Rapport de soutenance 2

Textomaton

par Trust us, we have Adaboost

Corentin Vigourt (vigour_b) *Chef de groupe*

Matthieu Moatti (moatti_m)

Tom Moulard (moular_b)

Noé de Larminat (delarm_n)

6 décembre 2016

Table des matières

Introduction

Ce projet entre dans le programme de la deuxième année de l'EPITA (École pour l'informatique et les techniques avancées) et est à réaliser en groupe de maximum quatre personnes. Sa durée est d'un semestre, de Septembre a Décembre. Ce rapport de soutenance décrit la nature du projet ainsi que les différentes parties qui ont composé sa création pour le moment.

Ce projet est la création d'un logiciel de reconnaissance de caractères, OCR (Optical Character Recognition) en anglais. Un tel logiciel extrait le texte contenu dans un fichier image (document scanné, photo...). Il peut aller jusqu'à la création d'un document de traitement de texte contenant les information lisible sur l'image (texte, mise en page, images hors texte et leurs positions / tailles...)

D'un point de vu technique, ce projet nommé *Textomaton* sera réalisé en langage C de standard C99 par le groupe **Trust us, we have Adaboost** constitué de Corentin Vigour, Matthieu Moatti, Tom Moulard ainsi que de Noé de Larminat.

Mais comment construit-on un OCR ? Nous avons séparé le problème en trois :

— Le traitement de l'image qui consiste a traiter l'image de façon à ce qu'elle soit traitable par le reste de l'ocr (cela peut comprendre la conversion en niveau de gris, la rotation d'une image non alignée etc...)

— La reconnaissance de caractères qui permet une découpe de l'image afin de séparer les caractère pour ensuite les reconnaitres un à un.

— Le réseau de neurones, qui est l'élément permettant de reconnaître chaque caractère, de manière individuelle.

1 Présentations

1.1 Le groupe

Nous sommes quatre étudiants a l'EPITA (école d'ingénieurs en informatique) en seconde année de classe préparatoire.

1.2 Les membres

1.2.1 Tom -Aikon- Moulard

Auteur du réseau de neurones.

1.2.2 Matthieu -Melyodas- Moatti

Run length smoothing. Segmentation. Aide au réseau de neurones.

1.2.3 Corentin – Vigourt

Auteur du traitement de l'image et de l'interface utilisateur

1.2.4 Noé – De Larminat

Initialement ouverture de l'image et création de la base de données, finalement aide pour l'interface utilisateur

2 Compte-rendu du projet

Dans ce rapport, nous allons vous présenter le résultat final de notre OCR, mais aussi le déroulement concret du développement de ce projet.

Ce projet fut pour nous l'occasion d'apprendre à utiliser le langage C sur un projet de plusieurs mois durant lequel nous avons dû, rechercher, apprendre, essayer, débugger et mettre le tout en commun car il n'y a qu'un seul programme à la fin.

En plus de la maîtrise du langage C, nous avons appris beaucoup. En effet, nos membres ont dû apprendre par soit-même afin de réaliser leurs tâches. Que ça soit pour le réseau de neurones, le traitement de l'image, de la détection de caractères ou bien de l'interface utilisateur.

De plus, il a fallu apprendre à travailler encore plus en groupe. Chaque partie du code étant nécessaire au bon fonctionnement de l'OCR, il fallait avoir finis sont travail dans les temps mais aussi mettre en commun afin de pouvoir utiliser les différentes parties du projet les unes avec les autres. Ce projet nous as demandé beaucoup de temps pour faire tout fonctionner ensemble.

3 Traitement de l'image

3.1 Chargement de l'image

3.1.1 La version expérimentale

Pour la préparation de cette première soutenance le chargement des images a été géré avec la bibliothèque SDL pour ne pas avoir a aussi bien attendre que dépendre des résultats sur l'avancé de la gestion manuel du chargement. Il y a tout de même eu des recherches sur la gestion du chargement d'images sans utilisation de la bibliothèque SDL, mais seulement des bibliothèques standards.

Nous avons maintenant accès a deux fonctions nommées bmpWidth et bmpHeight contenues dans images/database.c qui renvoient respectivement le nombres de pixels de l'image en largeur et en hauteur d'un fichier BITMAP (.bmp). Ces fonctions s'appuient sur la mise en forme des données binaires du fichiers contenues dans ses 54 premiers octets par le biais d'une définition de structure. Ces 54 octets constituent le header d'un fichier BMP. Ils contiennent diverses informations sur le fichier. Par exemple les deux premiers octets constituent un identifiant de type du fichier. La signature propre aux fichiers bmp est égale à 4D42 en Hexadécimal, donc si les 2 premiers bytes du fichier ne sont pas égaux à cela, il y a une erreur dans le type du fichier car ce n'est pas un fichier BMP.

FIGURE 1 – image représenté par cet affichage mémoire.

Voici des illustrations des deux parties du header (entête) du fichier BMP ici présent. Le premier contient des informations vraiment générales sur le fichier tandis que la seconde partie contient des informations spécifiques au format bmp, appliqués au fichier. Par exemple la taille du fichier se trouve dans la première partie, tandis que le nombre de pixels en largeur de l'image se retrouve dans la deuxième partie de cet entête.

