Reconnaissance Optique de Caractères (ROC)

Rapport de soutenance 1

Auteurs :
Romain Hermary
Louis Guo
Marile Lin
Valentin Roland

21 septembre 2018

FIGURE 1 – Logo de notre groupe fait par Marc Huang

RECONNAISSANCE OPTIQUE DE CARACTÈRES (ROC)

Date de la création du document : 21 septembre 2018
Date de la dernière modification du document : 21 septembre 2018
Auteur du document : Los Espadas

Adresse de l'établissement dans lequel se fait la réalisation du projet :
Epita
66 rue Guy Môquet
94800 Villejuif
Date de fin de projet : décembre 2018

Table des matières

1 Introduction

La reconnaissance optique de caractère (de son abréviation anglaise OCR pour Optical Character Recognition), est un système comme l'indique son nom, de détecter à partir d'une image standard un texte (soit non éditable) pour pouvoir le sauvegarder et ainsi le modifier. Ce projet est composé en deux grandes parties : les éventuels traitements d'images ainsi que d'autres opérations sur ces dernières, puis la reconnaissance des caractères par un réseau de neurones. L'objectif du projet est de pouvoir coder un algorithme en langage C99 qui aura à la fin une utilisation facilitée grâce à une interface graphique. Ce projet est réalisé dans le domaine des études du cycle préparatoire de deuxième année à Epita.

Pour cette raison, nous nous sommes regroupés en un groupe de quatre élèves essayant de mener à bien ce projet. Notre groupe a pour nom Los Espadas (voir le logo figure 1), terme espagnol qui signifient "Les Épées", mais étant une référence au manga *Bleach* ; ne se connaissant pas tous, nous avons émis l'idée de trouver un nom de groupe en rapport avec quelque chose que nous connaissions tous, que nous aimions tous, ce qui a résulté en cette appellation. Les membres sont : Romain HERMARY, chef de groupe, Louis GUO, Marile LIN et Valentin ROLAND, issus de la classe SPE C2 promotion 2022.

Dans ce premier rapport de soutenance, on vous présentera tout d'abord la répartition des tâches ainsi que les avancements sur ce projet, pour ensuite également détailler les parties réalisées, avec nos difficultés éventuelles lors de ces étapes, et on terminera par une brève mais pas moins efficace conclusion.

2 Présentations personnelles

Romain "Kazoryah" Hermary

Je viens de Terminale S-SVT, spécialité Mathématiques. J'ai toujours été intéressé par l'informatique, savoir comment les choses peuvent fonctionner avec des matériaux infimes et des lignes de codes. Ainsi, je me suis naturellement tourné vers ce domaine qui restera prometteur pendant longtemps. Je ne me destinais pas à entrer à l'EPITA en post-bac, mais l'intégrer par concours à la suite d'une CPGE me paraissait trop incertain. De plus, j'avais vraiment envie de découvrir le codage informatique dans les conditions que propose l'école. C'est d'ailleurs grâce à ce choix que j'ai la chance de pouvoir partir en Afrique du Sud pour le prochain semestre. Mon projet de l'année dernière était intitulé **Gamma**, un plateformer sur le thème de la musique et des temps médiévaux. Ce deuxième projet de groupe est une nouvelle occasion de progresser et de montrer nos capacités. Faire apprendre quelque chose à quelqu'un n'est pas toujours facile, alors à un ordinateur, cela ne peut être que plus intéréressant. De plus, il est important de connaître le langage C, et ce projet ne peut qu'y être bénéfique, c'est notamment pour cela, et pour satisfaire ma curiosité, que je compte m'investir dans la réalisation de ce projet.

Louis Guo

Je suis étudiant à l'EPITA en 2e année du cycle préparatoire. Ayant obtenu mon bac scientifique en 2016, j'ai fait deux années de classe préparatoires aux grandes écoles durant lesquels j'ai beaucoup réfléchi et me suis interrogé énormément à propos de mon avenir. Je me suis finalement décidé à rejoindre EPITA, attiré par les perspectives d'emplois que proposent le secteur de l'informatique. Mais n'étant que novice dans ce domaine, et étant donné le peu d'informatique vu précédemment au cours de mes études, un retard énorme s'établit d'ores et déjà par rapport aux autres élèves, qui ont suivit l'enseignement d'EPITA durant

un an. Conscient du fossé me séparant des autres, je suis certain que le projet OCR m'apportera énormément de bénéfices non seulement sur l'aspect pratique de l'informatique puisqu'il sera mon premier projet, mais aussi sur l'apprentissage du travail collectif indispensable au futur travail d'ingénieur. Je remercie mes collaborateurs de m'avoir accepté dans le groupe malgré mon manque d'expertise dans le domaine de la programmation. Conscient de l'handicap que j'apporte à mon groupe, je vais tout faire pour ne pas les ralentir et aider au maximum mes camarades dans la réussite de notre projet OCR.

