage fixé à 0.02.
Cela nous a permis d'obtenir un réseau de neurone reconnaissant les caractères minuscules :

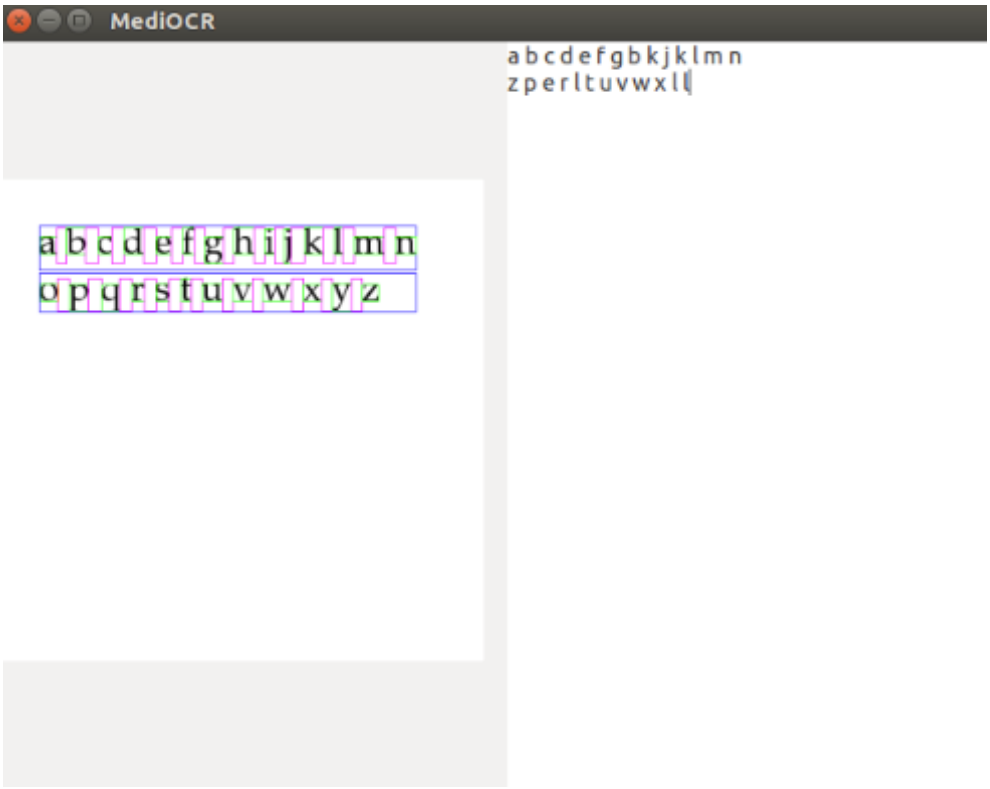

FIGURE 13 – Teste du réseau sur l'alphabet latin

10 Dictionnaire

Pierre Boulay

Une fois le traitement de l'image effectué, et le texte récuperé grâce au réseau de neurones, nous avions la possibilité de garder uniquement ce texte brut en sortie. Mais notre réseau n'étant pas parfait, certaines lettres demeuraient différentes de ce qui était attendu.

Nous avons donc décidé d'utiliser un dictionnaire qui traiterait un par un les mots en sortie du réseau de neurones. De cette manière, si une lettre diffère du mot d'origine, en calculant la différence entre le mot renvoyé par le réseau et les mots présent dans le dictionnaire, on pourra trouver le mot le plus proche de celui ci et le remplacer, permettant ainsi de supprimer l'insertitude inhérente au traitement du réseau de neurones.

Pour ce faire nous avons choisi d'utiliser la distance de Levenshtein. Celle ci est une distace mathématique donnant une mesure de la similarité entre deux chaînes de caractères. Elle est égale au nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne à l'autre.Elle est aussi connue sous le nom de distance d'édition. Ainsi en calculant la distance de Levenshtein entre notre mot en sortie du réseau et les mots du dictionnaire de la langue souhaité (française dans notre cas), on remplace le mot en question par celui du dictionnaire ayant la distance d'édition la plus faible avec lui.

```
entier DistanceDeLevenshtein(caractere
        chaine1[1..longueurChaine1],
    caractere chaine2[1..longueurChaine2])
  // d est un tableau de longueurChaine1+1 rangees et
  longueurChaine2+1 colonnes
  declarer entier d[0..longueurChaine1, 0..longueurChaine2]
  // i et j iterent sur chaine1 et chaine2
  declarer entier i, j, cout

  pour i de 0 a longueurChaine1
      d[i, 0] := i
  pour j de 0 a longueurChaine2
      d[0, j] := j

  pour i de 1 a longueurChaine1
      pour j de 1 a longueurChaine2
          si chaine1[i] = chaine2[j] alors cout := 0
                                     sinon cout := 1
          d[i, j] := minimum(
                                     d[i-1, j   ] + 1,      // effacement
                                     d[i,   j-1] + 1,      // insertion
                                     d[i-1, j-1] + cout    // substitution
                                   )

  renvoyer d[longueurChaine1, longueurChaine2]
```

11 L'exportation du texte

Pierre Boulay

Pour permettre à l'utilisateur d'exporter son texte, MediOCR propose deux types d'exportation :

— L'enregistrement dans un fichier .txt, on enregistre simplement les caractères les uns après les autres en mettant des retours à la ligne à la fin de chaque paragraphe.
— La sauvegarde dans un fichier pdf. Nous allons nous intéresser à cette méthode en particulier.