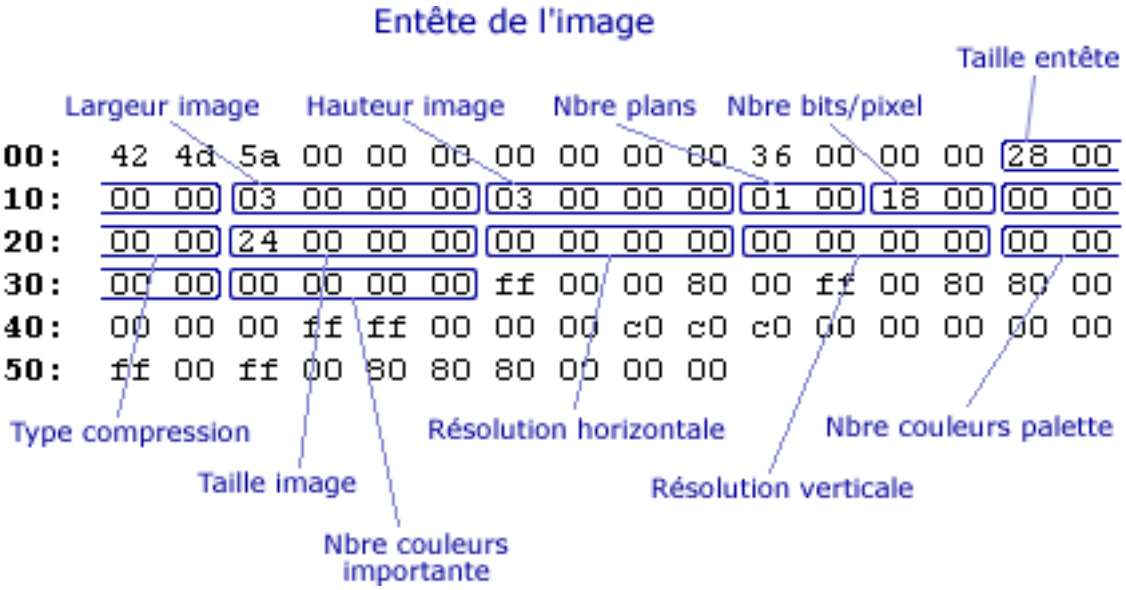

FIGURE 2 – Première partie d'un entête de fichier BMP, se basant sur le fichier d'une façon générale.

Ici nous pouvons remarquer que la signature est 424D contrairement au 4D42 indiqué un peu plus haut. L'ensemble de deux octets sera lu avec une orientation (endianness en anglais, souvent traduit par endianisme ou boutisme) petit-boutiste (big-endian), c'est à dire de droite à gauche pour la représentation ci-dessus.

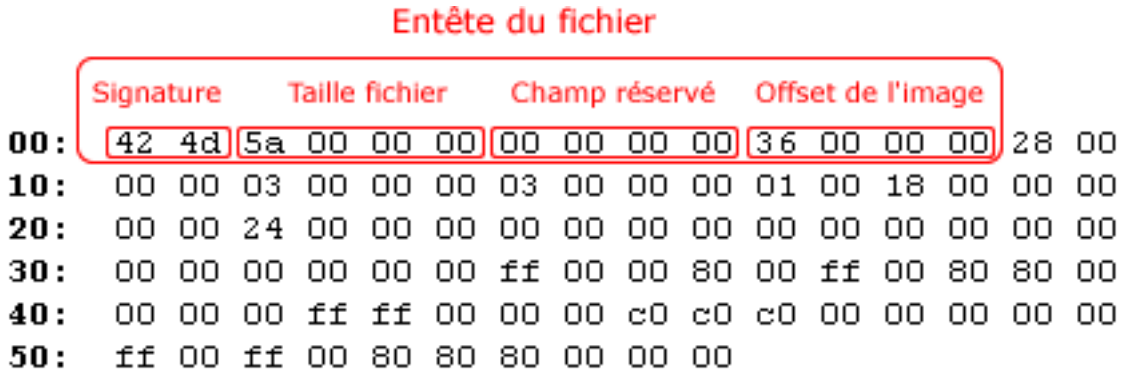

FIGURE 3 – Deuxième partie d'un entête de fichier BMP, contenant des informations du fichier d'un point de vue spécifique au format BMP.

Une base de donnée

L'implémentation d'une base de données contenant toutes les images contenues dans un dossier fut un grand challenge due aux limitations d'un point de vu des bibliothèques.

Finalement, la base de données actuelle se situe dans un fichier nommé data se trouvant dans le dossier images. Elle est créée par le biais de la fonction writeAllFiles qui fait appel à la fonction de lecture d'information du header de BMP readHeadBMP. Cette fonction cherche tous les fichiers nommés *i.extension* avec i un entier strictement positif et extension comprise entre : bmp, jpeg et jpg. La fonction s'arrête dès que aucune des extensions ne donnent a i un fichier existant (si ni i.bmp, i.jpeg ou i.jpg n'existent). 5 lignes sont associées par fichiers trouvés : le nom, le type de fichier (bmp, jpeg...), la taille du fichier en octets puis la largeur et hauteur de l'image en pixels.

3.1.2 Ouvrir l'image

Le but de ce projet étant de lire le texte dans une image, le point de départ est d'ouvrir l'image. Pour cela, on a utilisé la SDL afin de récupérer la valeur RGB de chacun des pixels de l'image. Pour le moment, nous ne lisons que les BMP car ils contiennent dans leurs headers leurs tailles à la différence des JPEG. Grâce un script, nous récupérerons les tailles d'un fichier BMP en parcourant son header.

Pour la suite, on stocke l'ensemble de l'image dans un nouveau type, une PixelMatrix qui est une matrice contenant des données de type Pixel.

```
typedef struct _Pixel{
    Uint8 r;
    Uint8 g;
    Uint8 b;
}Pixel;

typedef struct _PixelMatrix{
    size_t lines;
    size_t cols;
    Pixel *data;
}PixelMatrix;
```

Ces deux types nous simplifient énormément la tâche puisque les pixels ne contiennent que leurs valeurs RGB.

3.2 Conversion en niveau de gris

La première conversion à faire sur l'image c'est la conversion en niveau de gris. Pour cela, nous utilisons cette équation qui nous permet de calculer le niveau de gris pour un pixel avec ses valeurs RGB :

$$grey\_level = \sqrt{r^2 + g^2 + b^2} \tag{1}$$

On parcourt donc l'image en remplaçant les valeurs RGB des pixels par leurs niveaux de gris. Cette conversion est nécessaire pour utiliser la méthode d'Otsu (qui sera détaillé après) qui est l'une des meilleurs méthode de conversion d'image en noir et blanc.

