Rapport de première soutenance
octobre 2019

par
Nicolas INDJEIN, Marius HURBIN
William JOHNSON, Cédric ELAIN

PROJET S3 2019

Sommaire

1 Introduction **4**

2 Présentation des membres du groupe **4**

2.1 Nicolas INDJEIN 4

2.2 Marius HURBIN 5

2.3 William JOHNSON 5

2.4 Cédric ELAIN 5

3 Répartition des tâches **6**

4 Planning de réalisation **6**

5 Déroulement du travail **7**

5.1 Traitement de l'image 7

5.1.1 Soutenance 1 7

5.1.2 Soutenance 2 9

5.2 Segmentation 12

5.2.1 Soutenance 1 12

5.2.2 Soutenance 2 16

5.2.3 Mise à jour de la segmentation 16

5.2.4 Lien avec le réseaux de neurones 17

5.2.5 Difficultés rencontrées 20

5.3 Réseau de neurone 21

5.3.1 Soutenance 1 21

5.3.2 Soutenance 2 31

5.4 Interface utilisateur 36

5.4.1 Soutenance 1 36

5.4.2 Soutenance 2 36

6 Avis personnels **39**

6.1 Nicolas 39

6.2 Marius 39

6.3 William 40

6.4 Cédric 41

7 Conclusion **41**

1 Introduction

Ce projet est celui de la deuxième année à EPITA. D'une durée d'un semestre, il a pour but la création d'un OCR (Optical Character Recognition). Un OCR est un logiciel capable de reconnaître du texte dans une image et de le sauvegarder dans un fichier pouvant être exploité pour le traitement de texte et stocké sur un support exploitable par un système informatique. Un OCR peut être capable d'aller jusqu'à la recréation des éléments de l'image (texte, image hors texte, mise en page, tailles, polices,...) dans la même disposition dans un document de traitement de texte. Le travail a été décomposé en 4 grandes parties :

— **Traitement de l'image :** Cette partie consiste en la mise en niveau de gris puis en noir et blanc, la réduction du bruit de l'image et la binarisation de celle-ci.
— **Segmentation :** Cette partie consiste en la séparation des blocs de textes, des mots et des caractères pour conserver la mise en page de manière à pouvoir envoyer ces-derniers au réseau de neurone pour la reconnaissance de caractère.
— **Réseau de neurones :** Cette partie consiste en la réalisation d'un réseau de neurone capable d'apprendre à reconnaître les caractères alpha-numériques. Il devra ensuite être capable de reconnaître les caractères sans avoir besoin d'être entraîné à chaque lancement du logiciel.
— **Interface utilisateur :** Cette partie consiste en la réalisation d'une interface utilisateur (graphique et console) pour le logiciel, afin que celui-ci puisse être employé par un utilisateur.

2 Présentation des membres du groupe

2.1 Nicolas INDJEIN

Je m'appelle Nicolas. Parmi les différentes choses que j'aime il y a les mangas, le surf, la programmation et surtout comprendre comment les jeux vidéos fonctionnent. Cela peut aller de comprendre comme sont codées les mécaniques, jusqu'à comprendre comment les développeur utilisent la psychologie pour influer sur le comportement et les réactions du joueur pendant la partie. A mes yeux ce projet est un bon défi car nous le réalisons dans un langage que nous ne maîtrisons pas encore très bien contrairement au projet de S2, et car je m'occupe de la partie du réseau de neurone qui est un bon challenge et qui est très intéressante.

2.2 Marius HURBIN

Je m'appelle Marius. Si je devais citer quelques unes de des choses que j'aime je citerai d'abord les jeux de rôle, parce que c'est toujours bon de combattre du dragon. Ensuite l'escalade, parce qu'un peu de sport ne peut pas faire de mal. Mais aussi les jeux vidéos, parce qu'ils sont plutôt appréciables surtout quand ils font réfléchir. Et enfin la Corse, parce c'est quand même une sacre belle île ou la nature est belle et sauvage.

2.3 William JOHNSON

Bonjour, je me présente William Johnson, étudiant à Epita Lyon et venant d'Aix-en-Provence. Intéressé par les nouvelles technologies, et la programmation j'ai alors rejoins cette école afin d'essayer d'avoir mon diplôme. Après une première année réussi, j'ai donc continué le cursus en Spé. L'année de Sup a était très enrichissante et m'a plongé dans le monde de l'informatique. Mon premier projet était en sup, et consistait à faire un jeux-vidéo (WAMBA), avec mes camarades Adèle, et Anthony (petite pensé à Chris et Boris). Maintenant, nous sommes lancés dans un nouveau projet l'OCR, avec mes collègues Nicolas, Cédric et Marius. Ce projet m'intéresse d'avantage que celui de l'année dernière car il a une réelle utilité et il pourrait m'être utile plus tard dans la vie. Il permet aussi d'aborder en profondeur le langage C, tout nouveau pour moi, et une partie de ses concepts de programmation. Mon rôle pour cette première soutenance était la segmentation, c'est à dire le découpage en bloc de l'image, puis en caractère afin de faire parvenir chaque caractère sous forme de matrice au réseaux de neurones. A première vue, j'ai tout de suite eu une petite idée de comment j'allais procéder, mais ce n'était pas aussi simple que je ne l'imaginais.

2.4 Cédric ELAIN

Je m'appelle Cédric. J'ai pour principale occupation la musique ainsi que sa réalisation. En effet cette activité occupe la plus grande partie de mon temps. J'écris, je compose, j'enregistre et je mixe. L'informatique est depuis peu de temps une partie très important dans le domaine musical. Le bon maniement d'un ordinateur assure une bonne qualité du rendu sonore. C'est donc ma passion qui m'a guidé vers mon futur métier. Dans le cadre de la validation de la deuxième année du cycle préparatoire, notre groupe est en charge de la réalisation du projet "OCR". Ce projet ambitieux attise ma curiosité et me motive beaucoup à surpasser mes capacités.

3 Répartition des tâches

	Nicolas	Marius	William	Cédric
Traitement de l'image		X		
Segmentation		X	X	
Réseau neuronal	X			X
Interface utilisateur	X			X

4 Planning de réalisation

	Soutenance 1	Soutenance finale
Chargement des images	X	
Suppression des couleurs	X	
Détection & découpage en blocs, lignes et caractères	X	
Réseau neuronal Xor	X	
Sérialisation des poids	X	
Désérialisation des poids	/	X
Jeu d'images	/	X
Sauvegarde des poids dans un fichier	X	
Réseau de neurone		X
Reconstruction du texte		X
Sauvegarde		X
Interface utilisateur		X

X = élément fini / = élément commencé

Dans le cas où la réalisation de l'OCR se terminerai plus rapidement que prévu, il est possible que des éléments bonus tels qu'un pré-traitement ou la conservation de la mise en page soient réalisés.

5 Déroulement du travail

5.1 Traitement de l'image

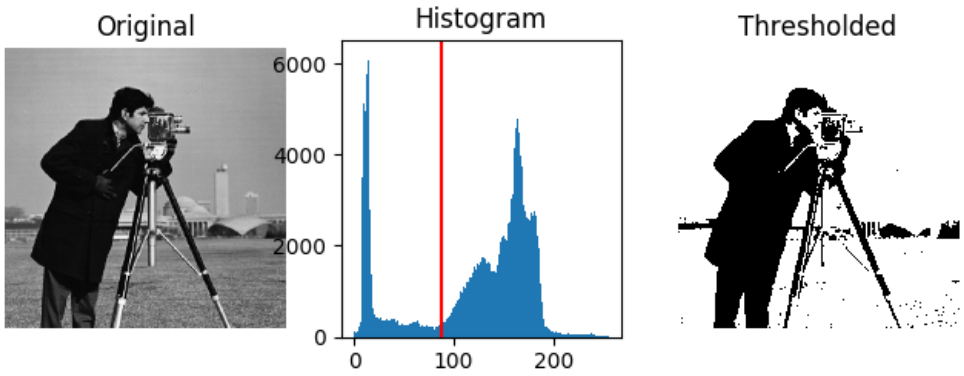
5.1.1 Soutenance 1

C'est Marius qui s'est occupé de cette partie. Le traitement de l'image commence avec le passage de l'image en nuances de gris. En effet, cette étape permet de ne plus avoir de pixels en format rouge-vert-bleu mais des pixels gris avec une valeur unique le représentant, sa nuance de gris. Pour ce script il a utilisé celui que nous avons réalisé dans le travail pratique numéro 3 (pourquoi s'en priver ?). Le reste du traitement de l'image a ensuite été son passage en noir et blanc et de trouver le moyen de différencier le texte du reste