Pour écrire dans un fichier pdf, MediOCR utilise la bibliothèque « LibHaru ». Cette bibliothèque n'est pas très connue et n'est pas disponible dans les répertoires de paquets Linux. Nous avons pris la décision d'utiliser LibHaru plutôt que LibPdf pour sa simplicité d'utilisation. Si quelqu'un veut simplement exécuter MediOCR (sans le compiler), il n'a pas besoin d'installer LibHaru. Si l'on choisit de compiler en résolution statique des liens, MediOCR peut s'exécuter sans avoir à charger LibHaru depuis un fichier extérieur au programme.

L'exportation n'est tout de même pas simple comme bonjour, même si l'on peut écrire des lettres simplement dans le pdf, il faut tout de même gérer les fins de lignes, les paragraphes, les pages, les titres, etc.

Pour gérer les lignes, on va garder la position en ordonnées dans une variable que l'on va incrémenter de l'interligne à chaque nouvelle ligne. De même, on va garder la colonne en mémoire dans une variable. Et on va l'incrémenter à chaque lettre de la taille de cette lettre. Par la suite, lorsqu'une ajoute une ligne, on vérifie que l'on ne va pas dépasser la taille verticale de la page, si on va la dépasser, on ajoute une nouvelle page et on écrit la nouvelle ligne dessus. On a juste à forcer le retour à la ligne quand on change de bloc de texte. On demande bien sur confirmation à l'utilisateur avant de réécrire sur un fichier existant déjà auparavant.

12 Interface Graphique

Valentin Lamatte

Partie souvent négligée mais néanmoins vitale de tout bon logiciel de reconnaissance optique de caractère, l'interface graphique (souvent abrégée «GUI» de l'anglais Graphical User Interface) représente donc une partie importante de notre projet.

12.1 Pourquoi une GUI ?

L'intérêt majeur de l'utilisation d'une interface graphique lors d'un projet réside dans le fait de permettre à l'utilisateur d'utiliser son logiciel de manière intuitive. L'utilisateur d'un programme possédant une GUI pourra donc utiliser son programme de manière optimale et ce, uniquement grâce a sa souris et une fenêtre. C'est donc dans une optique de développement pour un large publique que l'on utilise une interface graphique.

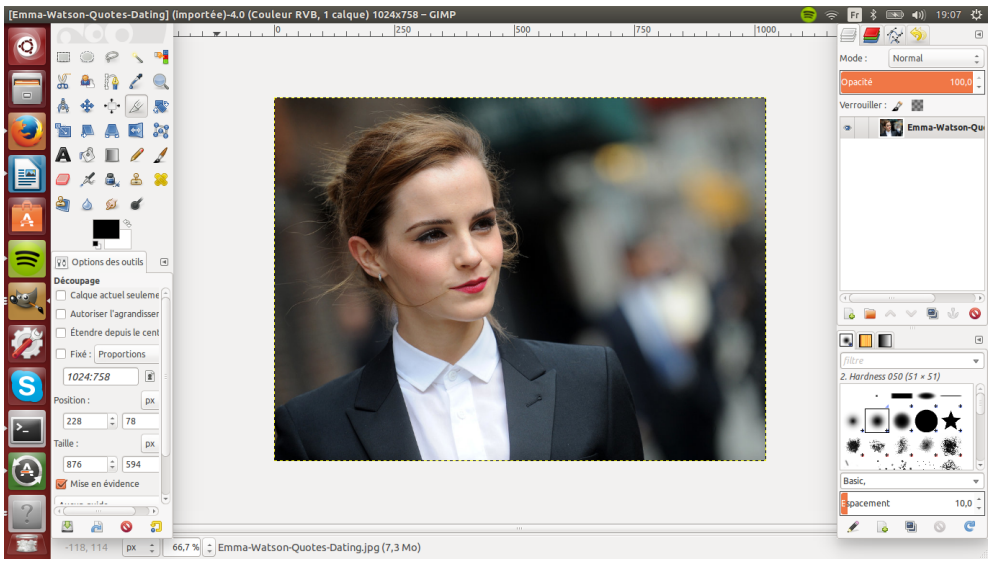

FIGURE 14 – Exemple d'une interface graphique, Gimp

Tout comme Gimp, nous avons utilisé la bibliothèque GTK+2 pour réaliser notre interface. Cette bibliothèque écrite en C est un toolkit offrant une grande quantité de widgets afin de proposer une interface graphique de qualité à l'utilisateur. Nous l'utilisons car c'est actuellement une des bibliothèques les plus utilisées et les plus documentées pour créer des interfaces graphiques en C.