FIGURE 4 – Image en nuance de gris

3.3 Méthode d'Otsu

Après quelques recherches, une des méthodes qui semblait être la plus fiable était la méthode d'Otsu. C'est une méthode qui effectue un seuillage automatique de l'image grâce à l'histogramme du nombre de pixel par niveau de gris. Cette algorithme se base sur le fait qu'une image dans la plupart des cas ne contient que deux plans, le premier plan et l'arrière plane. L'algorithme calcule le seuil optimal qui sépare les deux plans de l'image.

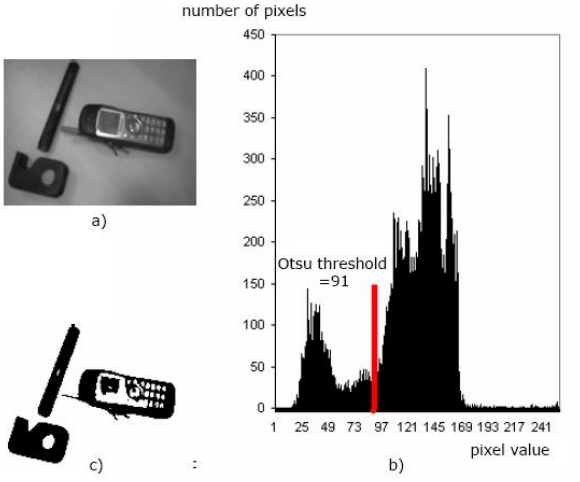

FIGURE 5 – Calcul du seuil

FIGURE 6 – Image binarisée

Pour stocker l'image après la conversion en noir et blanc, nous avons choisi d'implémenter un nouveau type de données qui rendra le travail plus simples pour la suite du traitement de l'image. Nous avons donc créer le type UnsignedMatrix :

```
typedef struct _UnsignedMatrix{
    size_t lines;
    size_t cols;
    unsigned *data;
}UnsignedMatrix;
```

Nous avons choisis d'utiliser des nombres stockés en unsigned parce c'est le plus petit type de données disponible. Ce type nous permet de stocker une matrice contenant juste des 0 et des 1 correspondants à l'image binarisée.

3.4 Rotation de l'image

Dans le cas où l'image rentrée par l'utilisateur n'est pas correctement orienté, il faut la réorienté afin de pouvoir faire fonctionner les algorithmes de reconnaissance de texte dessus.

Pour cela, on créer une image où l'on va utilisé ces équations en parcourant l'ensemble de l'image existante pour connaître la nouvelle position du pixel :

$$x1 = \cos(\theta) * (x0 - xcenter) - \sin(\theta) * (y0 - ycenter) + xcenter \tag{2}$$

$$y1 = \sin(\theta) * (x0 - xcenter) + \cos(\theta) * (y0 - ycenter) + ycenter \tag{3}$$

Dans cette équation :

1. (x0, y0) : coordonnées du pixel dans l'image source
2. (x1, y1) : coordonnées du pixel dans l'image destination
3. (xcenter, ycenter) : centre de la rotation dans l'image source
4. theta : angle de rotation

Pour ne pas perdre de données, il faut calculer la nouvelle taille de l'image en prenant les points les plus éloignés de la nouvelle image.

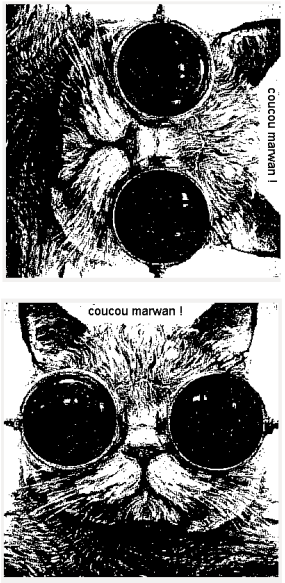

FIGURE 7 – Image avant/après une rotation de 90 degrés

4 Interface Utilisateur

Pour ce rendu final intervient l'implémentation d'une interface graphique (GUI pour Graphical User Interface). Mais qu'est-ce qu'une interface graphique, et quelle en est son utilité ?

L'interface graphique permet l'accès aux fonctionnalités d'un programme de façon, comme son nom l'indique, graphique. Ainsi l'utilisateur n'a dans notre cas, non pas accès au shell mais à une fenêtre agrémenté de différents bouton et champs de retour. Mais nous verrons tout cela un peu plus loin...

Nous pouvons bien évidemment nous passer d'interface graphique. Néanmoins, une telle interface permet de grandement simplifier l'interaction utilisateur-machine, surtout dans le cas d'utilisateurs novices ou n'étant pas familier au programme proposé.

L'interface graphique permet ainsi à la grande majorité des gens d'interagir avec le programme, car elle présente les différentes options, tout en laissant leurs exécution a une affaire de quelques clics.

Ceci donne à l'interface un aspect primordial dans un programme / logiciel. En parcourant diverses documentations et guideline de design d'interfaces machine nous pouvons remarquer quelques conseils très utiles. Par exemple proposer dans l'interface d'accueil des options par défaut qui conviendront à l'utilisateur lambda, tout en gardant cette interface claire en ne la surchargent pas. Les options avancées pouvant être activé par les utilisateurs les plus assidue si besoin.