Marile "Low_battery" Lin

Etant actuellement étudiante à Epita en deuxième année du cycle préparatoire, nous devons réaliser un projet informatique avec notre classe, par groupe de quatre. Pour me présenter simplement, je suis très dynamique et apprécie énormément le travail en groupe (dans le cas où tout le monde participe). J'ai effectué un bac S-SVT, de spécialité mathématiques (période où j'ai appris ce qu'était une matrice dont le concept sera énormément utilisée pour ce porjet). L'informatique est un domaine vaste, dans lequel la création de certains logiciels ou applications pourrait attiser la curiosité de n'importe qui. Les énigmes ainsi que les difficultés rencontrées pour parvenir à une concrétisation de nos connaissances, passe par l'application de celles-ci soit la réalisation d'un projet informatique. L'OCR (Optical Caracter Recognition) est un outil qui, personnellement, servira tout au long de la durée de ma scolarité, voire même après. Certes, des logiciels plus performant existent déjà sur le net mais ce n'est pas le même ressenti que d'utiliser un outil fabriqué de nos propres mains (ou plutôt de nos propres doigts). Je sais d'avance que ce projet m'apportera beaucoup en autonomie, en organisation ainsi qu'une amélioration des connaissances du langage informatique C99. J'ai donc hâte de voir l'aboutissement de ce projet.

Valentin "Ysoof" Roland

Je suis actuellement étudiant en deuxième année du cycle préparatoire de l'EPITA ayant un bac S-SVT spécialité mathématiques. J'ai rejoint cette école car j'étais intéressé par l'informatique et les nouvelles technologies. Différentes majeures qui sont proposées par l'école m'intéressent et je n'ai pas encore un choix fixé sur celle que je choisirai mais l'une d'elle est la majeure de sciences cognitives et informatiques avancée. C'est en parti pour cela que ce projet de S3 est intéressant car il me permet de découvrir et d'acquérir des bases en deep learning en travaillant sur le réseau de neurones de notre OCR. C'est un domaine qui me parait intéressant et futuriste car permettre à une machine d'apprendre de ses erreurs et de les corriger permet en quelque sorte d'imiter un mécanisme important du cerveau humain et peut donc permettre de créer une multitude d'applications plus formidables les unes que les autres. Ce projet me permet par ailleurs de me former à l'utilisation du langage C et donc à l'utilisation d'un langage bas niveau qui peut être utile pour la sécurité informatique qui est une autre majeure qui pourrait m'intéresser. C'est pourquoi je pense que ce projet sera très instructif et m'apportera de nombreuses notions qui seront indispensables à la continuité de ma formation. Il me tarde donc de mener ce projet à bien et de voir le résultat.

3 Répartition des charges

Dans le tableau ci-dessous, vous pourrez donc voir la répartition des charges de travail, qui étaient à effectuer pour cette première soutenance, au sein de notre groupe :

Tâche	Romain	Louis	Marile	Valentin
Initialisation		⊕	+	
Detection			⊕	
Réseau de neurone	⊕			+
Gestion du réseau de neurone	+			⊕

Légende :

⊕ : activité principale
\+ : activité secondaire

4 Avancement du projet

4.1 Avancement général

Le tableau que vous verrez ci-dessous cette fois-ci représente l'avancement général de notre projet. Il comporte les étapes terminées pour cette première soutenance, ainsi que celles qui ont été commencées et qui seront également à finir. La liste de ces étapes a été repris du cahier des charges initialement donné.

Tâche	Avancement prévu	Avancement réel
Chargement de l'image	100%	100%
Changement des couleurs	90%	100%
Détection et découpages des blocs	100%	100%
Fonction XOR	100%	100%
Chargement des poids du réseau de neurones	30%	30%
Jeu d'images pour l'apprentissage		
Le réseau de neurones complet et fonctionnel	30%	40%
Reconstruction du texte et sauvegarde		
Une interface graphique		
Pré-traitement (bonus)		

4.2 Avancements individuels

Dans cette partie, on présentera la réalisation individuelle de chacune de nos parties, en faisant part de nos difficultés ainsi que de ce que l'on a pu y apprendre.

4.2.1 Romain

Matrices, Forward Propagation, Structure générale

Pour cette première soutenance, je me suis intéressé au réseau de neurones à proprement parler ; je devais comprendre comment fonctionnait un tel réseau ainsi que la façon de l'implémenter. Après avoir lu plusieurs documents concernant le sujet, notamment le livre donné en lien dans le cahier rose, la première barrière était très logiquement le langage C. Comme la plupart des personnes, je n'y connaissais rien.

Sachant que pour notre réseau nous allions utiliser des matrices et qu'il était plus avantageux d'utiliser des listes et de les parcourir comme des matrices, la première chose que j´ai faite est donc d´implémenter une structure "Matrix", qui contient le pointeur vers la liste stockant les valeurs de la matrice, et sa taille, ainsi que plusieurs fonctions principales : initialisation d´une "Matrix" de taille donnée, ainsi que retourner ou alors changer une valeur à une adresse. Ces fonctions sont au coeur de l'implémentation de notre réseau.

```c
// Matrix object declaration
struct Matrix
{
        // Size of the matrix
        int rows;
        int columns;

        // Pointer to the matrix
        double *mat;
};

// Function to change a value in a matrix
void ChangeMatrix(struct Matrix matrix, int x, int y, float val);

// Function to navigate in the matrix; return matrix(x, y)
double NavMatrix(struct Matrix matrix, int x, int y);
```

FIGURE 2 – Structure "Matrix"

Ensuite, je me suis occupé de la création de notre structure "Neural Network" et de son initialisation. Nous avons choisi un réseau avec deux neurones dans sa couche cachée pour la porte XOR. À défaut de quelques nombres qui caractérisent les calculs pouvant (normalement, pour un réseau de cette taille) être stockées dans des variables, notre réseau est principalement divisé en plusieurs matrices : deux pour les poids (une entre l'entrée et la couche cachées, et une autre entre la couche cachée et la sortie), deux pour les biais (une pour ceux des neurones de la couche cachée, une autre pour les neurones en entrées), sans oublier toutes celles nécessaires aux calculs des dérivés, des gradients, et des variations des poids et des biais. Chacune d'elles est initialisée en fonction du nombre d´entrées, du nombre de neurones dans la couche cachée et de neurones en sortie, ce qui nous permet d'avoir seulement trois variables à changer si l'on veut changer la taille du réseau.