5.1.1.1 Binarisation de l'image

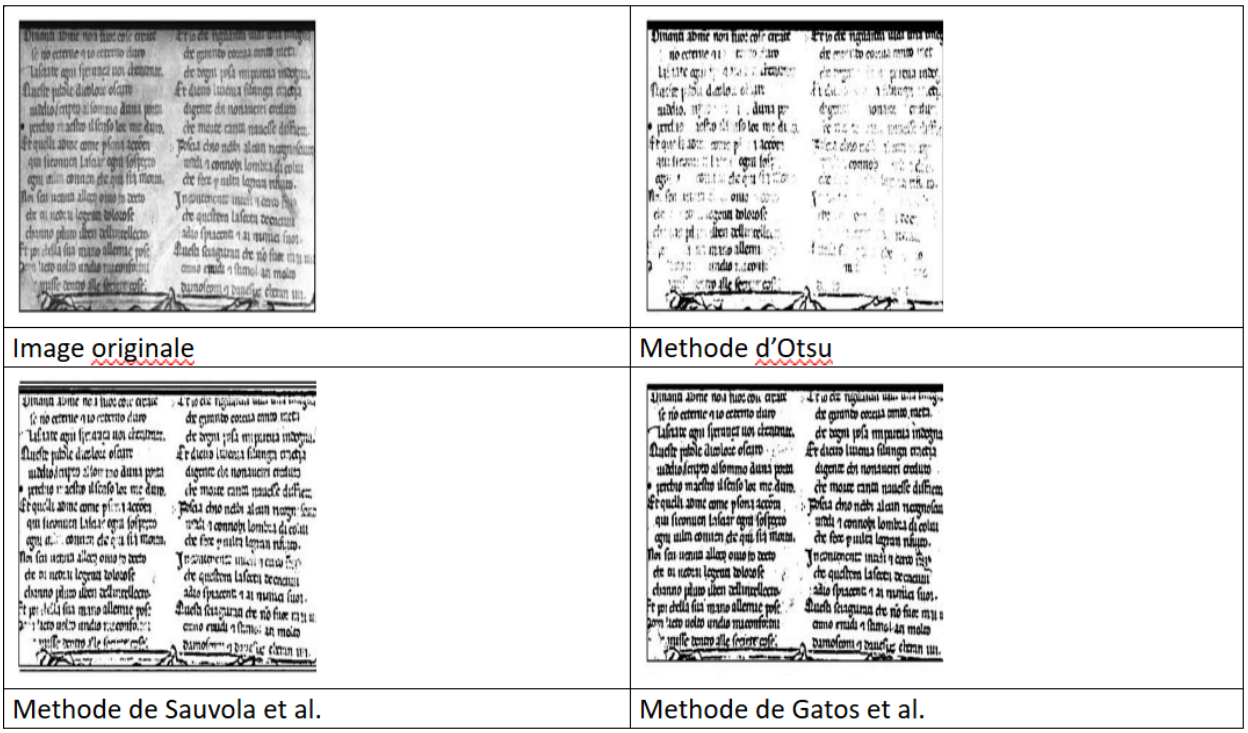
Pour la binarisation, nous avons décidé de commencer par faire un algorithme simple et fonctionnel, puis de compléter et/ou améliorer celui-ci plus tard. C'est à nouveau Marius qui c'est occupé de cette partie. Il a réalisé une fonction qui se charge d'appliquer un filtre passe bas sur une image selon le seuil que l'on lui donne. Dans un premier temps il ne faisait qu'une moyenne du gris de l'image. Cependant, cette option n'est pas très performante. C'est pourquoi Marius s'est intéressé a d'autres techniques pour trouver un seuil.

Après quelques recherches, il a choisi d'utiliser la méthode d'Otsu. Cette méthode a pour avantage d'être une méthode globale, c'est-à-dire qu'elle définie un unique seuil pour toute l'image. Elle est donc plus rapide a implémenter dans l'algorithme qu'il avait déjà fait. Cependant elle possède un défaut majeur : elle fonctionne très mal sur un document éclairé de manière non uniforme. En effet, la méthode d'Otsu consiste a trouver le seuil optimal pour séparer deux plans (le fond et le reste). Ainsi, s'il y a des ombres sur l'image cela fausse les calculs. Marius a donc fait d'autres recherches et a finalement trouvé l'algorithme de Gatos. Il a ensuite commencé a l'étudier pour tenter de compléter l'algorithme d'Otsu pour la binarisation.



Exemple de binarisation avec Otsu

L'algorithme de Gatos repose sur la complétion de deux autres algorithmes : la méthode Sauvola et le filtre de Wiener. La méthode de Sauvola est très puissante sur un document au fond blanc. A l'inverse de la méthode d'Otsu, cette méthode ne défini pas de seuil global. Elle consiste a faire circuler une zone sur l'image et a obtenir des seuils propres a chaque zones. Comme dit précédemment, cette méthode fonctionne bien sur un fond uni mais fonctionne moins bien sans. C'est là qu'intervient le filtre de Wiener qui permet de savoir les variations du fond. Ainsi, en comparant les deux résultats on obtient de très bonnes performances. Cependant, Marius n'a pas encore compris avec précision son fonctionnement ; il ne l'a donc pas encore implémenté.



Les différentes méthodes de traitement

5.1.1.2 Difficultés rencontrées

La plus grande difficulté que Marius a rencontrée est sûrement la compréhension des méthodes de binarisation. Il a passé du temps à comparer les différentes méthodes. Car il a au début cherché une algorithme absolu avant de comprendre qu'il ne serait pas possible d'en trouver un tel algorithme. En effet il n'en existe pas aujourd'hui et s'il était capable de le créer il n'aurait rien à faire dans une école d'informatique. Aussi il s'est donc fixé pour objectif de faire une méthode qui marche bien avec un document scanné blanc et bien éclairé. La méthode d'Otsu est très bien pour cela (et semble d'ailleurs être la plus vieille de toutes les méthodes qu'il a trouvées et elle est encore pertinente), c'est d'ailleurs celle que nous avons utilisé. Il a ensuite fait des recherches pour la compléter et trouvé la méthode de Gatos et al comme cité précédemment. Cependant il s'est heurté à la compréhension de cette méthode et ne préfère pas l'appliquer sans savoir ce qu'elle fait précisément. Pour tenter d'améliorer mes résultats il a aussi fait un égalisateur d'histogramme. Le problème est que l'égalisateur d'histogramme ne permettra que de meilleurs performances sur un document restant dans un domaine déjà maîtrisé avec Otsu.

5.1.2 Soutenance 2

Comme Marius avait déjà effectué des recherches sur des moyens pour améliorer la binarisation, c'est lui qui s'est chargé de la continuer. Il a donc entrepris de trouver des méthodes et de procurer des outils permettant de compléter ceux déjà existant. Nous allons donc voir quelles sont méthodes choisies par Marius, pourquoi et comment elles ont été utilisées pour améliorer le traitement de l'image.

5.1.2.1 Adaptation a plus de document

L'un des principal problèmes de la méthode que nous utilisions, la méthode d'Otsu, est que celle ci gère mal le changement d'éclairage sur une partie du document. Un autre problème important de la méthode d'Otsu est que cela ne trouve pas bien le seuil optimal lorsque le contraste est faible. Pour pallier à cela Marius a d'abord fait des recherches suite à un conseil de Madame DERRODE sur la possibilité d'utiliser un égalisateur d'histogramme. L'utilisation d'un égalisateur d'histogramme permet d'augmenter le contraste ce qui pallie à l'un des problèmes cités plus haut. En effet en trouvant le seuil sur une image égalisée, on augmente le contraste et facilite la binarisation sur les images manquant de contraste. Cela permet aussi de réduire le bruit quelques fois mais cependant cela fonctionne mal sur une image déjà assez contrasté. C'est pourquoi nous n'appliquons cette méthode que lorsque nous suspectons une présence excessive de noir.

5.1.2.2 Réduction du bruit

Pour pouvoir mieux gérer les changements d'éclairage et les moment ou le contraste trop faible, Marius a utilisé la méthode de Sauvola. Cette méthode présente de nombreux avantages qui permettent de compléter la méthode d'Otsu. Le premier intérêt de cette méthode est qu'elle fait un seuil local pour chaque pixel. À la différence de la méthode d'Otsu qui donne un seuil global la méthode de Sauvola donne un seuil local, "personnalisé", pour chaque pixel. Cela permet notamment de mieux gérer un changement d'éclairage sur le document. De plus la méthode de Sauvola a la différence d'autres méthodes de seuillage local, comme Niblack par exemple, a la particularité de réduire le bruit. C'est un bonus non négligeable. Cependant l'un des défauts de la méthode de Sauvola est qu'elle a deux paramètres qui doivent être réglés manuellement pour mieux s'adapter aux différents types de document. Mais ce problème n'est pas très important si nous utilisons des document de même format. Enfin la méthode de Sauvola est aussi utilisée par la méthode de Gatos et al. Cette méthode que Marius a déjà mentionnée à la première soutenance est très puissante et peut compléter voir remplacer les méthodes d'Otsu et de Sauvola. Une autre voie de progression que Marius avait repérée est la mise en parallèle d'Otsu et de Sauvola pour donner de meilleurs résultats. Cependant du à des difficultés de la gestion des matrices, Marius n'a pas eu le temps d'implémenter ces méthodes.
Comme nous utilisons un format d'image qui diffère peu, la méthode de Sauvola est souvent plus performante que la méthode d'Otsu. En effet elle fonctionne vraiment très bien sur les document blanc texte au fond blanc. Aussi nous avons préféré utiliser celle-ci de préférence.

5.1.2.3 Liens avec la segmentation et le réseau de neurone

Pour faciliter les échanges entre le pré-traitement et la segmentation Marius s'est chargé de la transformation d'une image en matrice. Il a aussi fait des dossiers propres et séparés pour faciliter l'appel de Sauvola et de Otsu. Cela a permis à William et lui même de tester la segmentation plus en profondeur. Ce qui a permis de trouver quelques problèmes notamment lors de la segmentation d'image de travers.

5.1.2.4 Difficultés rencontrées

L'une des plus grandes difficultés rencontrées a été l'utilisation des matrices pour Sauvola. Avec la découverte des pointeurs et des allocations en C cette année cela n'a pas été facile de se plonger dans le bain. De plus Marius avait d'abord schématisé son algorithme de la méthode de Sauvola en python, ce qui n'était pas le plus facile a passé en C avec les matrices. C'est d'ailleurs pour cela que Sauvola n'est pas aussi optimisé que souhaité. Cependant à cause du manque de temps nous avons préféré faire quelque chose de fonctionnel et moins optimisé qu'un algorithme très optimisé

mais qui bogue encore pour la soutenance.