Cela est confirmé par l'existence d'un très bon créateur d'interface graphique, Glade. Celui-ci permet en effet de créer, dans une interface graphique et sans aucune ligne de code, les éléments d'une interface graphique GTK. L'utilisateur peut donc placer à la souris les différents widgets GTK, tels qu'une image, une boîte de texte, des combobox, etc. Cela permet de s'affranchir de très longues lignes de codes souvent indigestes et difficilement modifiables.

12.2 Notre GUI

Contrairement a de nombreux programmes, au sein de notre projet, l'interface graphique n'a pas juste vocation a faciliter la vie de l'utilisateur, elle sert aussi a mettre en commun toutes les parties qui ont été réalisées de manière indépendante, par chacun des membres du groupe.

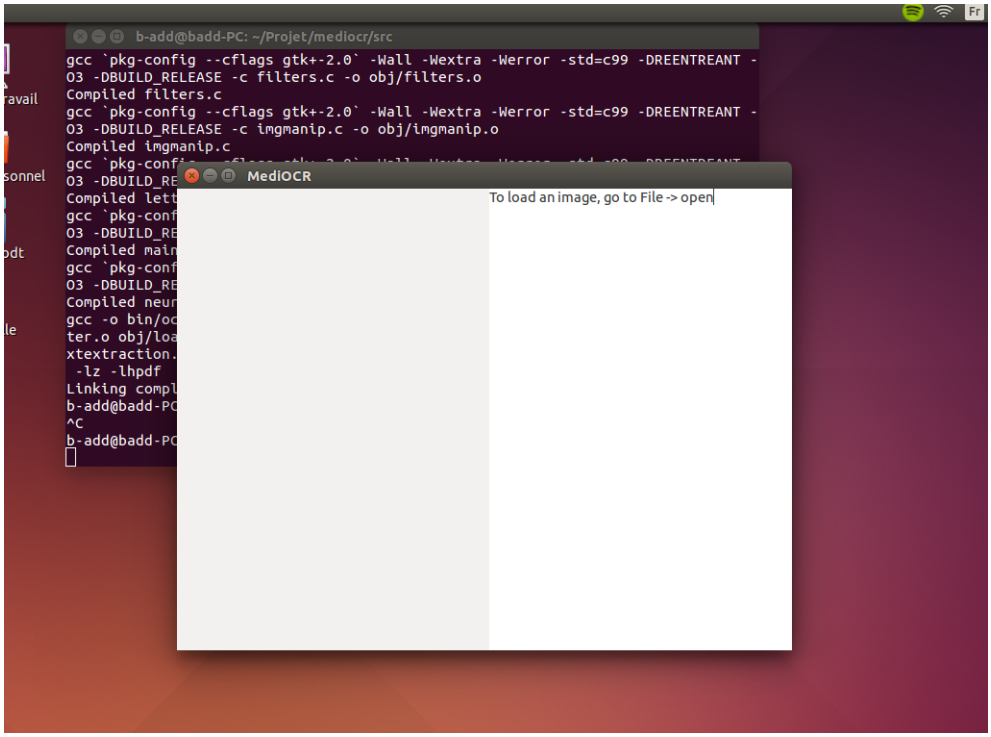

FIGURE 15 – Notre interface graphique une fois lancée

Comme écrit sur la partie droite de notre interface, pour utiliser notre OCR, il faut donc aller dans la barre de menu, dans la sous section «File», puis open, et choisir grâce au sélecteur de fichier, l'image qui sera traitée par notre OCR.