Après l´initialisation, il faut parcourir une première fois le réseau pour avoir un nombre en sortie ; la "Forward Propagation" est la fonction principale que j'ai choisi d'implémenter, Valentin faisant majoritairement l'autre dont il parlera dans son propre récapitulatif. Cette fonction était plutôt simple, puisqu'il s'agit de multiplications et additions entre matrices de poids et biais, ainsi que l'application de la fonction sigmoïde.

Pour mettre à jour les poids et les biais, après le calcul entre autres des gradients, nous avons conçu avec Valentin une fonction qui mettait à jour les valeurs dans les matrices. Cependant, nous nous sommes aperçus que les valeurs qui n'étaient pas stockées dans des structures "Matrix" avaient des problèmes de pointeurs. En effet, les valeurs étaient bien changées dans les fonctions mêmes, mais pas en dehors. Après quelques tentatives infructueuses, et ne voulant pas prendre le risque de devoir réorganiser tout le code à cause des pointeurs, en ne sachant pas vraiment les maîtriser, nous avons décidé pour cette soutenance, d'utiliser des matrices de taille (1,1) pour stocker des doubles qui devaient être changés par nos fonctions.

Enfin, je me suis occupé du peaufinage de la fonction "main" faite part Marile : pour charger une image lors de la première soutenance il faut simplement entrer l'adresse de stockage de l'image seule, et pour afficher les sorties du réseau de neurones pour la porte XOR, il faut

mettre "display xor" [1] en argument ; tout autre argument renvoie une erreur.

FIGURE 3 – "./main" uses

1. Voir 4.2.4 pour l'affichage de "display xor"

4.2.2 Louis

Mon rôle dans cette soutenance était de faire à la base la segmentation. En effet, je devais découper l'image sous un format quelconque mais de préférence bitmap (bmp) en bloc de pixels qui permettrait au réseau de neurones de détecter les caractères. Pour ce faire, j'ai eu plusieurs idées. Dans un premier temps transformer l'image binarisée (voir partie de Marile). Puis parcourir lignes par lignes jusqu'à détecter une ligne contenant des pixels noirs pour connaître les coordonnées des pixels du haut de la première ligne. Ensuite, les coordonnées du bas de la première ligne sont données par la coordonnée de la première ligne. J'ai également pensé à faire une moyenne pour éliminer les impuretés. J'ai eu de nombreuses idées pour répondre au problème de la segmentation mais l'implémentation en C de ce dernier dépassait mes compétences. Après des résultats peu concluants, Marile prit le relais de la segmentation.

Pour cette première soutenance je vais donc présenter le chargement d'image. L'image que nous manipulerons aura la majorité un format bitmap (donc bmp) car à la différence des autres images (jpg ou png), les fichiers bmp ou bitmap ne sont pas compressés ce qui veut dire que tous les bits de l'image sont présents et donc la qualité est meilleure. En terme de taille de stockage, ces derniers sont généralement plus volumineux.

Afin de charger notre image, nous utilisons la bibliothèque SDL2 (Simple Directmedia Layer). N'étant pas une bibliothèque standard, nous avons dû la télécharger depuis internet pour pouvoir travailler avec. Vous avez ci-dessous l'exemple de notre fonction permettant d'importer l'image avec une "path", soit le chemin de celui-ci.

```
SDL_Surface *load_image(char *path);
```

L'image est affichée grâce à la fonction display qui également dépend de la bibliothèque SDL et s'adapte en fonction des dimensions de l'image :

```
SDL_Window* display_img(SDL_Surface* picture);
```

Pour appliquer ces deux fonctions et en voir le résultat, il faudra passer en paramètre du terminal la path correspondant à l'image (bmp, png

ou jpg) pour pouvoir appeler les fonctions load_image et display_img. Voici un exemple du résultat si la path donnée est correcte :

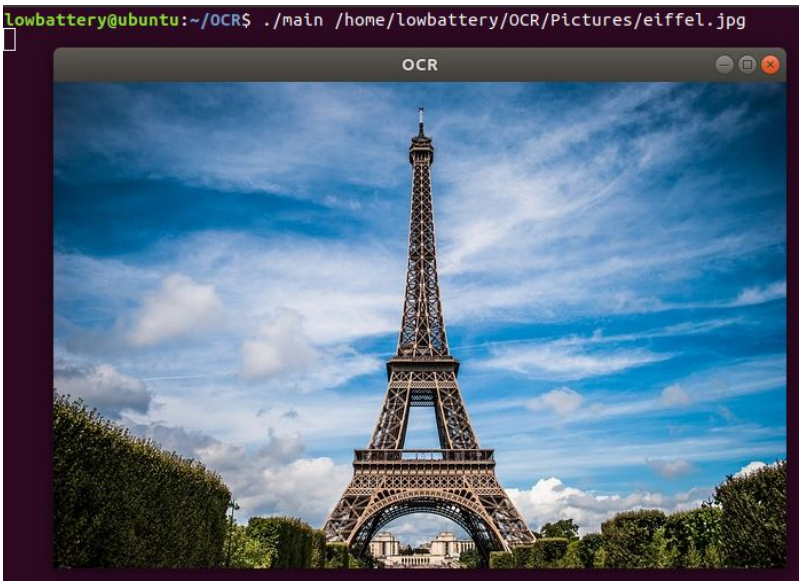

FIGURE 4 – Importation d'image et affichage

Mais voici ce que cela affiche si jamais la path est incorrecte ou que le fichier cherché n'est pas à l'emplacement donné :