Exemple d'interface graphique

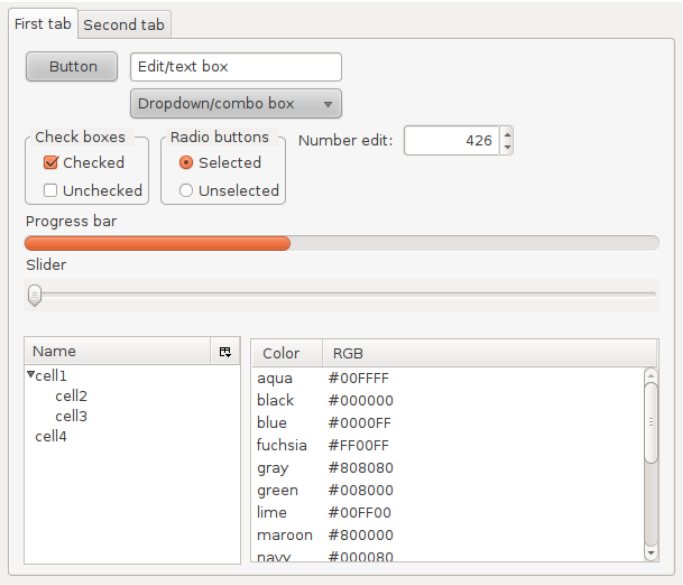

FIGURE 8 – Exemple d'interface contenant plusieurs widgets typique des interfaces graphique, Wikipédia

Le design de ces widgets fait partis de ceux généralement utilisé sous Linux. On peut ainsi remarquer aussi bien un bouton (accompagné de son label "Button") qu'une barre de progression. Sur l'interface de Textomaton, nous pouvons remarquer les boutons, labels, mais aussi un explorateur de fichiers appelé à l'aide d'un bouton spécifique ainsi que des affichages d'images et de textes.

4.1 Découverte de GTK et Glade

Il nous a été demandé de réaliser un GUI (alias Graphical User Interface) afin de rendre l'utilisation de notre programme plus facile. Après quelques recherches, nous avons décidé d'utiliser la bibliothèque GTK en nous aidant du logiciel Glade afin de mettre en forme notre GUI.

GTK est une bibliothèque qui nous permet d'afficher via des Widgets divers contenus (texte, image, ect...). Elle offre une variété de type différent afin de pouvoir avoir un affichage performant et facile à manipuler. D'ailleurs, elle est disponible dans plusieurs langages (C, C#, C++ et python). Notre choix fut vite fait, GTK proposant le plus de contenu pour créer une interface graphique facile à manipuler en C.

De plus, nous avons pu utiliser Glade, un logiciel qui permet de mettre en forme des éléments graphiques de GTK via une interface plutôt facile d'utilisation et qui génère un fichier qu'il suffit d'importer via GTK dans le code C afin d'afficher les éléments programmés.

4.2 Création de la première interface

Il fallait bien commencer quelque part. Dans un premier temps, nous avons fait une première interface qui contenait un bouton pour lancer le code et une boîte de dialogue permettant à l'utilisateur de choisir un fichier. À ce moment-là, on ne faisait que d'afficher l'image choisie dans une autre fenêtre.

Ensuite, nous avons essayé de rajouter des boutons pour faire faire une rotation à l'image. Pour cela, nous avons rajouté 5 boutons pour faire une rotation de : -1degré, -10 degrés, +90 degrés, +10 degrés, +1 degré.

Il a fallut faire le code derrière pour gérer les différents widget de GTK placé grâce à Glade. La première étape fut de récupérer dans le fichier Glade la fenêtre ainsi que les widgets placés dessus. Une fois cela fait, il suffisait d'utiliser les bonnes fonctions correspondantes aux types de widgets utilisés et le tour est joué.

4.3 Version finale

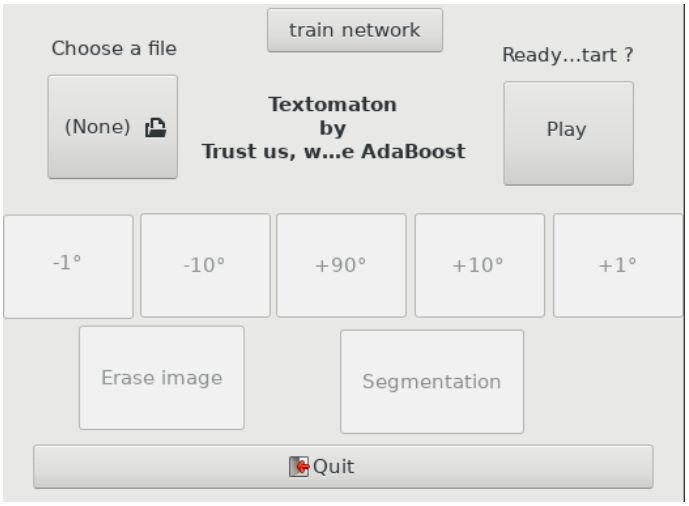

FIGURE 9 – Version finale de l'interface utilisateur de TEXTOMATON

À la fin du développement de l'interface, nous avons : une interface permettant de charger une image, la faire tourner et lancer l'OCR sur cette image qui affiche à la fin une nouvelle fenêtre avec le texte détecté

Nous avons rajouté un bouton pour lancer l'entraînement du réseau de neurones mais aussi deux autres boutons afin d'enlever des photos pouvant être présentes sur l'image rentrée par l'utilisateur mais aussi pour segmenter le texte selon la mise en page de

5 Reconnaissance de caractères

5.1 Algorithme basique

Utilité

L'algorithme basique est utile afin de découper facilement des lettres dans une image composée uniquement de texte, celui-ci sera ainsi pratique pour apprendre plus rapidement au réseau de neurone.

Principe

Ainsi on parcourt une image "binarisée" pixel par pixel, lorsque l'on rencontre un pixel noir on applique l'algorithme suivant :

- On passe le pixel courant en blanc.
- On garde la position dans quatre variables : xmin, xmax, ymin, ymax
- On génère la liste des pixels noirs dans une zone prédéfinie autour du pixel courant.
- Tant que la liste n'est pas vide :
  - On applique l'algorithme récursivement sur le premier élément de la liste.
  - On met à jour les variables xmin, xmax, ymin, ymax.
  - On génère à nouveau la liste des pixels voisins.
- On retourne les valeurs (xmin,ymin) et (xmax,ymax) qui correspondent à deux angles opposés du cadre entourant la lettre.