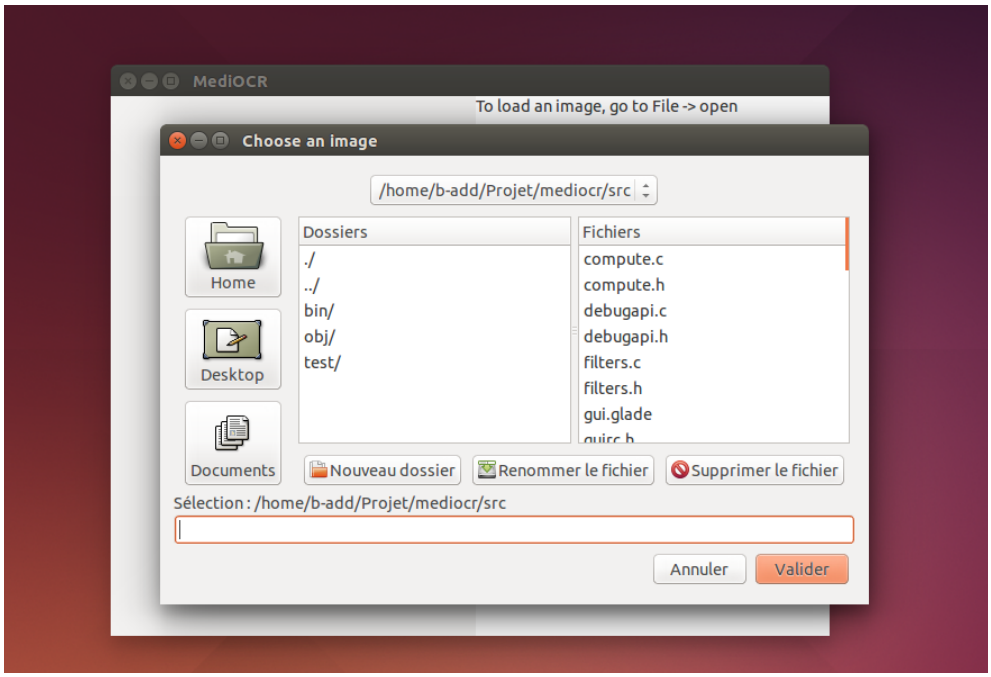

FIGURE 16 – Sélection du fichier cible

Une fois le fichier sélectionné, une boite de dialogue s'affiche pour nous permettre de choisir les différentes options applicables à notre programme, à savoir, les trois types de binarisation (Sauvola, Otsu ou un seuil fixe), la possibilité de retirer le bruit et la rotation, l'utilisation ou non d'un dictionnaire, l'angle de rotation du RLSA et enfin le nombre maximum de threads que l'on souhaite utiliser lors du traitement.

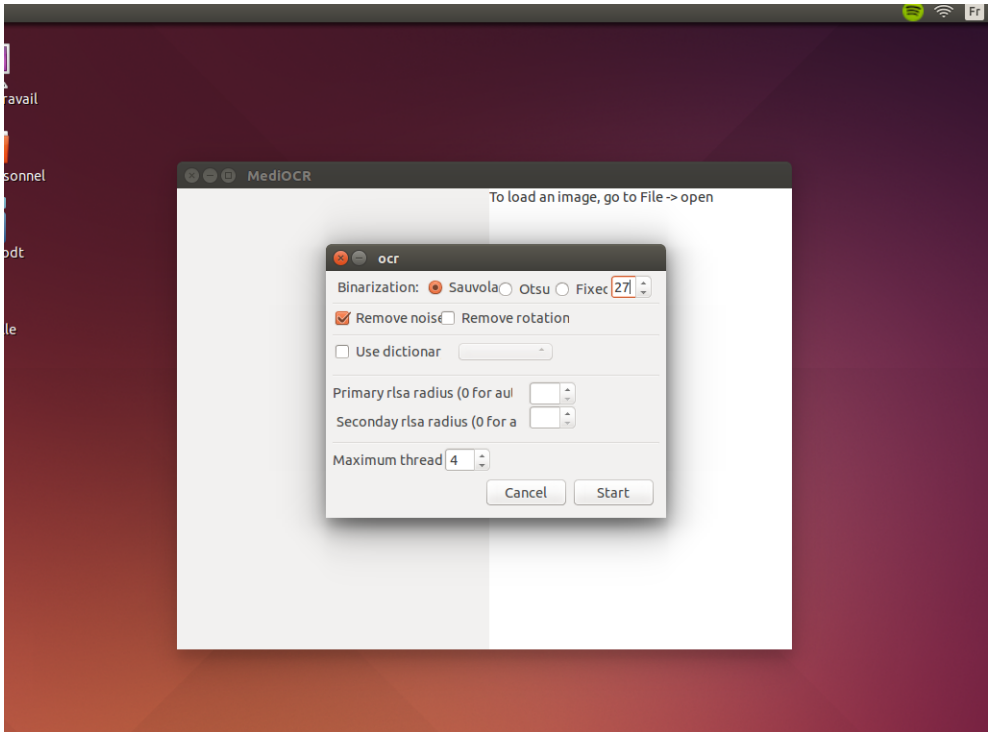

FIGURE 17 – Choix des options

S'affiche ensuite, après le choix des options, l'image chargée sur la gauche de l'interface et sur la droite, après traitement de l'image cible, le résultat de tous les traitements effectués sur celle ci. On a donc en sortie, le texte traité par les différents filtres, le réseau de neurones et le dictionnaire.