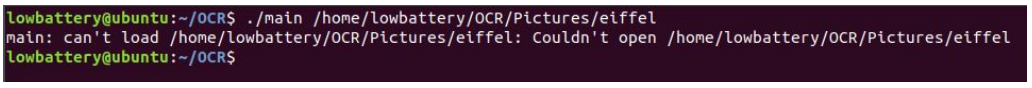

FIGURE 5 – Erreur de l'importation de l'image

Cependant il n'y avait pas qu'une seule image à afficher, par exemple pour afficher le résultat de l'image binarisée, il fallait "update" l'écran avec cette fonction :

```
SDL_UpdateWindowSurface(SDL_Surface* screen);
```

Mais il fallait penser aussi à donner une indication à l'ordinateur pour lui demander d'exécuter le rafraîchissement d'image. Ainsi intervient la fonction :

```
void wait_for_keypressed();
```

Cette fonction permet d'attendre que l'utilisateur appuie sur n'importe quelle touche de son clavier pour passer à l'exécution suivante.

Cela pourrait permettre l'affichage d'une autre image (binarisée, découpée ou autres) ou alors d'éteindre la fenêtre en libérant la mémoire avec la fonction :

```
void SDL_FreeSurface(SDL_Surface *screen);
```

Nous sommes conscients que l'interface que nous utilisons grâce à la SDL, n'est pas la plus optimale (dû à son aspect peu dynamique) et pour cette raison on commencera à s'intéresser à la bibliothèques GTK+ (écrite en C) qui permet justement de créer des interfaces graphiques pour des applications.

4.2.3 Marile

Binarisation d'une image

La binarisation d'une image, consiste simplement à une diversité de couleur limitée sur l'image. On a le choix entre deux couleurs : le blanc ou le noir. Avant d'en dire plus sur la manière dont on s'y est pris il est important de définir au préalable ce qu'est un pixel.
Le pixel est en réalité un petit carré composé de trois composantes primaires par synthèse additive : RGB (Red Green Blue) et respectivement en français le rouge, le vert et le bleu. La valeur de ces trois couleurs peut varier de 0 à 255 inclus (0 étant la couleur noire et 255 la couleur blanche).
Pour cette raison les types utilisés pour chaque composante était un Uint8 :

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

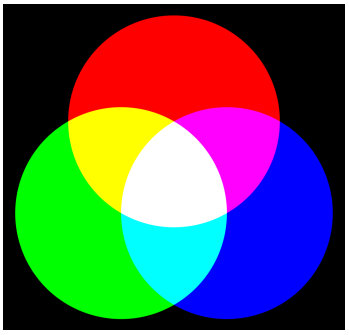

FIGURE 6 – Synthèse additive

Selon la plus grande présence de rouge, vert ou de bleu, la couleur entière du pixel sera plus orientée vers celles-ci. Prenons à présent 50% de ces trois composantes (soit une moyenne de 128 pour chacune de celles-ci), la couleur finale sera le gris. On prendra également cette référence comme seuil pour pouvoir déterminer si oui ou non, on change le pixel

soit en noir soit en blanc.
On rappelle donc que la binarisation est le choix entre le blanc ou le noir. Donnons un exemple :

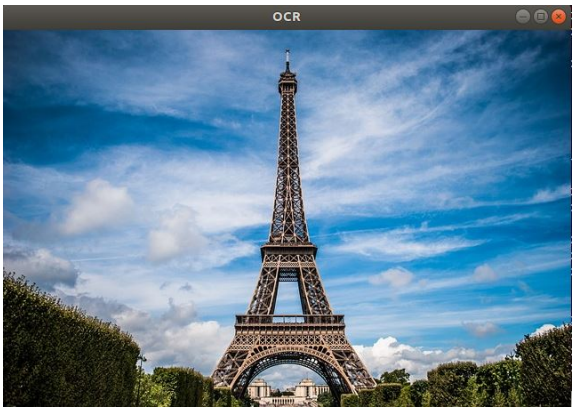

FIGURE 7 – Image originale

Après parcourt de toute l'image qui est en réalité une matrice à deux dimensions, on vérifie la moyenne de gris à chaque pixel rencontré. C'est à dire que si la moyenne de celui-ci dépassait 128 alors sa couleur tendait plus vers 255 (couleur blanche), et donc on doit changer les trois composantes de manières à ce que le pixel soit entièrement blanc avec la valeur du rouge, vert et bleu égales toutes les trois à 255. Et vice-versa si la moyenne de gris du pixel était inférieure à 128 (plus proche du noir) alors on mettrait les trois composantes primaires à 0. Voici le résultat de la figure 5 après binarisation :

FIGURE 8 – Image après binarisation

La binarisation peut se faire de pleins de manières différentes avec éventuellement d'autres désavantages à prendre en compte (comme la

mauvaise qualité de l'image), et pour cette raison il faudrait effectuer au préalable différents traitements d'images. Cependant nous avons préféré nous concentrer sur la fonctionnalité du début avant de se lancer dans les améliorations d'images.