Implémentation

Afin d'implémenter cet algorithme nous avons choisi de passer par deux structures afin de faciliter le passage de valeurs d'une fonction à une autre.
Une structure Coordinates qui prend une coordonnée x et une coordonnée y.

```
typedef struct Coordinates Coordinates;
struct Coordinates{
  size_t x;
  size_t y;
};
```

Une structure Rect qui prend deux coordonnées a1 et a2.

```
typedef struct Rect Rect;
struct Rect{
  Coordinates a1;
  Coordinates a2;
};
```

Ainsi il est possible de retourner une liste de Rect qui correspond à la liste des cadres entourant les lettres de l'image analysée.

5.2 Le découpage XY récursif

Utilité

Le découpage XY récursif (ou XY-cut) est un algorithme de segmentation qui parcourt une image de gauche à droite et de haut en bas afin de renvoyer les différentes zones de texte dans l'ordre de lecture.

Principe

L'algorithme consiste à parcourir l'image de haut en bas afin de trouver la plus large bande blanche horizontale et de séparer l'image en deux à partir de celle-ci (découpage Y). On stocke ensuite les deux parties découpées et on applique l'algorithme récursivement sur les deux parties, cette fois-ci en parcourant l'image de gauche à droite afin de trouver la bande blanche verticale la plus large et de séparer l'image en deux à partir de celle-ci(découpage X). on réapplique alors le découpage Y sur chaque image alors générée.
L'algorithme s'arrête soit lorsqu'il n'y a plus de bande blanche d'une largeur suffisante (valeur choisie en amont).

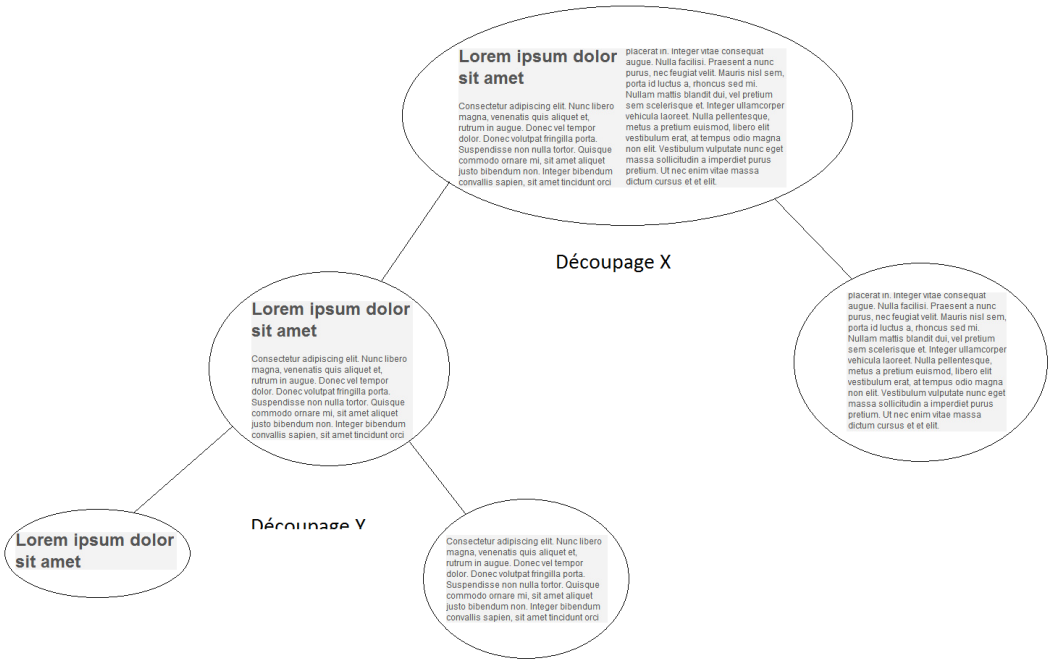

FIGURE 10 – arbre représentant le découpage XY

Implémentation

Afin d'implémenter le découpage XY récursif nous avons choisi de passer par une structure MatBinTree. Cette structure est un arbre binaire qui prend en clef la matrice correspondant au nœud courant et un Rect qui correspond à la position de la matrice du nœud courant dans celle de la racine.

```
typedef struct _MatBinTree{
  UnsignedMatrix     *key;
  Rect               pos;
  struct _MatBinTree *left;
  struct _MatBinTree *right;
}MatBinTree;
```

5.3 Segmentation de paragraphe

5.3.1 Utilité

L'algorithme basique est certes utile pour vite récupérer l'ensemble des lettres d'un paragraphe mais il a plusieurs défauts :

— Il est récursif sur chaque pixel de chaque lettre

— Il est très compliquer de récupérer l'ordre de lecture

Ainsi l'algorithme de segmentation que nous avons implémenté résout ces deux problèmes

5.3.2 Principe

Ainsi pour segmenter l'image en gardant l'ordre de lecture et pouvant être facilement appliqué sur chaque feuille de l'arbre récupéré via le découpage XY nous avons choisi de faire un algorithme se rapprochant beaucoup de celui-ci à la différence près que le découpage n'alterne pas entre découpage horizontal et vertical mais que le texte est découpé horizontalement jusqu'à ce qu'il n'y ait plus de lignes blanches puis verticalement avec la même condition d'arrêt.
Ceci nous permet alors d'avoir un arbre binaire dont les feuilles sont les lettres du texte de la racine dans l'ordre de lecture.