5.2 Segmentation

C'est William qui c'est occupé de la partie partie segmentation.

5.2.1 Soutenance 1

5.2.1.1 Découpage Vertical/Horizontal

Ce sont les deux fonctions principales, qui vont permettre de quadriller l'image en bloc. Afin de traiter la segmentation nous traiterons sur une matrice de 0 et 1 représentant l'image.Puis que qu'il faudra seulement les rappelées x nombre de fois jusqu'à que la découpe de l'image en caractère est faite. Pour le découpage et ainsi délimiter où sont mes caractères et les séparer des espaces vides. Ces deux fonctions font appel à deux autres fonctions de coloration qui vont permettre de changer les pixels qui ne font pas partie d'un bloc de texte et leur donné une nouvelle valeur. Lors du parcours de la matrice si on arrive à aller d'un bord à l'autre, ou d'une ligne rouge à une autre sans rencontrer de pixel noir, alors la ligne change en remplaçant les 0 par des 3 en appelant coloration. Ce qui permettra de par la suite de délimiter les blocs en rognant l'espace vide. Lorsque l'on appellera les fonctions et qu'elles ne changeront plus rien à la matrice, cela signifiera la fin du découpage de caractère. La matrice sera alors composée de 0 et de 1, ce qui caractérisera les caractères et de 3 pour séparer toutes les matrices représentant blocs,lignes et caractères.

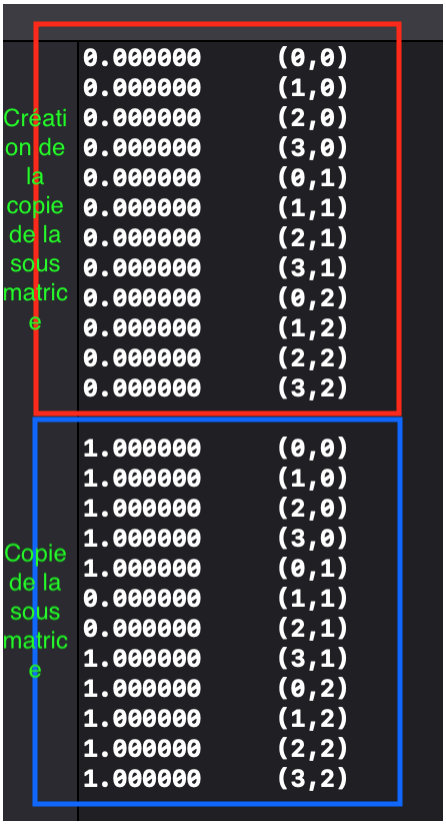
5.2.1.2 Récupération des blocs

Après avoir procédé au découpage de la matrice en caractère, il nous faut maintenant pouvoir les extraire afin de les données par la suite au réseau de neurones.Tout d'abord nous effectuerons un parcours de la matrice, représentant l'image, afin de pouvoir passer par tous les blocs que nous avions pu découper au préalable. Lorsque l'on rencontrera un groupement de 0 et de 1, c'est-à-dire un caractère, il faut alors pouvoir récupérer cette sous-matrice.

— La première fonction permettra tout d'abord de créer une nouvelle matrice de même taille que celle rencontrée, elle sera alors entièrement composée de 0.

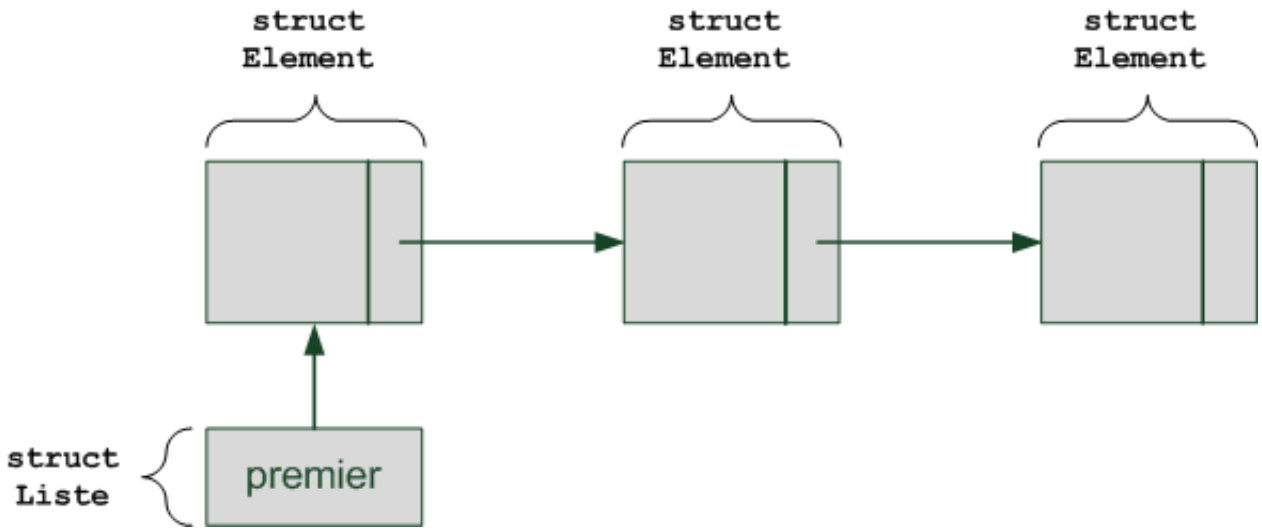
— La deuxième fonction quant à elle copiera la première matrice rencontrait lors du parcours, sur la deuxième.

Grâce à ces deux fonctions on pourra alors récupérer chaque caractère (sous-matrice) de l'image pour par la suite pouvoir les stocker dans une liste chaînée.

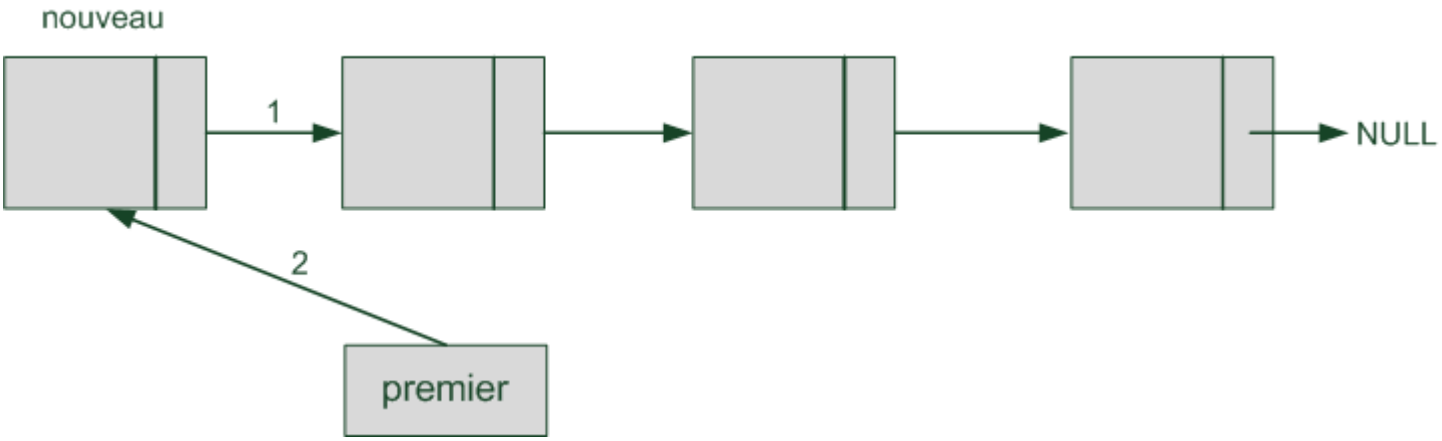


5.2.1.3 Structure Liste et Matcoor

Afin de stocker les matrices à un endroit, et voyant que l'on ne pouvait pas créer de tableau car il est impossible en C, impossible d'y ajouter des cases. Il a alors fallu créer alors notre propre structure liste afin d'initialiser une liste chaînée. La liste pourra alors être contrôlée par un pointeur vers le premier élément, ce qui permettra d'accéder à tous les autres éléments de la liste jusqu'au dernier qui lui pointera vers NULL.



Il a fallu aussi créer la structure de l'élément que l'on mettra dans notre liste, il contiendra évidemment une matrice mais aussi sa position sur l'image (x, y) ce qui permettra de facilement retrouver où l'insérer lors de la reconstruction du texte. Ce qui amènera à la création de MatCoor qui contiendra une matrice et sa position. Quelques fonctions permettant de manipuler notre liste on aussi étaient créées : - comme l'insertion en début de liste qui consistera à faire pointer le nouvel élément vers son futur successeur, qui est l'actuel premier élément de la liste, puis de faire pointer le pointeur qui pointe le premier élément de la liste vers le nouvel élément.