Détection et découpage en blocs

Après la phase de l'initialisation, c'est à dire le chargement d'une image sous format quelconque ainsi que la binarisation de l'image, nous nous retrouvons avec une image de couleurs binaires (soit noir, soit blanc). En effet, comme expliquée dans l'étape précédente les pixels possèdent trois bases de couleurs : le rouge, le vert et le bleu (RGB = Red Green Blue). Selon la valeur donnée à ces trois couleurs de bases, cela peut faire changer la couleur du pixel entière. Dans notre cas, nous nous concentrerons que sur les pixels blancs, qui ont pour valeur RGB = 255 ou noir avec RGB = 0 (expliqué précédemment). Par conséquent, si nous nous basons sur une image qui possède du texte sur un fond de couleur clair (voir figure 9 ci-dessous), nous pouvons remarquer qu'après binarisation (voir image suivante, soit la figure 10) que les textes ressortent noirs et le fond est blanc.

FIGURE 9 – Image originale

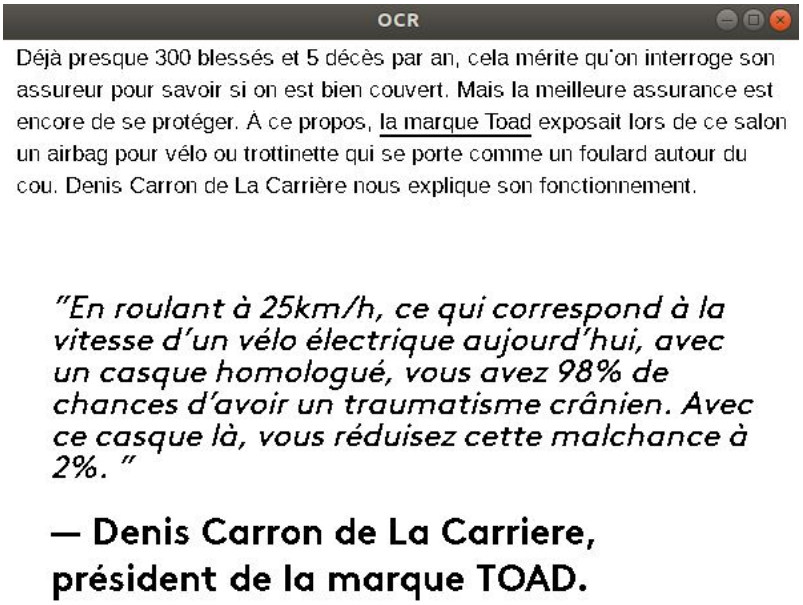

FIGURE 10 – Image binarisée avant le découpage

Pour pouvoir détecter à présent le(s) bloc(s) de texte, il faudra pour cela parcourir toute notre image sous forme de matrice. En effet, nous rappelons qu'une image reste une matrice à deux dimensions composées de pleins de pixels.
Mais tout d'abord, nous allons diviser cette tâche en deux : la détection des lignes et la détection des caractères. C'est à dire d'essayer de détecter les lignes du texte, pour ensuite détecter les caractères à travers celui-ci. L'algorithme qui permet la réalisation de la première détection consiste à parcourir toute la matrice de l'image, en parcourant chaque colonne dans chaque ligne (picture étant ici une image binarisée donnée en paramètre).

```
for(int y = 0; y < picture -> h; y++)//lines
{
    for(int x = 0; x < picture -> w; x++)//columns
    {
        //code
    }
}
```

- **Détection des lignes** : A chaque fois que l'on rencontre un pixel noir, on supposera celui-ci comme étant le début d'un caractère et ainsi le début d'un mot ou d'une phrase. On décide alors de

tracer une ligne horizontale de couleur (autre que noir et blanc) sur la ligne avant la rencontre de ce pixel.
Tracer une ligne horizontale sur une image, consiste en réalité à parcourir toutes les colonnes d'une ligne de l'image pour en changer les valeurs basiques (soit le rouge, vert et bleu) de chaque pixel.
Le premier trait étant tracé, il faut à présent savoir quand en tracer un second permettant ainsi d'encadrer la ligne du texte. Puisque nous parcourons colonne par colonne sur chaque ligne, nous supposerons que si nous ne rencontrons aucun pixel noir sur ceux-ci, cela signifiera que nous venons de parcourir une ligne entièrement blanche et que la phrase du texte est finie. Il est donc temps de tracer le second trait de couleur correspondant. Pour mieux illustrer ces propos, voici ce que nous obtenons de l'image présenté ci-dessus après détection des lignes (les traits de couleurs sont ici en rose) :