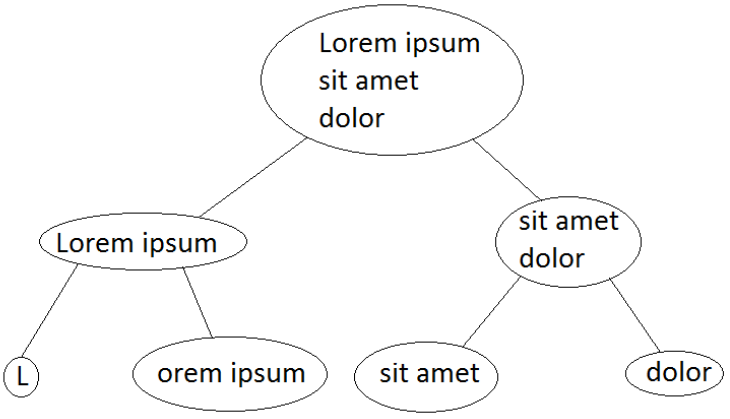

FIGURE 11 – arbre représentant le découpage XY

5.4 Reconstruction

5.4.1 Utilité

Une fois que notre texte a été segmenté et que les lettres ont été détectées par le réseau de neurone il faut pouvoir récupérer la mise en page d'origine.

5.4.2 Principe

Le texte ayant été segmenté dans un arbre binaire, la chaîne de caractère portée par chaque nœud correspond à la concaténation des chaînes de caractères portées par les fils avec ajout ou non d'espaces ou de retours à la ligne. L'algorithme se présente dont ainsi :

- Si l'arbre n'existe pas :
  - Retourner une chaîne de caractère vide
- Si c'est une feuille :
  - Appeler le réseau de neurone pour récupérer le caractère
- Si c'est un nœud :
  - Créer une chaîne de caractère qui correspond à la coupure effectuée.
  - On concatène la chaîne de caractère obtenue avec le retour de l'appel récursif sur le fils gauche et la précédente
  - On concatène la chaîne avec le retour de l'appel récursif sur le fils droit
- On retourne la chaîne de caractère porté par le nœud.

5.4.3 Implémentation

Afin de savoir si on ajoute ou non des espaces ou des retours à la lignes, nous avons choisi de modifier la structure de l'arbre binaire afin de rajouter l'information du découpage effectué pour créer les deux fils.

```
typedef struct _MatBinTree{
  UnsignedMatrix     *key;
  Rect               pos;
  int                hor;
  int                ver;
  char*              txt;
  struct _MatBinTree *left;
  struct _MatBinTree *right;
}MatBinTree;
```

Ainsi la variable hor (respectivement ver) correspond à la largeur de la bande blanche horizontale (respectivement verticale) qui a été découpée afin de pouvoir déterminer si il y a un espace ou un retour à la ligne entre les chaînes de caractères portées par les fils.
La variable txt, elle correspond à la chaîne de caractère portée par le nœud courant. (celle de la racine correspondant ainsi au texte étudié).

5.5 Le Run Length Smoothing Algorithm

Utilité

Le RLSA (ou CRLA), est utilisé afin de séparer le texte du reste dans une image binaire.

Principe

Le RLSA basique est appliqué sur une image "binarisée" où les 0 correspondent au blanc et les 1 au noir, l'algorithme transforme une séquence binaire x en une séquence y en suivant ces deux règles :

1. Les 0 dans x sont changés en 1 dans y si le nombre de 0 adjacents est inférieur ou égal à une limite prédéfinie C
2. Les 1 dans x restent inchangés dans y

Par exemple avec C = 4 la séquence x est transformée en y :

x : 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0
y : 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1

Ainsi sur une image "binarisée", cet algorithme va joindre les pixels qui sont à une distance inférieure ou égale à C.
On applique alors le RLSA lignes par lignes et colonnes par colonnes et on crée une nouvelle image via un ET logique entre les deux images obtenues.
Les images sont alors détectées et supprimées via une série de calculs appliqués sur l'image d'origine et l'image obtenue.

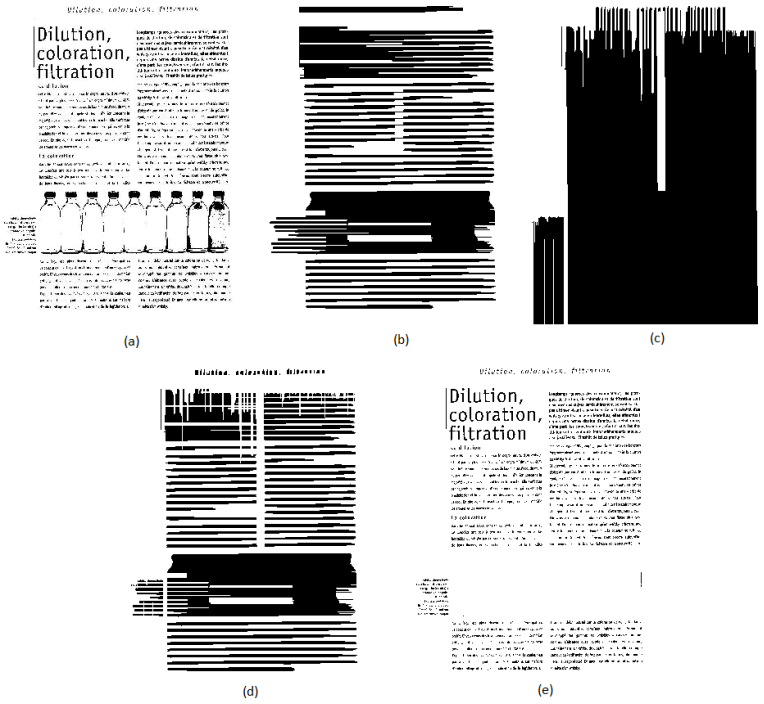

FIGURE 12 – a. image binarisée b. Application d'un RLSA horizontal c. Application d'un RLSA vertical d. Résultat après application d'un ET logique e. suppression de l'image

5.6 Le découpage par composantes connexes

Utilité

Le découpage par composantes connexes est utilisé afin de regrouper les différents pixels qui composent un élément dans une image en leur attribuant une étiquette différente pour chaque élément, il sert ainsi à extraire plusieurs informations de ces éléments tels que sont aire ou sa position via un parcourt itératif de l'image.