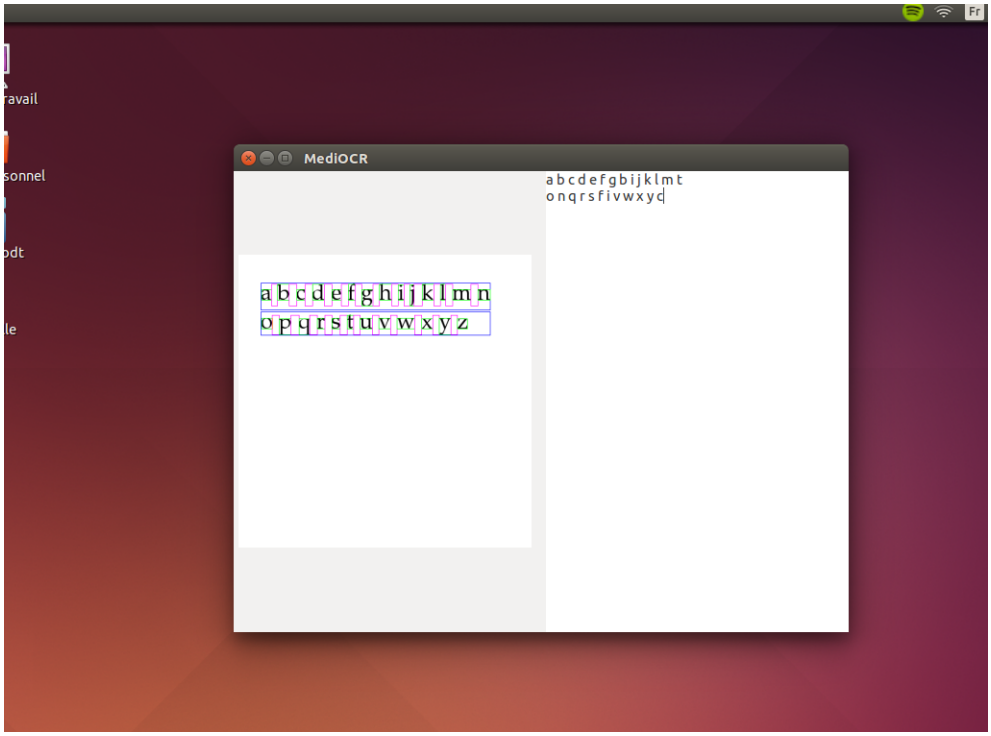

FIGURE 18 – Alphabet traité par notre OCR

Notre GUI possède aussi une fenêtre d'aide où l'on peut trouver le lien vers le site web du projet, ainsi que la date de la dernière compilation du programme et les noms de chacun des quatre membres du groupe.

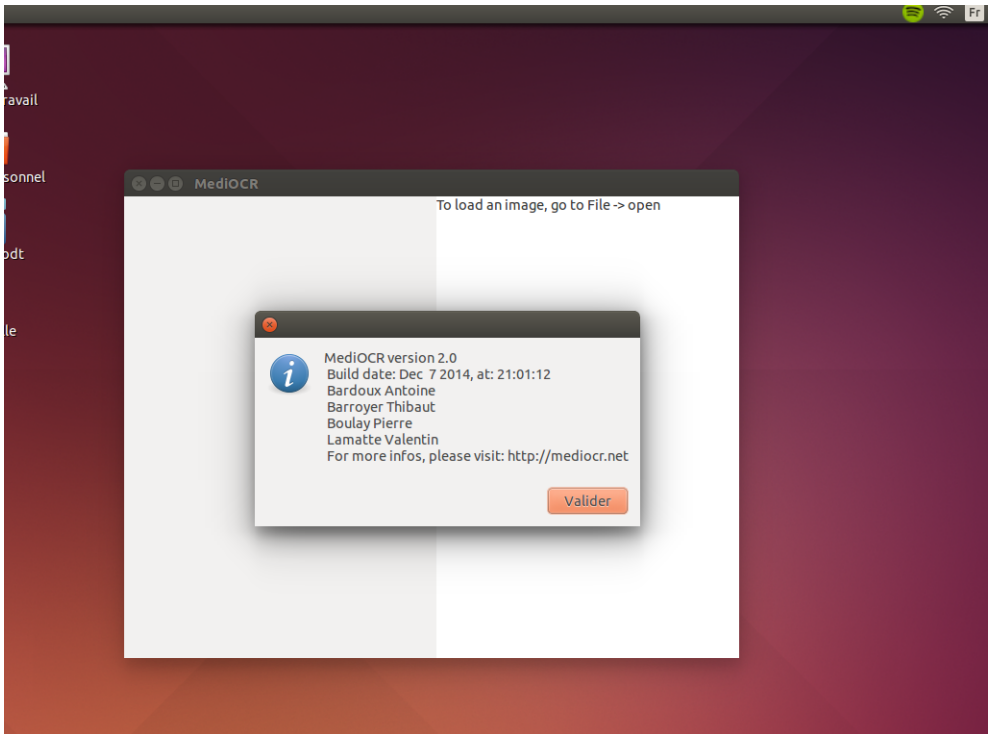

FIGURE 19 – Onglet option de la boite de menu

Et en la dernière feature de notre interface graphique mais non des moindres, nous avons la possibilité d'exporteren PDF le fichier texte, une fois l'image traitée, passée par le réseau de neurones et le dictionnaire.