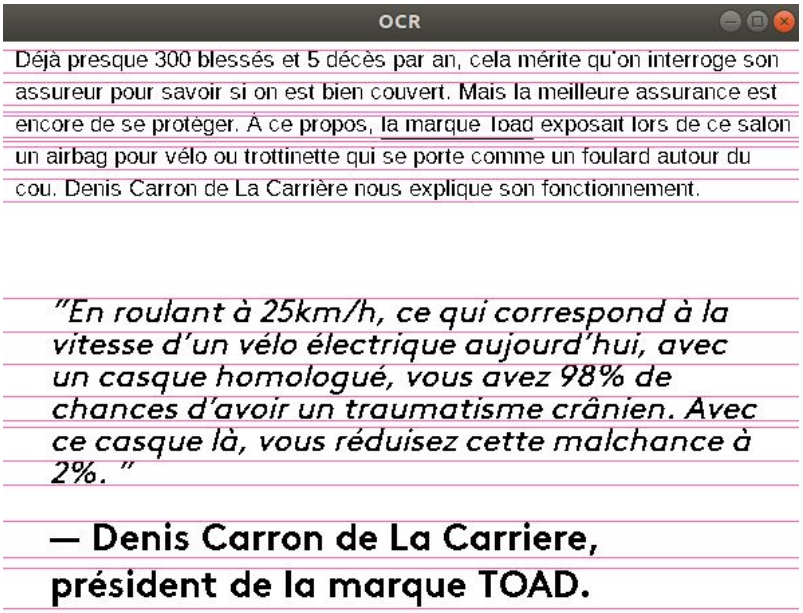

FIGURE 11 – Image binarisée avec le découpage des lignes

• **Détection des colonnes** : A présent que les lignes sont détectées, il faut maintenant s'occuper des caractères ou plus précisément des lettres de chaque mot. Il y avait ici plus de difficulté dû à la nécessité de parcourt de chaque colonne pour chaque lettre par ligne. Contrairement à la détection des lignes, il ne suffisait plus de juste se contenter du premier pixel noir rencontré sur une seule ligne mais bien de parcourir chaque colonne à chaque ligne, pour essayer de tracer un trait vertical avant chaque caractère et un trait après celui-ci (soit d'encadrer notre lettre). Comme précédemment, on utilisera ici un encadrement de couleur différent au noir, au blanc ainsi qu'au rose (simple préférence de notre part pour mieux distinguer notre découpage).
En premier lieu, j'ai pensé à écrire cette seconde fonction indépendamment de la première (celle qui consistait à la détection des lignes). Pour cela, il a fallu parcourir toutes les colonnes de chaque ligne (encore) et changer les couleurs avant et après chaque pixel noir rencontré. L'erreur d'avoir pensé ainsi m'a mené tout d'abord à un premier résultat peu ressemblant à ce que je voulais (voir figure suivante) :

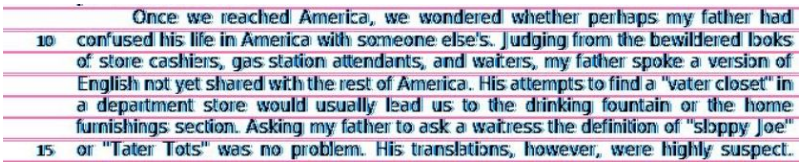

FIGURE 12 – Prmier problème du découpage de caractères

Cela m'avait entouré toutes les lettres et non pas encadré comme je le souhaitais. J'ai alors eu l'idée de me servir de la première fonction pour pouvoir faire le parcours de l'image efficacement. Pour cela il fallait que je renvoie en tant que paramètres dans ma deuxième fonction la ligne correspondant au début d'une ligne de texte (soit l'index du trait horizontal rose au-dessus de la ligne de texte) ainsi que sa fin (soit l'index du trait horizontal rose en dessous de la ligne de texte). Cette fonction traitait donc tous les mots d'une ligne de texte à la fois. Ayant l'index des traits horizontaux, je devais tracer cette fois-ci un trait vertical avant et après chaque caractère compris entre ces lignes. Je devais donc

me concentrer sur le parcours des colonnes et de la manière dont la fonction saura où se trouve la fin de mon caractère. Trouver le début de celui-ci est plus simple que l'inverse, car entre les deux index (entre les deux traits roses) il y aura forcément un caractère minimum et donc qu'il y aura forcément deux traits verticaux à tracer (un avant, un après).
Trêve de bavardage et passons plutôt à la démonstration, donc voici le résultat après correction de la deuxième fonction :

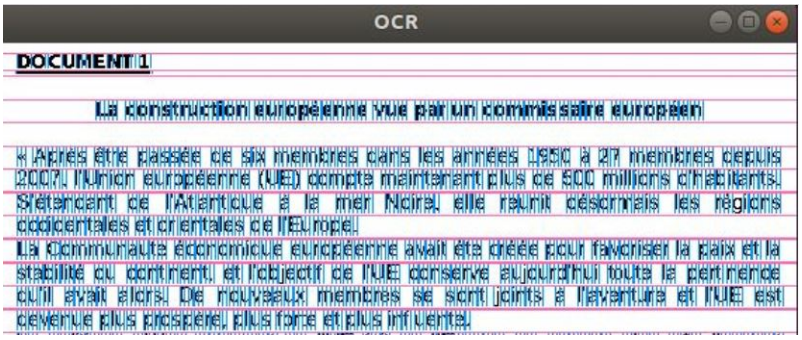

FIGURE 13 – Deuxième problème du découpage de caractères

On comprend donc que ce n'est pas encore le bon résultat et qu'il faut continuer à chercher. Les traits verticaux (bleus ici) sont bel et bien tracés mais ne correspondent aucunement à la séparation de caractères. Il y avait donc un problème uniquement au niveau de la détection de fin de caractères (puisque si l'on observe bien, les espaces blancs sont respectés). Etant si proche du but il était hors de questions d'abandonner en si bon chemin, et pour cette raison je remarquais qu'il fallait rajouter une condition nécessaire avant de tracer le trait censé marquer la fin du caractère, et en reprenant la figure 5, voici ce que l'on obtient avec le bon découpage :