Principe

Lorsque nous parlons d'évaluer le voisinage d'un pixel, nous utilisons le principe de 8-connexité, c'est à dire que nous évaluons les 8pixels qui entourent le pixel courant.

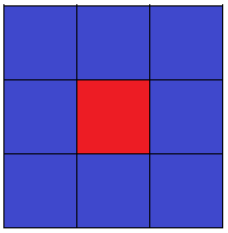

FIGURE 13 – 8-connexité

Afin d'optimiser l'algorithme, lors de la descente (parcours de gauche à droite et de haut en bas), nous n'utilisons que les trois pixels situés au dessus du pixel courant et celui directement à sa gauche :

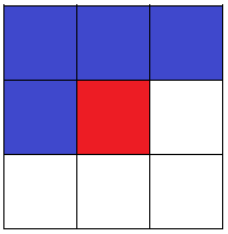

FIGURE 14 – Pixels observés à la descente

Et lors de la remontée (parcours de droite à gauche et de bas en haut), nous n'utilisons que les trois pixels situés en dessous du pixel courant et celui directement à sa droite :

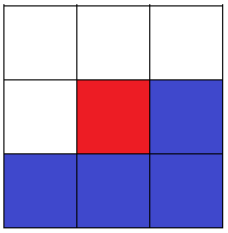

FIGURE 15 – pixels observés à la remontée

Le découpage par composantes connexe consiste tout d'abord à parcourir l'image pixel par pixel de gauche à droite et de haut en bas, lorsque l'on rencontre un pixel noir on lui attribue une étiquette au pixel courant selon ces règles avec N = le nombre d'étiquettes présentes dans le voisinage :

1. N = 0 : création d'une nouvelle étiquette.
2. N = 1 : le pixel courant reçoit la valeur de cette étiquette.
3. N > 1 : le pixel courant reçoit la valeur de la plus petite étiquette dans son voisinage

Lorsque l'on arrive à la fin de l'image on remonte l'image en affectant au pixel courant la plus petite étiquette entre la sienne et celle de ses voisins. On continue jusqu'à ce qu'aucune étiquette ne soit modifiée lors de la descente ou de la remontée de l'image.

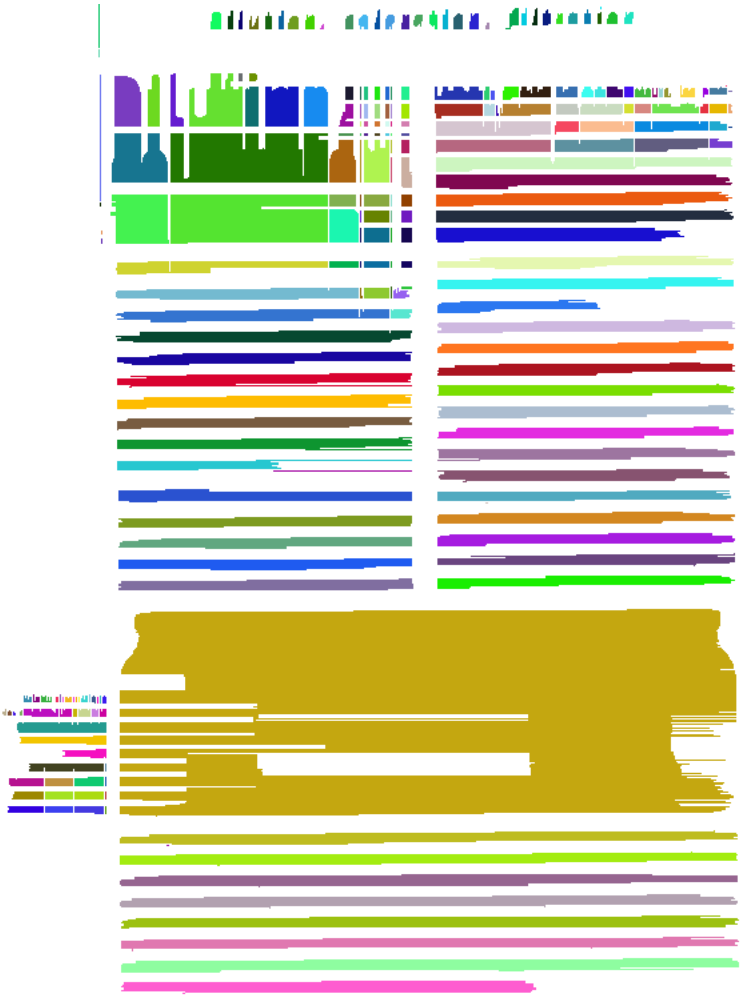

FIGURE 16 – Image après application de l'algorithme

6 Réseau de neurones

6.1 Introduction

Pour cette seconde soutenance, nous avons mis en place un réseau de neurones capable d'apprendre et de détecter la valeur d'un caractère.
En effet, l'implémentation du réseau suit la description présente sur ce site web : `http://neuralnetworksanddeeplearning.com/`, il est capable d'apprendre et d'évoluer en fonction des résultats qu'il fournit.
Ce réseau est modulaire et permet de gérer plusieurs formes de données : c'est-à-dire, permet de détecter, en fonction de la forme du réseau, une image pour détecter un visage, une forme et donc une lettre, ou alors, deux entrées binaires pour effectuer une fonction.
Nous avons fait en sorte que le réseau ait une forme facilement interchangeable, pour pouvoir passer facilement entre les différentes fonctions du réseau.

Nous avons fixé le réseau pour qu'il n'ait qu'une seule couche cachée de 100 neurones. Il possède 900 neurones d'input conformément à la taille de chaque lettre après les avoir découpées, et autant des neurones de sorties qu'il y a de caractères différents à apprendre. Pour l'ocr, nous avons choisis de ne reconnaître que les 64 caractères présents sur la figure 16 pour plus de simplicité.