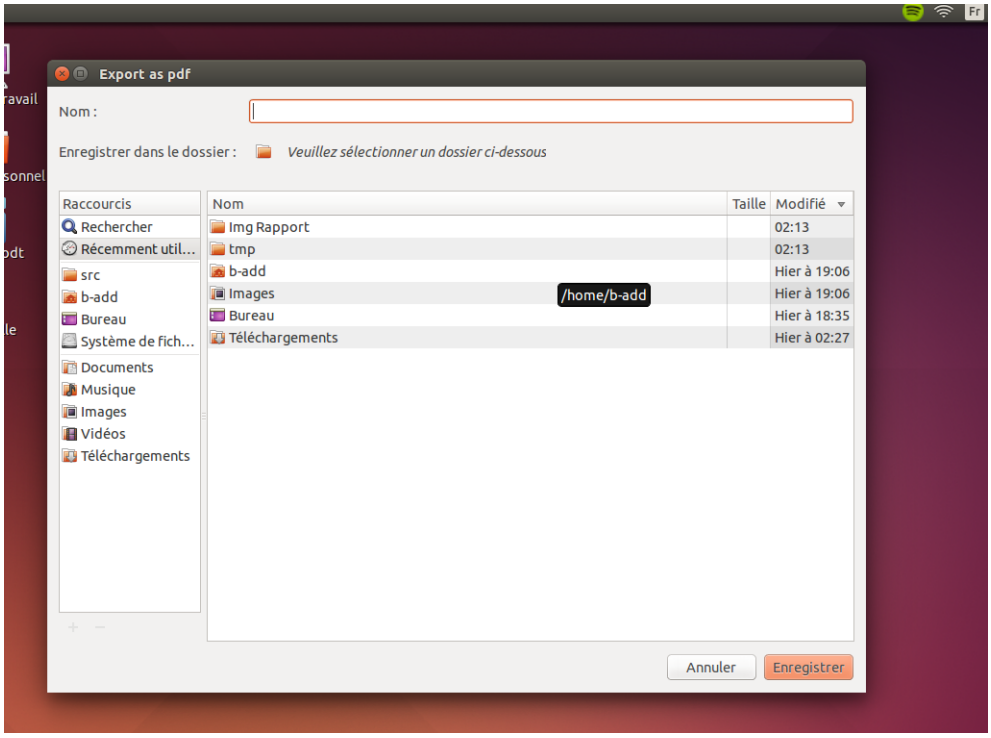

FIGURE 20 – Exportation en PDF du fichier texte

13 Sites web

Antoine Bardoux

Tout comme lors de la première soutenance, nous disposons d'un blog, regroupant différentes choses, telles que la progression de notre projet, des liens pour télécharger les rapports et les versions pré-soutenance de notre projet, ainsi que des liens vers les sites des membres du groupe en possédant un. Nous avions également un site web permettant de tester notre OCR en ligne. À ces deux sites web se rajoute désormais un site web présentant la documentation du code source du projet.

13.1 Blog du projet

Le blog de notre projet est un site statique généré grâce au moteur Pelican. Celui-ci génère des fichiers HTML à partir de fichiers écrits en Markdown ou en reStructuredText. Pelican propose énormément d'options afin de simplifier la génération automatique du site, comme par exemple un hook git afin de re-générer le site à chaque commit sur un dépôt qui contiendrait les fichiers à compiler.

FIGURE 21 – Capture d'écran du blog du projet

13.2 OCR en ligne

J'ai également réalisé un site web en Python qui permet de lancer l'OCR sur mon serveur personnel. Ce site web a été réalisé grâce au framework web Flask, qui permet de créer des sites des plus complexes au plus compliqués. Je n'ai pas utilisé Django car celui-ci n'aurait fait que me compliquer la vie. En effet nous n'avons besoin d'aucune base de donnée et le site ne repose que sur un appel système à l'exécutable via la bibliothèque sh. J'utilise cette bibliothèque car, même si elle n'est pas dans la bibliothèque standard de Python, elle offre une simplicité et une sécurité incomparable, contrairement à `subprocess` et autres.

Ce site web est donc disponible à l'adresse online.mediocr.net. Il permet pour l'instant de binariser l'image ou de détecter les lignes, caractères et blocs, mais il sera amélioré par la suite, avec l'avancement du projet.