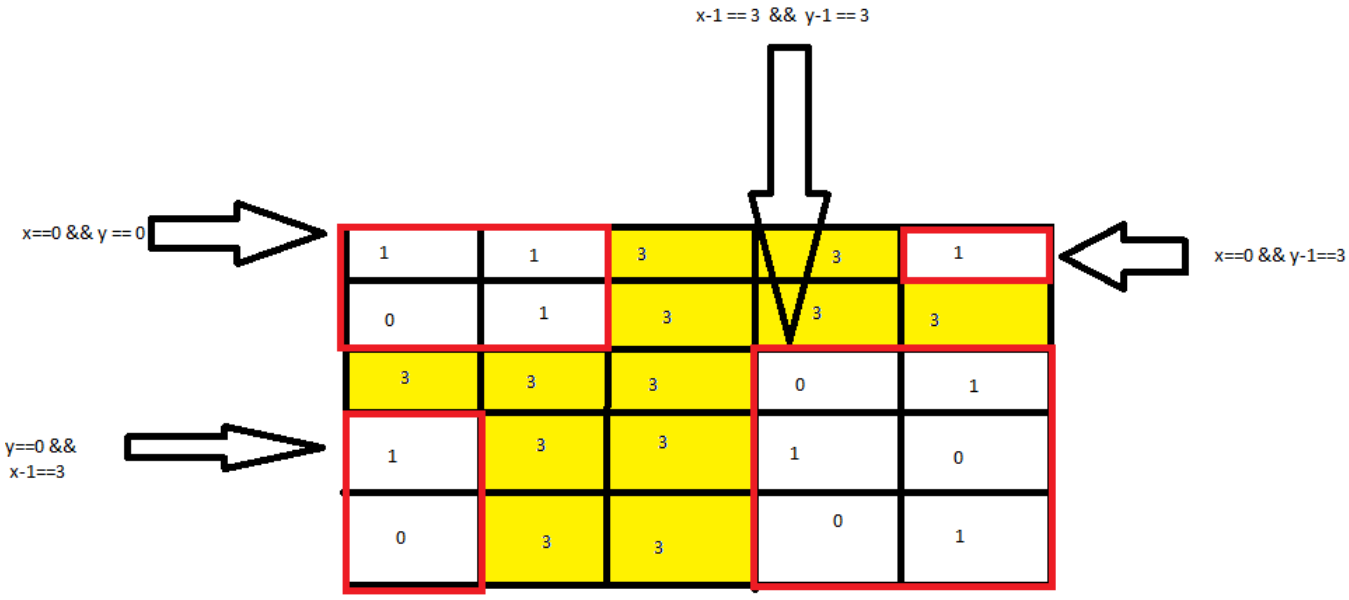
- Printlist : qui permettra d'afficher la liste dans la console. Dès qu'il faudra allouer de l'espace de stockage nous utiliserons malloc et sizeof(*liste) qui permettra de calculer automatiquement la taille à allouer pour le stockage de la structure liste.

5.2.1.4 Insertion des sous matrices dans la liste

Comme dernière fonction pour la segmentation de l'image sera alors de chercher chaque caractère (représenter par une matrice de 0 et 1, et encadré de 3) dans la matrice les copier puis les insérer dans la liste avec la position de leur premier pixel rencontrée celui en haut à gauche. Il y a seulement 4 possibilités de coordonnées pour savoir si le pixel et le premier de sa matrice :

- Déjà il ne doit être pas égal à 3 sinon cela voudrait dire qu'on ne se situerait pas dans un caractère.
- Si x est égal à 0, y aussi alors c'est la première case de la matrice de l'image, et le premier de son caractère.
- Si x est égal à 0, y-1 égal à 3.
- Si y est égal à 0, x-1 égal à 3.
- Si x-1 est égal à 3 et y-1 égal à 3.

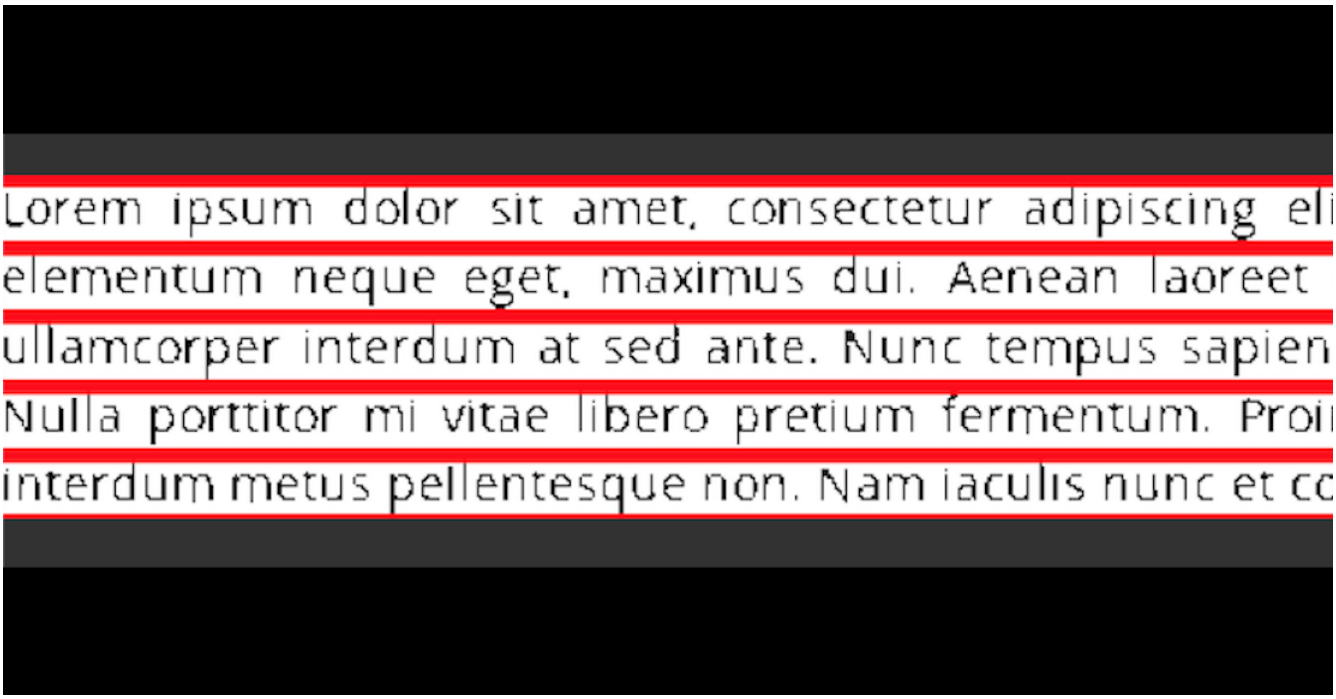
Voilà les 4 possibilités de coordonnées que l'on pourra trouver pour chaque rencontre avec le premier pixel d'une sous-matrice. Lorsque l'on trouvera le premier caractère ce sera à ce moment que l'on l'initialisera notre liste puis on se contentera de rappeler trace() et copy() afin d'y ajouter le caractère à notre liste (toujours en forme de matrice) avec son point de départ.



5.2.2 Soutenance 2

5.2.3 Mise à jour de la segmentation

Après la première on s'est rendu compte que la segmentation ne marchait pas totalement, cela découpait seulement la première ligne en caractère et ne faisait pas des matrices rectangulaires pour toutes les lettres. William a alors dû s'occuper de corriger la segmentation. Il a alors d'abord regardé sa fonction qui permet de découper et séparés toutes ces lignes en transformant la valeur des lignes blanches en 3, ce qui nous permettra après de séparés chaque lettre. On a rencontré plusieurs problèmes lors du découpage vertical, le découpage s'effectuer bien sur la première ligne comme ils n'y avaient aucun caractère au-dessus. Mais pour les caractères en dessous nous n'avions pas du tout les résultats voulus. On a donc changé notre façon de penser et regardé le problème autrement. Après le découpage horizontal nous avions ça :

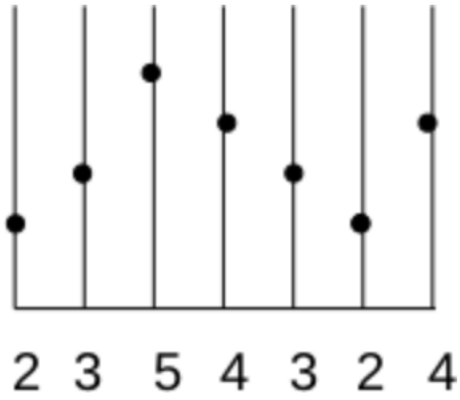


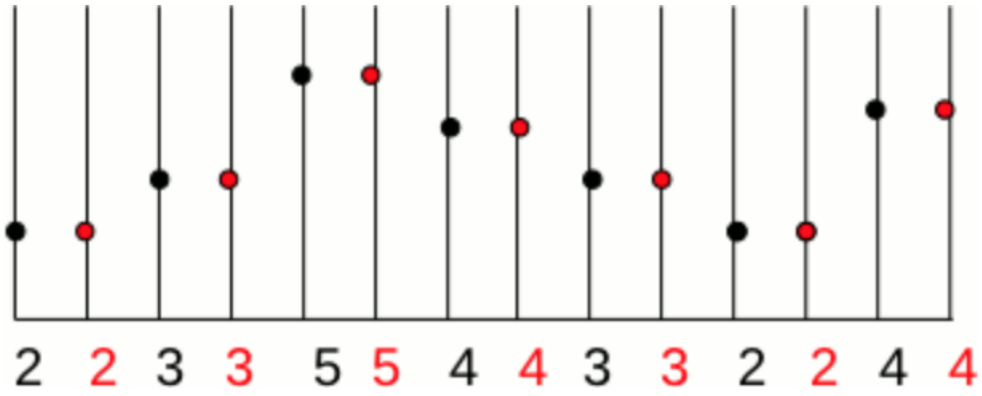
Des bandes blanches horizontales que l'on veut découper verticalement dès que l'on y trouve une ligne blanche verticale entre les deux bandes rouges, on a donc parcouru toute la matrice représentant l'image, et à chaque début de bande blanche, on va chercher à tracer une ligne rouge verticalement afin de séparer tous les caractères, lorsque l'on a fini une bande on recommencera dès la prochaine bande blanche. On obtiendra alors à la fin chaque caractère dans de l'image en les séparant par des 3. Chaque matrice est un rectangle qui sera pour l'instant pour chaque bande

de largeurs différentes, et chaque lettre aura une longueur différente correspondant à la taille de celle-ci.