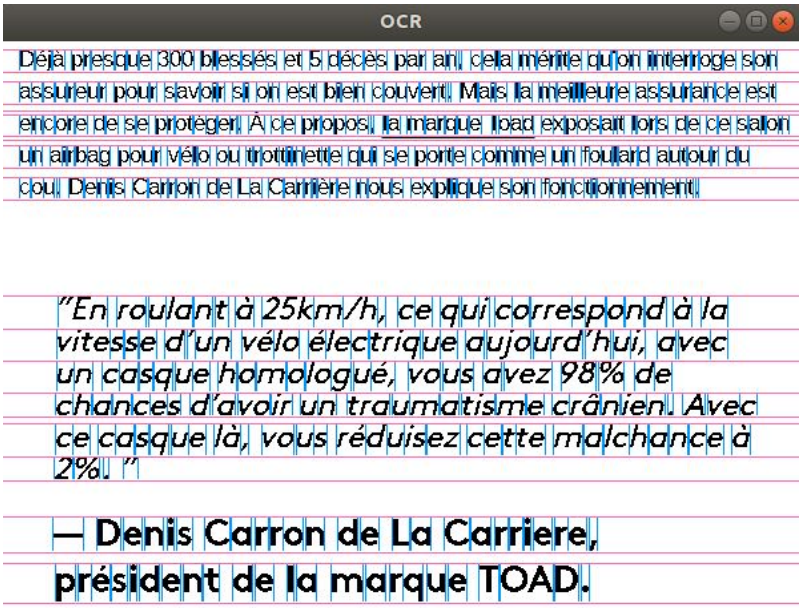

FIGURE 14 – Problèmes du découpage de caractères résolus

Nous avons donc terminé cette partie essentielle pour la suite, c'est à dire la reconnaissance des lettres à effectuer grâce au réseau de neurones.

4.2.4 Valentin

Backpropagation, Structure générale

Pour cette première soutenance j'ai travaillé avec Romain à l'élaboration du réseau de neurones capable d'apprendre la porte XOR. Nous nous sommes réparti les tâches, Romain s'est occupé de l'implémentation de notre classe matrice ainsi que du forward propagation tandis que je me suis occupé d'implémenter la backpropagation. Ces parties étant intimement liées, nous avons dû nous mettre d'accord sur l'initialisation des différentes matrices afin qu'elles puissent se correspondre mutuellement entre le feedforward et la backpropagation. Pour ce qui est de la backpropagation, il a tout d'abord fallu que je comprenne le principe de fonctionnement. Celle-ci avait pour but de traverser le réseau de neurone dans le sens inverse afin de mettre à jour les poids ainsi que les biais utilisés dans les calculs du feedforward pour se rapprocher de la valeur qui est attendue. Nous devions donc commencer par calculer cet écart entre la valeur obtenue à la sortie de notre réseau de neurones et la valeur que nous aurions normalement dû trouver selon l'entrée initialement choisie. Nous avons ensuite dû implémenter les formules de dérivées partielles ainsi que de sigmoid qui est la fonction d'activation afin de pouvoir calculer la mise à jour de nos matrices de poids et de biais.

Après avoir fini l'implémentation de ces différentes formules formant notre fonction backpropagation, nous avons créé une fonction test appelant nos fonctions feedforward et backpropagation en boucle sur les 4 patterns pour un nombre d'itérations appelées époques très élevé afin de tester si la convergence de notre résultat fonctionnait. Nous avons alors fait face à différents dysfonctionnements. Premièrement, notre fonction convergeait vers environ 0.6 quel que soit les entrées, or ceci ne correspondait en rien à ce à quoi on s'attendait. Nous avons donc avec Romain commencé à déboguer tous les deux afin de trouver d'où venait le problème. Nous avons tout d'abord vérifié chacune de nos formules afin de vérifier si l'erreur ne venait pas directement de là. Nous y avons trouvé 1 ou 2 erreurs et nous sommes donc dit que cela venait sûrement de là.

Après avoir de nouveau essayé notre fonction, le même bug persistait et nous nous sommes alors dit que l'erreur venait donc d'avant cela. Nous avons alors affiché les modifications appliquées à chacune de nos variables et matrices après chaque formule pour vérifier si elles étaient bien modifiées. C'est alors que nous nous sommes rendu compte que les variables étaient bien modifiées à l'intérieur de la fonction mais que la modification ne se conservait pas à la sortie de celle-ci. Nous nous sommes alors dit que c'était donc un problème de pointeurs. Nous avons donc essayé de changer ces variables en pointeurs mais cela ne fonctionnait toujours pas. Ne comprenant pas ce que nous avions fait de mal avec les pointeurs nous avons alors trouvé une autre solution pour régler ce problème. La classe matrice que Romain avait implémenté gérait correctement les pointeurs et nous avons ainsi décidé comme Romain a pu le dire précédemment de stocker chacune de nos variables dans des matrices 1/1 afin que les modifications sur celles-ci soient prises en compte. Cela a eu pour effet immédiat de faire converger notre réseau vers 1 lorsque l'on choisissait les entrées 1/0 ou 0/1 et vers 0 lorsque l'on choisissait les entrées 1/1 ou 0/0. Nous avons ensuite réduit graduellement le nombre d'époques pour voir à partir de combien d'itérations la convergence dépassait la barre des 99%. Nous sommes donc arrivés à cette convergence pour un nombre d'époques de 43500 nous donnant ce résultat (un affichage se fait tous les 100 époques).