13.3 Documentation en ligne

Durant les débuts de notre projet, nous savions qu'il était très important de bien documenter notre code source afin de pouvoir le relire après un petit moment d'absence mais également pouvoir comprendre et utiliser le code des autres membres sans passer un temps inutile à essayer de comprendre à quoi il servait. Cependant, rechercher la documentation d'une fonction dans le code source lui-même n'est pas toujours des plus aisés, car il y a énormément de code autour et la présentation n'est pas des plus agréables.

Ainsi, je me suis attelé à la création d'une configuration Doxygen qui nous permettrait d'avoir un rendu agréable et efficace de notre documentation. Pour rappel, Doxygen est un des standards dans la génération de documentation à partir de code C, C++, Java, Python, etc. Il permet d'exporter la documentation dans différents formats tel du LaTeX, des PDF ou encore des pages HTML. Ce dernier formattage m'a donc permis la création du documentation en ligne. Cependant, afin de ne pas avoir à générer la documentation à la main à chaque commit envoyé sur le dépôt Bitbucket, il m'a fallu écrire un service HTTP recevant les requêtes POST générées par Bitbucket quand un nouveau commit est fait. Ce service a été écrit avec Tornado, afin d'avoir une génération totalement asynchrone. Cette documentation est disponible à l'adresse doc.mediocr.net.

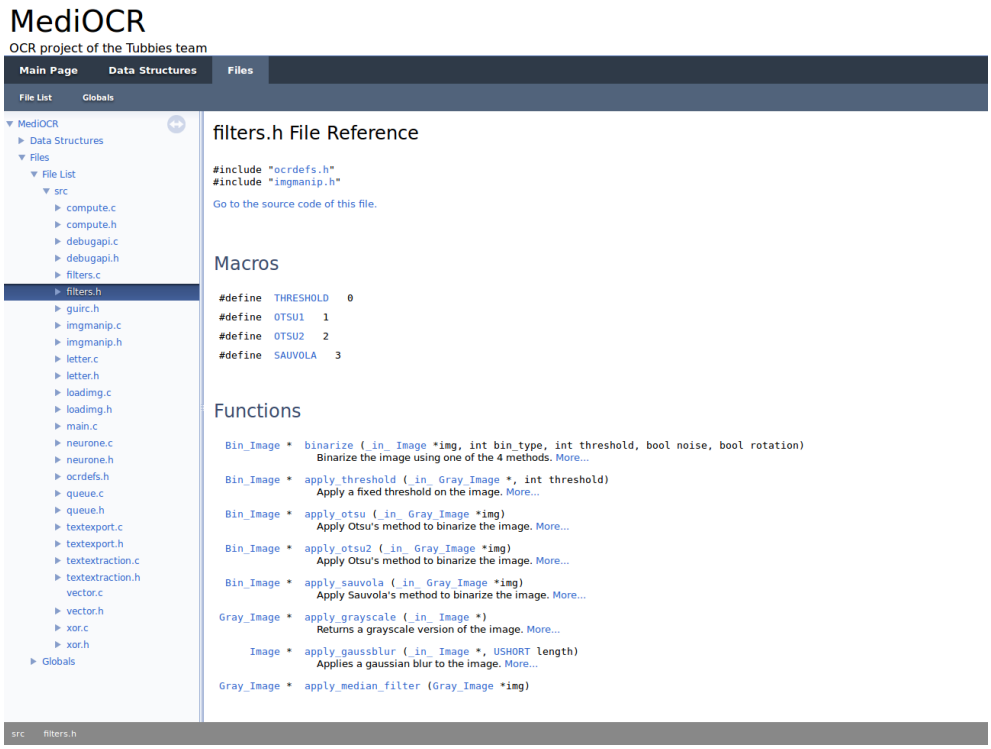

FIGURE 22 – Capture d'écran de la documentation en ligne

14 Impressions personnelles

14.1 Valentin « B-add » Lamatte

C'est donc après près de quatre long mois de travail acharné que ce projet se termine. Au travers de celui j'ai eu la possibilité de doser mes capacités, apprendre à me connaître en temps que codeur, savoir où se trouvent mes limites et quand demander un regard neuf sur mon travail pour continuer à avancer. La seconde chose que j'ai découverte au cours de mon travail est la puissance d'un gestionnaire de version tel que Git, celui ci nous a permis de rester organiser dans notre code, vis à vis de nos parties respective ou même celles des autres. La répartition des tâches a elle aussi beaucoup joué, de cette manière nous avons pu éviter de se marcher sur les pieds lors de nos séances de travail. D'un point de vue de programmation pur, ce projet a constitué une très bonne introduction au C qui se trouve être moins abordable, mais ô combien plus intéressant que le C#. Pour conclure, c'est donc la seconde fois que l'on doit former un groupe de quatre en vue de réaliser un projet au sein de l'EPITA. Cette fois ci, j'ai utilisé ce que m'avais appris mon travail de groupe de l'an dernier pour minimiser les problèmes qui auraient pu survenir et il semblerait que ça ait payé.