6.2 L'apprentissage

Par rapport à la première version, nous avions focalisé le réseau sur un calcul de XOR, nous avons finalisé le réseau et il est maintenant capable de reconnaître les caractères.

FIGURE 17 – Image pour l'apprentissage du réseau de neurones.

Pour apprendre au réseau de neurones la réponse à toutes les questions (aka toutes les images) auxquelles il va être confronté, nous allons donc faire en sorte qu'il puisse "apprendre".
Pour cela, on va donc découper l'image ci dessus lettre par lettre et en construire une liste ordonnée. Grâce à la structure Bashint présente ci-dessous.

```
typedef struct Bashint Bashint;
struct Bashint{
  double *input;
  double  res;
};
```

Les images, qui ont au préalablement été découpées et remises a la bonne taille, représentent une liste de 900 bits qui correspondent aux 900 pixels binarisés de la lettre. Suivant l'ordre dans lequel la lettre apparaît dans le document, on lui assigne une valeur (a == 1, b ==2, z == 26, A == 27, "," == 64) que l'on stocke dans le champ res de la structure Bashint.
Le réseau va donc pouvoir commencer à apprendre.
L'apprentissage suit la description du livre écrit par *Michael Nielsen* et présent sur le site `http://neuralnetworksanddeeplearning.com/`. Nous allons donc faire apprendre un grand nombre de fois au réseau ce qu'il doit trouver et donc il va finir par reconnaître chacun des caractères présents sur l'image de test.
processus utilise le *Sum Squared Error*, pour cela le réseau va calculer la sortie probable de chaque caractère, et la la fin de chaque *epoch*, il va calculer l'écart entre la réponse du réseau et la vraie valeur de sortie (appelée *Error*) et il va se recalibrer

en fonction de cette *Error*.

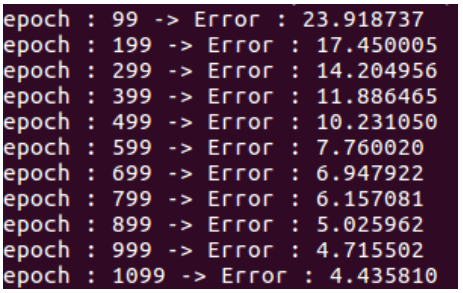

FIGURE 18 – Premiers *Epoch*

Nous pouvons observer une amélioration nette entre les 100 premiers *Epoch* et les 200 qui suivent

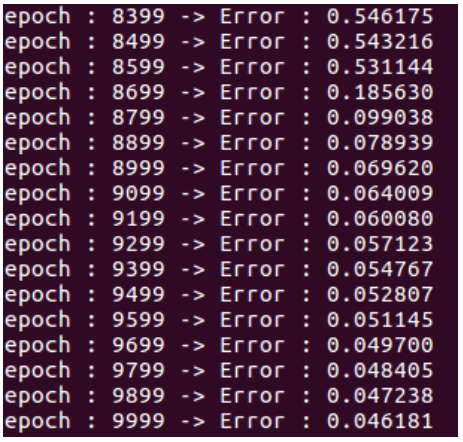

FIGURE 19 – Premiers *Epoch*

A la fin de l'entraînement, l'*Error* est suffisamment petite pour pouvoir avoir une réponse assez précise.

Nous pouvons donc commencer a tester :

Pour cela nous allons donner l'image qui a servit pour tester et toutes les lettres. L'affichage correspond a chaque lettre et la réponse : une liste dont la place de valeur maximum correspond à la valeur de la lettre (a == 1, b ==2, z == 26, A == 27, "," == 64)

FIGURE 20 – Résultat pour la lettre "a"

FIGURE 21 – Résultat pour la lettre "c"

6.3 L'utilisation

Il suffit donc d'utiliser la *struct Bashint*, stocker la liste de pixels en binaire dans *input*, et de ne rien mettre dans *res* pour avoir le résultat.

Puis chaque lettre est renvoyé au traitement de l'image qui va l'associer pour avoir toutes les lettres dans le bon ordre et avec les bon espacement.

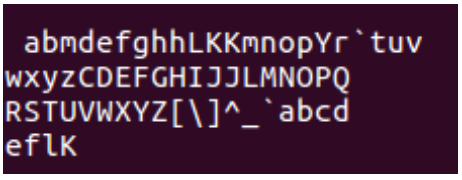

FIGURE 22 – Résultat des algorithmes l'image d'apprentissage du réseau de neurones

Pour finir, à chaque fois que nous allons améliorer le réseau, nous allons le sauvegarder dans un fichier *"neuralNetwork.nr"*. Ce fichier contient toutes les information dons le réseau a besoin pour pouvoir reconnaître les textes.
En effet, pour que l'utilisation de ce réseau soit plus rapide pour la reconnaissance des caractères, nous n'allons pas créer un réseau pour chaque textes sachant qu'il faut l'entraîner plusieurs fois avant d'avoir un résultat convenable. Nous chargeons donc le réseau à partir de ce fichier mais s'il n'existe pas, nous allons en le créer, l'entraîner puis le sauvegarder pour l'utiliser une prochaine fois.

7 Conclusion

Lors de la première soutenance nous avions conclus que nous étions content du résultat d'un point de vue général. En effet, nous avons réussi à produire un début d'OCR opérationnel respectant le cahier des charges donné.

Ce projet nous a appris beaucoup sur le C mais aussi sur les réseaux de neurones ou bien encore sur les algorithmes de détection des caractères.

Nous espérons que cela nous soit utile pour nos prochains projet à Epita voire quand nous serons dans le monde professionnel.

Néanmoins après cette dernière date de rendu nous ne sommes pas aussi satisfait comme nous n'avons pas tenu l'entièreté de notre cahier des charges. Notamment, certains problèmes au niveau de l'implémentation de l'interface graphique ont eu comme conséquences, dans les derniers instants de rendu, d'enlever les possibilités de démonstrations de certaines fonctionnalités.

Table des figures