5.2.4 Lien avec le réseaux de neurones

Les réseaux de neurones à besoin de recevoir pour chaque caractère une matrice de même dimensions, de 28*28 ou de 16*16.Alors William a dû implémenter une fonction permettant de redimensionner une matrice afin de les mettre dans le format voulu.Il y avait le choix entre deux méthodes, redimensionner par le plus proche voisin ou par interpolation bilinéaire.William a choisi d'implémenter la fonction par plus proche voisin, le principe est de calculer pour la longueur et la largeur de la matrice du caractère, un coefficient en divisant par les longueurs et largeurs voulues.





$$\frac{longueur(oulargeur)delamatriceducaractre}{longueur(oulargeur)delamatricevoulue}$$

Ces deux coefficients permettront de savoir quel pixel nous allons récupérer dans la matrice du caractère afin de le mettre dans la nouvelle qui a les bonnes dimensions. Nous parcourrons alors les deux matrices simultanément mais à chaque nouvelle itération de la boucle, nous ajouterons les coefficients aux positions x et y (coordonnées utilisées pour parcourir la matrice de la lettre), contrairement aux coordonnées (i, j) qui lui seront à chaque fois incrémentées de 1 afin de parcourir

et rajouter les pixels dans la nouvelle matrice. Cela nous permettra donc à la fin d'avoir une matrice redimensionner dans les proportions voulues.

Puis après avoir redimensionné les matrices ont les envois dans la liste chainée, qui contiendra les Mat Coor, structure composée des matrices et de leurs coordonnées, qui nous permettra des passés une par une au réseau de neurones. William a changé le fonctionnement de la liste, elle ajoutait au début chaque élément, ce qui n'était pas très pratique pour la reconstruction de texte. Il a donc corrigé sa fonction, en allant sur le dernier élément contenu dans la liste qui pointait sur NULL, le faire pointer sur le nouvel élément, et faire pointer ce dernier sur NULL.

Pour exemple, ci-dessous vous pouvez voir un résultat du terminal, en premier la liste représentant toute mes matrices(ce sont leurs coordonnées qui sont affichés ici) ensuite chaque lettres qui a était découpés avec la taille de leur matrice en dessous.

5.2.5 Difficultés rencontrées

L'une des grandes difficultés rencontrées, était la création du makefile afin d'inclure les bonnes bibliothèques, et les bons fichiers afin de tester son code, ce qui a beaucoup retardé William, qui obtenait des erreurs dont il ne savait pas si elle venait du makefile ou de son code. La découpe verticale a aussi causé quelques problèmes dut au fait de pensé qu'il aurait seulement fallu inverser le sens de parcours de la découpe horizontale pour que cela marche, mais dès qu'il a changé sa manière de pensée, il a su résoudre le problème.

5.3 Réseau de neurone

Pour la première soutenance, c'est Nicolas et Cédric qui se sont occupé du réseau de neurone. Pour la seconde soutenance il a été décidé que Nicolas continuerait seul sur le réseau pour que Cédric puisse s'occuper de l'interface graphique.

5.3.1 Soutenance 1

Vu que le réseau de neurone est une notion assez longue et complexe à comprendre, Nicolas et Cédric ont décidé de faire des recherches chacun de leur côté. Ils ont tout d'abord lu les premiers chapitres du livre en ligne *Neural Network and Deep Learning*. Ils ont ensuite fait des recherches sur différents sites. Le but était de réaliser dans un premier temps un réseau neuronal capable d'apprendre la porte Xor. La fonction d'activation retenue était la fonction sigmoïde, dont la formule est la suivante :

$$\frac{1}{1 + \exp^{-f}}$$

f étant égale à la formule suivante (où w est un poids du nœud, x l'input lié au nœud par w et b étant le biais du nœud) :

$$f = \sum_j w_j x_j + b$$

5.3.1.1 Réseau de Nicolas

Nicolas a d'abord réalisé un réseau en python, afin de comprendre comment cela fonctionnait. Il a eu du mal a comprendre toutes les subtilités ainsi que toutes les notions mathématiques. Au début il essayait de comprendre les formules mais aussi les démonstrations. Il a fini par abandonner ces-dernières, car elles devenaient bien trop compliquées. Ensuite, une grosse partie du travail a été de recréer les fonctions de calculs sur des matrices. La majorité des personnes qui réalisent un réseau de neurone en Python utilisent la librairie numpy pour pouvoir réaliser facilement ces calculs. Cependant, elle n'existe pas en C. Cela a pris pas mal de temps a Nicolas de recréer les fonctions nécessaires sans qu'il n'y ait d'erreurs. Cela n'a pas toujours été facile car, même s'il avait déjà fait quelques unes de ces fonctions l'année précédente, il avait oublié comment les coder. Par exemple, il a passé beaucoup de temps sur la produit matriciel car il avait oublié que trois boucles étaient nécessaires. Après avoir avoir fini toutes ces fonctions, il a terminé son réseau de neurone. Cependant celui-ci ne fonctionnait pas.

Le débogage d'un réseau de neurone peut s'avérer très long et pénible. Il faut afficher de

nombreuses valeurs et refaire les calculs soit même à côté afin de vérifier que le réseau obtient les bons résultats. Parfois il arrive aussi qu'un oubli dans le code ne soit pas forcément remarquable par calcul et provoque l'apparition de résultats anormaux. C'est ce qui c'est passé pour le réseau de Nicolas. Il avait oublié de faire la transposée d'un matrice dans un calcul et cela faussait totalement les résultats. Après une bonne semaine de débogage, son réseau était finalement fonctionnel. Il ne restait qu'à le transcrire en C. La première difficulté à laquelle il fit face était le découpage du code. Il fallait en effet réfléchir à comment répartir le code dans divers fichiers afin de garder le tout propre, compréhensible et modulaire. Mais le premier vrai problème qu'il a rencontré a été la création de la structure de matrice et la retranscription des fonctions de calculs. En effet, la taille des matrices n'est pas connu d'avance. Il fallait donc pouvoir créer des matrices de toutes les tailles n'importe quand. Il a donc fallu qu'il se renseigne sur l'allocation dynamique. Quelques uns de nos camarades avaient déjà étés confrontés au même problème et l'orientèrent vers les fonctions malloc et calloc.

Après avoir trouvé ce qu'il cherchait, il lui a fallu apprendre le fonctionnement des pointeurs à la dure. En effet, le TP sur ces-derniers n'était alors pas encore sorti. Il a donc réalisé de nombreuses recherches sur différents sites tels qu'OpenClassroom, Developpez pour les principes de bases, des sites comme Koor ou Tutorialspoint pour obtenir la description précise de certaines fonction C (comme malloc et calloc), mais aussi des sites comme Stackoverflow pour voir si d'autres personnes avaient les mêmes problèmes que lui et comment il pourrait améliorer son code. Nicolas a rencontré pas mal d'erreur à cause des pointeurs mais, à force d'expérience, les choses à faire et à ne pas faire ont commencé à rentrer. La deuxième bête noire quand on découvre les pointeurs, ce sont les fuites de mémoires. Nicolas a dû s'y confronter de nombreuses fois. Chez nos camarades ayant utilisé l'allocation dynamique de mémoire, ces fuites ne dépassaient généralement pas l'ordre de la trentaine d'octets. Mais dans son cas, la fuite était de l'ordre de 20 000 octets. Cela était dû a des plusieurs fuites de mémoire pour les matrices dans une boucle. Sachant que pour tester son réseau l'entraînait pendant 10 000 époque, il était compréhensible d'atteindre un tel chiffre. Dans ces moments là, valgrind a vraiment été utile car il permettait de trouver où se produisait les fuites de mémoires.

L'un des petits problèmes en C est la génération de nombres pseudo-aléatoires. En effet, contrairement à de nombreux langages (OCaml, Python et C# pour ne citer que des langages que nous avons utilisé) le système de génération pseudo-aléatoire du C ne permet pas de choisir des bornes. Le nombre généré est un entier positif compris en 0 et RAND_MAX valant 2147483647. Les poids et les biais du réseau de neurones sont normalement initialisés entre 0 et 1. Une première possibilité de résoudre ce problème serait de multiplier le nombre généré par un nombre dans $\mathbb{R}$, puis de faire un modulo 1. Cependant, pour un réseau de neurone il est préférable d'utiliser une formule de

génération pseudo-aléatoire à répartition uniforme. Lors de ses recherches, Nicolas est tombé assez vite sur cette formule. De nombreuses personnes utilisent le modulo mais il a trouvé un thread de Stackoverflow où une personne parlait justement du fait que la génération n'était pas uniforme dans le cas de l'utilisation de modulo et qu'il valait mieux utiliser cette formule. La personne donnait une version de la formule qui ne convenait pas vraiment à Nicolas car celle-ci ne permettait que de préciser une borne extérieure. Quelques recherches de plus ont lui ont cependant suffit à trouver la formule qu'il cherchait. Cette formule est la suivante :