14.2 Thibaut « tbarroyer » Barroyer

Ce projet était pour moi le premier projet à réaliser sous le système d'exploitation Linux. Cela m'a permis de découvrir un monde que je ne connaissais pas. Un monde sans IDE. Un temps d'adaptation a été nécessaire afin que je puisse coder de façon rapide et ordonnée. J'ai appris à utiliser Vim, à le configurer et à utiliser ses nombreux raccourcis fort utiles. J'ai également appris à me servir du débogueur "gdb", difficile à utiliser au début, mais qui s'avère être en réalité très simple d'utilisation. J'ai pris l'habitude d'utiliser les branches sur git, chose que je ne faisais pas l'an passé, ce qui est bien pratique pour ne pas embêter mes camarades avec du code incomplet. Je me suis occupé du réseau de neurone, tâche nécessitant beaucoup de documentation. J'ai donc appris à me documenter, à rechercher. J'ai appris beaucoup de choses sur l'intelligence artificielle, sur les réseaux de neurones, ainsi que sur le "machine learning". Ce fut une expérience très enrichissante sur un domaine que je ne connaissais pas. Mon seul regret est de ne pas avoir eu le temps de me renseigner sur le traitement d'image, étant donner le travail nécessaire pour obtenir un réseau fonctionnel.

14.3 Antoine « Pluggi » Bardoux

Ce projet a avant tout été pour moi l'occasion d'apprendre le C et à m'améliorer dans mon utilisation de Vim. En effet, j'utilisais déjà Arch Linux depuis plusieurs années mais je ne m'étais intéressé à Vim qu'avant l'été et n'avait donc pas eu le temps de vraiment approfondir mes connaissances. Mais ce projet d'OCR nécessitant de coder pendant très longtemps, je me suis créé une configuration m'aidant lorsque je code, grâce à divers plugins et raccourcis claviers créé par mes soins. J'ai aussi eu à me confronter au monde de Git et de ses différentes fonctionnalités telles que les branches durant ce projet. C'est un réel investissement qui me sera très profitable par la suite pour garder un dépôt propre. Nous avons en effet créé pas moins de 9 branches qui nous ont servi pour coder sans se gêner les uns les autres. De plus, je n'avais jamais réellement codé dans un langage aussi bas niveau que le C, même si j'avais fait rapidement du C++. C'est un tout autre univers de celui auquel je suis habitué (les langages haut niveau type Python et OCaml) et je me suis donc retrouvé face à des erreurs auxquelles je n'avais jamais eu à faire face auparavant. Ainsi, malgré les manques flagrants du C, j'apprécie coder avec car j'en apprend un peu plus à chaque erreur et que je sors de ma zone de confort.

14.4 Pierre « Bluescreen » Boulay

Avant de commencer le développement de MediOCR, je n'avais pas beaucoup d'expérience avec Linux. J'ai été surpris de découvrir des outils tels que Vim, Emacs, grep, sed, etc. d'une puissance incroyable. Ces nouveaux outils m'ont permis de donner le meilleur de moi-même et de m'investir à cent pour cent dans ce projet.

De même, j'avais de faibles bases sur le traitement d'image en ce début d'année et je me rends compte aujourd'hui des progrès que j'ai fait dans ce domaine. J'ai dû réfléchir à des algorithmes que je n'avais jamais eu besoin d'utiliser avant, ce qui m'a permis d'élargir mes horizons, en voyant de manière nouvelle des problèmes complexes.

De plus, la complexité de ces algorithmes m'a fait réaliser à quel point le cerveau humain est performant pour lire du texte à partir des images qu'il reçoit. J'ai passé beaucoup de temps à mettre en place l'extraction des caractères et j'ai bien vu qu'il existe un grand nombre de situations dans lequel mes algorithmes ne seraient pas capables de détecter le texte. Je garde donc un très bon souvenir de ce projet qui m'a ouvert une fenêtre sur le monde de la recherche et du traitement d'image.

15 Conclusion

Voici donc venir les dernières lignes du rapport de notre «MediOCR» projet, et ainsi se conclu ce qui qui aura constitué la majeure partie de notre troisième semestre à l'EPITA. Le travail sur cet OCR nous a donc au final beaucoup apporté au niveau personnel, mais aussi, bien évidemment, au sein du groupe en lui même, car contrairement au projet de SUP où l'on apprend plus ou moins à coder, il nous a fallu cette fois-ci savoir s'adapter à la façon de travailler des autres pour continuer à évoluer dans un environnement de travail sain. C'est donc, d'une certaine manière, une part de nous même que nous laissons au travers de ces dernières lignes...