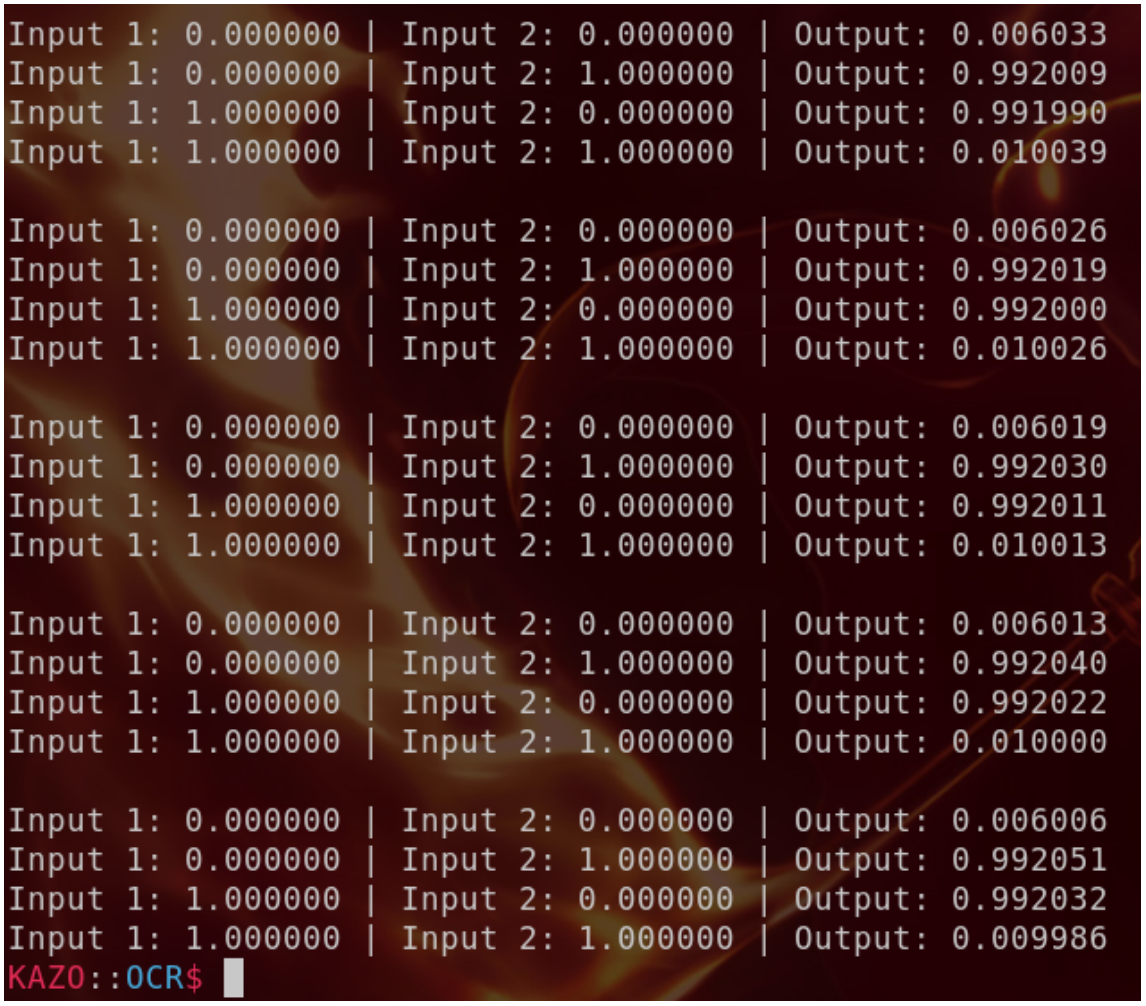

FIGURE 15 – "displaying the output of XOR gate

5 Avance, retard et prévisions

Nous avons pour cette première soutenance bien progressé, avancé, sur de nombreux points ; en effet, le code pour le chargement et la binarisation de l'image est complet, la segmentation de l'image se fait correctement : la détection des lignes et des lettres ne donnent pas d'erreurs et semble être prêt à donner des matrices de pixels bien formées.

Pour ce qu'il y est du réseau de neurones, il est fonctionnel et plutôt clair. Le code implémenté pour la porte XOR sera très utile, la plupart des fonctions pourront être réutilisées pour la deuxième soutenance et l'objectif final de notre projet, puisque les calculs ne changeront pas, seul le nombre d'entrées et de sorties variera. Néanmoins, il sera toujours possible d'essayer d'optimiser le nombre d'itérations nécessaires à l'apprentissage avec un taux de précision de 99%, en modifiant certains coefficients, valeurs fixes, voire le nombres de neurones cachées.

Tout ce qui concerne la gestion du texte, c'est à dire l'entrée de pixels représentants la lettre dans le réseau adapté ainsi que la reconstitution du texte, et l'interface graphique est à commencer. Le pré-traitement n'est pas encore dans nos objectifs puisqu'il est bonus, mais nous ne l'oublions pas et s'y intéresserons si l'avancement de notre projet nous le permet.

6 Conclusion

Ce projet est mené à bien par une équipe de quatre étudiants en informatique motivés et prêts à y consacrer le temps qu'il faut. Le projet se décompose en trois grandes parties : le traitement d'image, l'apprentissage par un réseau de neurones, et enfin l'exploitation de la sortie du réseau pour éditer le texte obtenu. Nous avons déjà réalisé le chargement de l'image et sa transformation en image binaire, la détection des lignes et des lettres dans le texte qu'elle contient, ainsi que l'implémentation d'un réseau de neurones permettant d'apprendre la porte logique XOR à l'ordinateur.
Le projet est sur la bonne voie, les bases de la suite sont déjà présentes et permettront de continuer de la même façon dont ce projet a commencé : avec de l'efficacité et sans encombre.