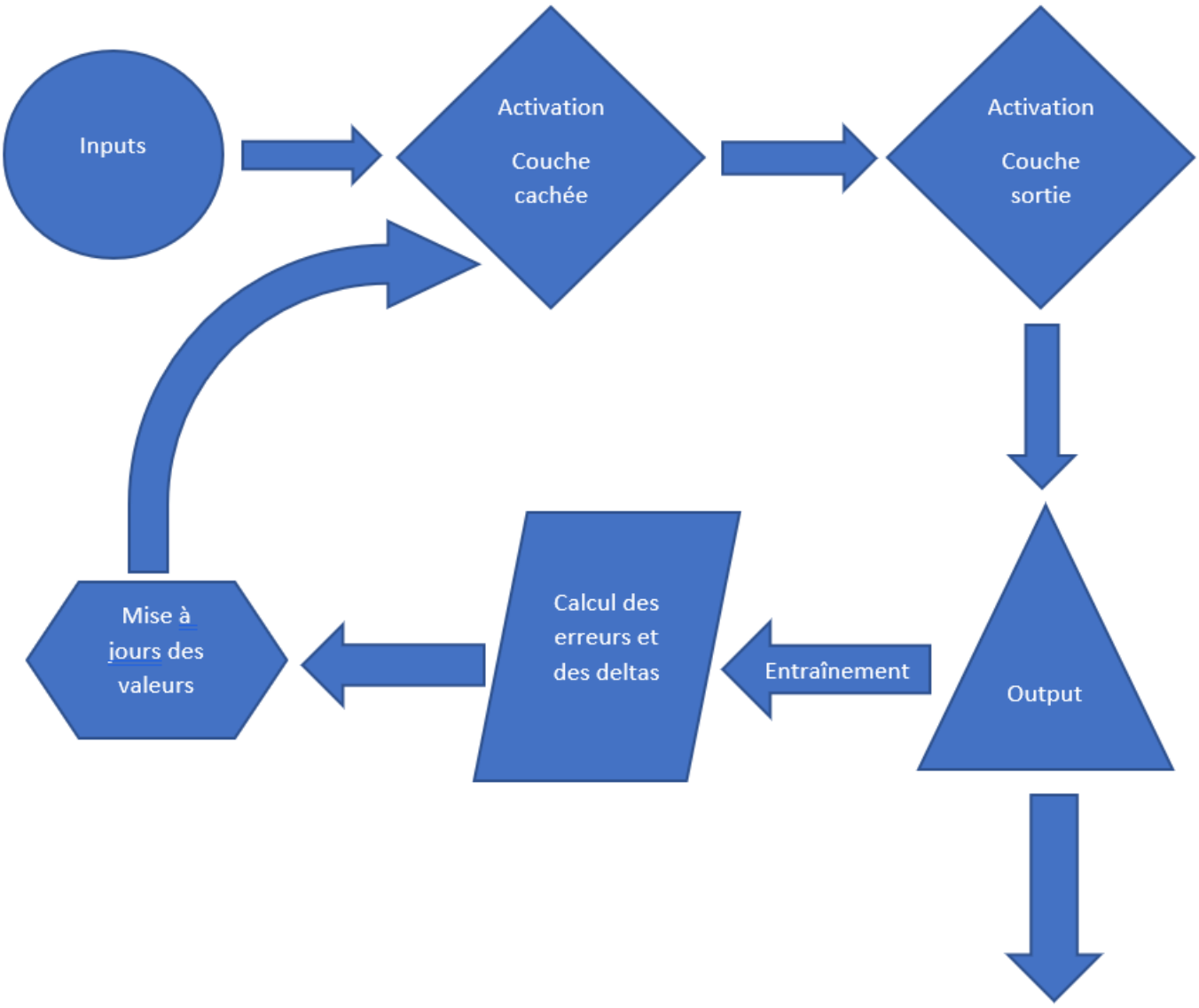
$$\frac{rand()}{RAND\_MAX * (max - min) + min}$$

Un autre problème relatif à l'allocation de mémoire que Nicolas a eu du mal à résoudre a été la double libération de mémoire. Il voulait faire en sorte que les entrées du réseau soient mélangées à chaque époques. Cependant, à cause d'une histoire de pointeurs, il libérait une première fois la mémoire, puis le refaisait une deuxième fois au même endroit. Une fois le problème réglé, Nicolas remarqua que son réseau ne donnait plus les bonnes valeurs. Normalement la mélange des données se fait lorsque le réseau utilise un système de mini-batch. Le problème était que ce qu'il avait fait ne servait pas dans le cas d'un réseau entraîner pour reconnaître la porte Xor. En effet, il n'y a que 16 possibilités de disposition des données. De ce fait, le mélange des données à chaque époque de l'apprentissage n'est pas efficace. Au contraire il est même contre-productif. Le mélange des données fait qu'il est possible que parmi les 16 versions possible de la combinaison 00, 01, 10 et 11, une pouvait apparaître plus que les autres. Il aurait donc fallu augmenter le nombre d'époque afin de compenser ce problème et ainsi harmoniser les changements, sans pour autant être sûr que le problème ne continu pas de se poser par malchance. Nicolas a donc effacé ce qu'il a fait. Cependant, les fonctions qu'il a utilisées ont été conservées, dans la vision d'une possible utilisation.

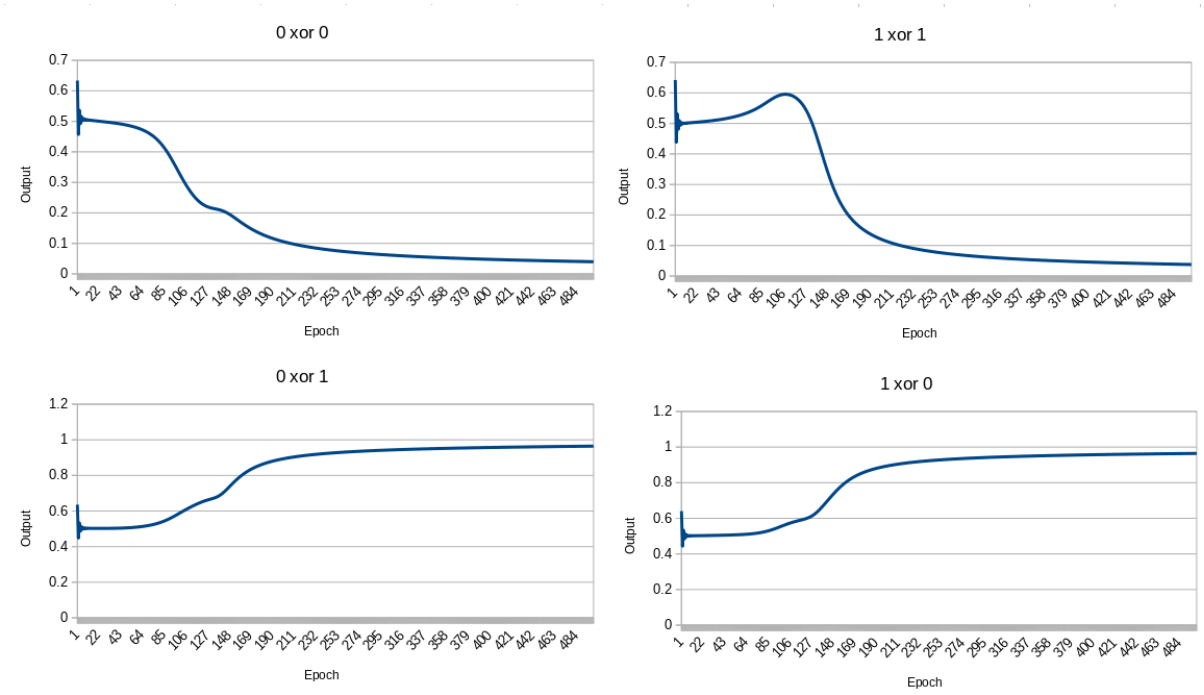
Il serait en effet intéressant de mettre en place le système de mini-batch car il prendrait du sens dans le cas de la reconnaissance de caractères. Dans le cas où nous prendrions un matrice de dimensions 16x16 par caractères, le réseau posséderait 256 neurones, juste en entrée. Les calculs seraient donc bien plus longs que ceux actuels, surtout que notre base de données ne va pas seulement contenir une seule version de chaque caractère alpha-numérique. Le principe du mini-batch est de prendre aléatoirement une partie des éléments du jeu de données afin d'obtenir une portion relativement petite de ce-dernier tout en restant approximativement représentatif de l'ensemble. Le but est ainsi de faire moins de calculs tout en conservant une précision acceptable, et ainsi d'accélérer de manière notable le calcul. Par exemple, imaginons que notre jeu de données soit composé de 200 images de caractères. Si l'on prend un mini-batch de 10 éléments, on augmenterait la vitesse de calcul par 20. Sachant que l'alphabet alpha-numérique est composé de 62 caractères,

si l'on rajoute les éléments de ponctuation basiques tels que : . ,?! etc on obtient environ 70 caractères. Sachant qu'il y aura plusieurs versions de chaque caractères, le système de mini-batch prend du sens dans notre situation.

Voici comment fonctionne actuellement le réseau de neurone :



Voici comment évolue les valeurs trouvées par le réseau lors d'un entraînement sur 500 époques avec un taux d'apprentissage de 3.2 :



Nicolas s'est ensuite penché sur la sérialisation des valeurs de son réseau de neurones. Il s'est compliqué la vie et la majorité de ce qu'il a fait c'est finalement avéré inutile, sauf d'un point de vu exercice sur les pointeurs. En effet sa grosse erreur a été de vouloir faire une sérialisation renvoyant une chaîne de caractères. Cependant cela est compliqué à gérer car il faut allouer la bonne quantité de mémoire pour stocker tous les caractères nécessaires. Le format qu'il a utilisé est le suivant : x y lw lb Où x est le nombre d'inputs, y le nombre de neurones, lw l'ensemble des poids et lb l'ensemble des biais.

Pour réaliser ce format, Nicolas a décidé de créer une chaîne de caractères par ligne, puis de les concaténer. Pour transformer les valeurs en chaînes de caractères, il a utilisé la fonction sprintf qui permet d'écrire une chaîne formatée dans un buffer pré-alloué. Il était donc nécessaire de calculer la place nécessaire pour stocker chaque chaîne, ainsi que la chaîne globale. Après avoir bien réfléchi Nicolas produisit ses premières formules pour les allocations de mémoire. Elles ne fonctionnèrent pas du premier coup, mais ont nécessité quelques ajustements. Après le problème de la taille des buffers des grandes chaînes est venue le problème de la taille de ceux d'une petite chaîne temporaire. Cette dernière était utilisée pour stocker une par une les différentes valeurs de poids et de biais pendant une durée très courte. Cela permettait de convertir les valeurs en chaîne de caractères et d'avoir un format. Après quelques modifications dans les différentes formules, tout semblait fonctionner. Cependant après quelques testes, il s'avéra que lorsqu'une valeur était négative les buffers n'avaient pas une taille suffisante pour les stocker. En effet, Nicolas avait oublié de

prendre en compte le symbole - devant les nombres négatifs. Cela faisait un caractère de plus à chaque nombre négatif. Il fallait donc parcourir les matrices afin de compter le nombre de valeurs négatives pour que cela soit pris en compte dans les formules.

Première formule de calcul de position du premier caractère de la valeur dans la chaîne de sérialisation des poids (j est la coordonnée x d'un élément dans la matrice et i la coordonnée y) :

$$9 * (i * nbinputs + j) + k$$

Un autre problème s'est posé : celui des décalages dans les chaînes en fonction de si la valeur ajoutée était négative ou positive. Il était nécessaire de prendre en compte le nombre de valeurs négatives et positives ajoutées afin de corriger les positions données par les formules. Il a donc fallu modifier les formules en conséquences.

Formules de calcul de position du premier caractère de la valeur dans la chaîne de sérialisation des poids pour prendre en compte les négatifs (j est la coordonnée x d'un élément dans la matrice, i la coordonnée y, neg_passed (resp. pos_passed) le nombre de valeurs négatives (resp. positives)) :

— Dans le cas où la valeur est positive :

$$9 * (i * nb\_inputs + j) + k + neg\_passed$$

— Dans le cas où la valeur est négative :

$$10 * (i * nb\_inputs + j) + k - pos\_passed$$

Après tous ces ajustements, voici à quoi ressemblait la formule de calcul de l'allocation pour la chaîne finale de la sérialisation (c_i étant le nombre de chiffre du nombre d'inputs, c_n le nombre de chiffre du nombre de neurones de la couche, x_d le nombre total de valeur sur l'ensemble des matrices de poids et de biais, n_w le nombre de poids négatifs et n_b le nombre de biais négatifs) :

$$sizeof(char) * c_i * c_n + 9 * x_d + n_w + n_b + 3$$

Cette formule s'explique de la manière suivante :
C'est la somme du nombre de chiffre du nombre d'inputs, du nombre de chiffre du nombre de neurones de la couche, du nombre de valeurs à virgules multiplié par 9, du nombre de caractères non numériques (l'espace, le signe - et le retour à la ligne) et d'un caractère NULL qui indique la fin de chaîne en C. Cette multiplication par 9 n'est pas forcément évidente à comprendre, mais elle est en réalité assez évidente lorsqu'on déroule un peu le calcul. Un double possède 6 chiffres

après la virgules, un point qui indique la virgule et des chiffres à gauche de celle-ci. Cependant la fonction sigmoïde renvoie des valeurs comprises entre 0 et 1. En paramétrant le réseau avec un taux d'apprentissage suffisamment faible, les valeurs des poids et des biais seront toujours comprises dans l'intervalle]-10,10[. Cela permet de déterminer qu'il n'y aura qu'un seul chiffre à gauche de la virgule. Cela fait donc 8 caractères par valeur. La raison pour laquelle le 9 apparaît est que si on ne prend que la fin de la formule on a :

$$\begin{aligned} caracteres\ pour\ les\ doubles + caracteres\ speciaux &= 8 * x_d + x_d - 2 + n_w + n_b + 5 \\ &= 9 * x_d + n_w + n_b + 3 \end{aligned}$$

Cette implémentation de la sérialisation comportait un défaut majeur : elle ne fonctionnait plus dans le cas où les poids et/ou les biais avaient une valeur n'appartenant pas à]-10,10[. Cela restreignait donc les possibilités de paramétrage. De plus, elle ne servait pas à vraiment car elle était inutilement complexe par rapport a ce que l'on pouvait faire si l'on sérialisait directement dans un fichier ; ce que l'ont peut faire dans notre cas. En effet, lorsque nous allons sauvegarder les poids et les biais, aucun traitement ne sera effectué sur la chaîne qui pourrait être produite. Ainsi, si l'on utilise juste la fonction fprints comme Cédric l'a fait dans sa version, on peut écrire dans le fichier avec le bon format sans s'embêter avec des allocations de mémoires, et donc avec tous les calculs qui vont avec. Bien que tout cela n'ait pas été très utile dans l'avancement du projet, cela n'a pas non plus été une perte de temps. Cela a permis a Nicolas d'améliorer ses connaissances sur les pointeurs, et cela lui évitera de refaire la même erreur pour la désérialisation.

5.3.1.2 Réseau de Cédric

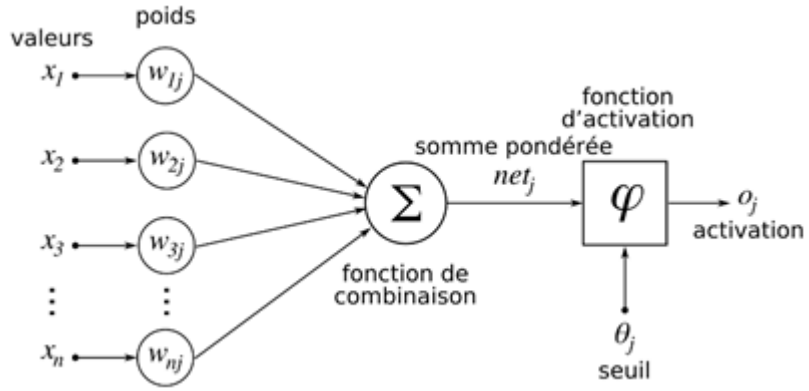
Cédric a commencé par de longues heures de documentations. Deux problèmes principaux se démarquais de manière évidente :

— L'apprentissage du langage C
— Le fonctionnement d'un réseau de neurone

Dans un premier temps il a fallu s'imprégner des différences entre un langage ancestral comme le C et un langage plus récent tel que C# qu'il connaissait déjà. En effet de nouvelles caractéristiques comme la gestion de la mémoire ont posé beaucoup de problème et nécessité beaucoup temps pour leur résolution. Cependant, cela a permis un apprentissage rapide et efficace de ce nouveau langage de programmation.

Le fait que le réseau de neurone informatique ressemble beaucoup à la version biologique a

beaucoup aidé à la compréhension des différentes mécaniques. Contrairement à ce qui été conseillé, Cédric a préféré commencer la programmation du réseau directement dans le langage C. Ce choix entraîna certains points positifs comme le fait d'avoir passé énormément de temps à utiliser le langage, et certains point négatifs comme les doutes très réguliers entre les erreurs d'implémentions et de conception du réseau de neurone.



Il a tout d'abord implémenté la structure de mon réseau, actuellement composé de :

— La valeur de chaque synapse
— La valeur de chaque neurone
— Les valeurs d'erreurs de chaque neurone
— Les valeurs d'erreurs de sortie
— La taille de chaque couche
— L'index des synapses
— Le nombre de couche
— Le nombre de neurones
— Le nombre de synapse

Cette structure va nous servir dans un premier temps à créer le réseau en allouant de la mémoire pour chacun de ses composants, ainsi qu'en en initialisant aléatoirement chaque poids synaptique par une valeur entre –1 et 1 (si un poids synaptique est négatif il sera alors inhibiteur et s'il est positif il sera excitateur). Cette partie sur l'allocation mémoriel a posé beaucoup de problème avec la fameuse erreur "Segmentation Fault".

Une erreur de segmentation est une erreur qui survient lorsque le programme tente d'accéder à un emplacement mémoire qui ne lui était pas alloué. Bien entendu cette erreur fait planter programme avant la fin de son exécution. Cédric a commencé à programmer son réseau sous Windows, ce qui a eu pour avantage le débogage plus facile de ce type d'erreur. Cependant en passant sous Linux, pour une raison qu'il ignore, des erreurs de ce type sont réapparut. A ce stade là le débogage fut bien plus compliqué. Cédric a donc décidé de se servir de Gdb.

Le logiciel gdb est un logiciel GNU permettant de déboguer les programmes C (et C++).
Celui-ci est programmer pour répondre aux 3 problématiques suivantes :

— La ligne ou s'arrête le programme en cas de terminaison incorrecte, notamment en cas d'erreur de segmentation.
— Les valeurs des variables du programme à un moment donné de l'exécution.
— La valeur d'une expression donnée à un moment précis de l'exécution.

Gdb permet donc de lancer le programme, d'arrêter l'exécution à un endroit précis, d'examiner et de modifier les variables au cours de l'exécution et aussi d'exécuter le programme pas-à-pas.

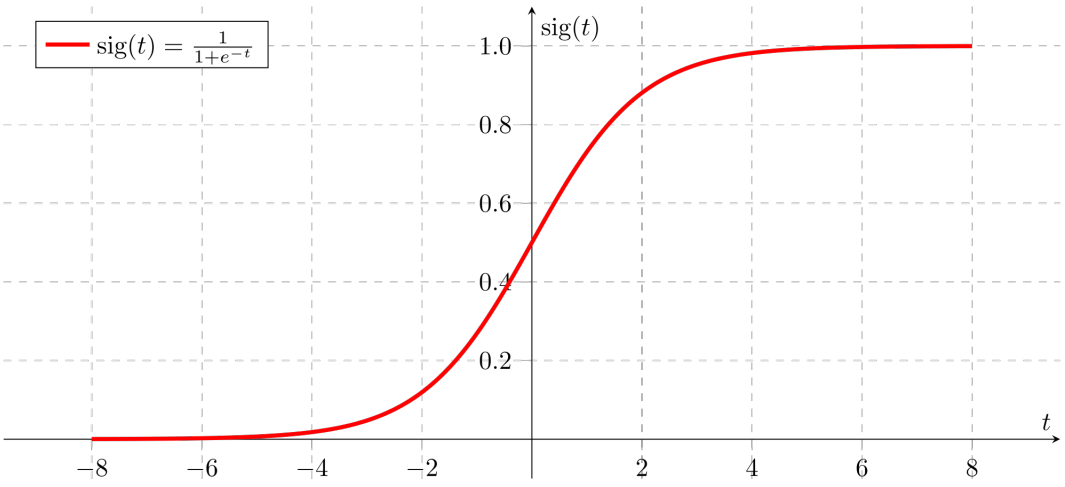
Ce debogueur a permis à Cédric à mieux se diriger durant le combat contre les erreurs de segmentation.

Cédric mit également beaucoup de temps à identifier la source d'un problème qui faussait les résultats du réseau. En effet si l'on veut utiliser une variable aléatoire sans qu'elle reste la même à chaque appelle de la fonction rand(), il faut d'abord initialiser rand au temps actuel (srand(time(NULL))), que l'on peut ensuite ramener dans notre intervalle grâce à diverse formules mathématiques comme :

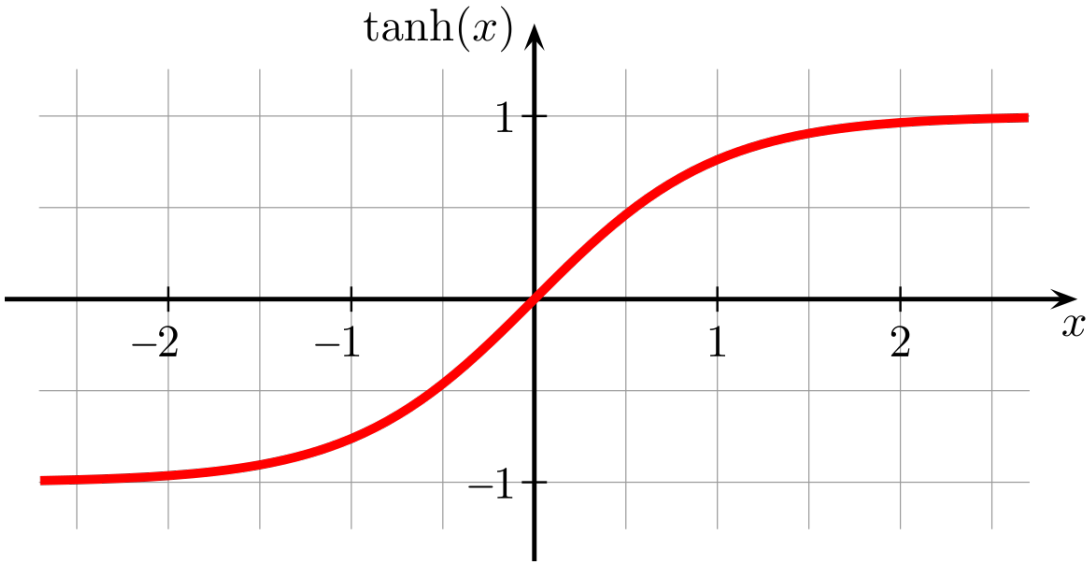
$$\frac{rand()}{RAND\_MAX * (max - min) + min}$$

Avec du temps et beaucoup de patience, ces problèmes se sont finalement résolus.

Une fois le réseau "créé" il a fallu définir les valeurs de nos neurones d'entrées et ainsi calculer la valeur de chaque neurone. Pour calculer la valeur d'un neurone, il suffit de faire la somme (pour chaque synapse entrante) du produit entre le poids de la synapse et la valeur du neurone d'où elle part, et de mettre ce résultat en paramètre de la fonction d'activation choisie. Pour des raisons pratiques, Cédric a choisi d'utiliser la tangente hyperbolique de la synapse et la valeur du neurone d'où elle part, et de mettre ce résultat en paramètre de la fonction d'activation choisie. Pour des raisons pratiques, Cédric a choisi d'utiliser la tangente hyperbolique plutôt que la fonction sigmoïde. Il est important de noter qu'il ne faut pas faire passer le résultat par une fonction d'activation si l'on calcule la valeur d'un neurone de sortie.



Fonction Sigmoïde



Fonction Tangente hyperbolique

Viens alors l'étape de l'apprentissage. La méthode de rétro-propagation fût choisie. Celle-ci consiste à corriger le poids de toutes les synapses (dont les biais) en partant des neurones de sorties jusqu'aux neurones d'entrées. La correction se fais dans un premier temps grâce au calcul de l'erreur de sortie (sortie – valeur désiré), qui va nous servir à calculer les poids des dernières synapses :

$$Poids\_Synapse = Poids\_Synapse - (Erreur * Taux\_Apprentissage * Neurone\_Amont)$$

$$Poids\_Biais = Poids\_Biais - (Erreur * Taux\_Apprentissage * 1)$$

Le taux d'apprentissage est une constante.

Pour les neurones intermédiaires, on calcule également leur erreur cette fois égal à la somme du produit entre la valeur d'erreur de sortie et le poids de la synapse (pour chacune des synapses reliées au neurone).

La formule mathématique calculant le nouveau poids de la synapse est presque similaire à la précédente.

$$Poids\_Synapse = Poids\_Synapse - (Erreur * Taux\_Apprentissage * Neurone\_Amont*$$

$$(1 - Neurone\_Aval * Neurone\_Aval))$$

La seule différence est qu'on a multiplier par (1 – Neurone Aval * Neurone Aval) qui est justement le dérivé de la tangente hyperbolique sur "Neurone Aval". D'où l'utilité d'utiliser cette fonction que Cédric connaissait déjà.

Pour conclure cette partie, un récapitulatif des connaissances qui ont été acquise s'impose.

— Une maîtrise plus approfondie de l'utilisation des pointeurs et de l'allocation mémoriel.
— Un apprentissage sur la conception d'un réseau de neurone informatique
— Découverte de fonctions mathématiques quelque fois intéressantes
— De la patience, beaucoup de patience

5.3.2 Soutenance 2

Lors de la deuxième partie de la réalisation de ce projet il y a eu une légère réorganisation des tâches. Il a été décidé que Nicolas continuerait à travailler sur le réseau neuronal, tandis que Cédric allait travailler sur l'interface graphique. De cette manière nous avons voulu nous assurer que chaque élément de la chaîne de production soit réalisé, afin que cette chaîne soit complète pour la soutenance.
Sachant que nous possédions 2 réseaux de neurones, il a tout d'abord été nécessaire de choisir duquel il allait falloir repartir. Après analyse, le réseau sélectionné fut celui de Nicolas car celui-ci était plus malléable que celui de Cédric.
En suivant les conseils donnés par le jury à la première soutenance, la seconde étape a été de rendre le réseau encore plus malléable que ce qu'il était alors. A la première soutenance, il était

possible de choisir le nombre de neurones d'entrée, de l'unique couche cachée et de la couche de sortie. Il a tout d'abord été question de pouvoir choisir le nombre de couches cachées et le nombre de neurones de ces couches, mais pas de manière individuelle. Ainsi, il n'était pas possible d'avoir des couches cachées avec des nombres de neurones différents. Cependant, pour plus de contrôle, la possibilité de choisir individuellement le nombre de neurones pour chaque couche cachée. Cela n'a pas été très long et très difficile à implémenter. Le code du réseau de neurones était structuré d'une telle manière que le changement de quelques lignes a suffit à implémenter ces options. Mais l'implémentation de ces choix à tout de même causé quelques problèmes mineurs. En effet, cela a nécessité de rajouter des boucles dans différentes parties du code afin de gérer la possibilité qu'il y ait plusieurs couches cachées. Ces petites modifications ont été sources d'erreur qui ont fait que le réseau ne renvoyait plus les bonnes valeurs. Ce problème n'a cependant pas été très difficile à régler vu que la cause a très vite été identifiée.

Ensuite, Nicolas a commencé à implémenter la fonction softmax pour la couche de sortie. C'est une fonction probabiliste qui est utile pour la classification en classes multiples. Par exemple dans notre cas, la fonction softmax est utilisée pour avoir la probabilité qu'une matrice binaire représente telle ou telle lettre de l'alphabet. Cela nous permet ainsi de reconnaître les caractères qui ont été extraits d'une image. La fonction softmax (σ) se présente de la manière suivante (z_i étant la valeur d'un neurone i avant activation) :

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Comme on peut le voir avec cette formule, il est nécessaire d'avoir accès aux valeurs de tous les neurones de la couche avant d'appliquer softmax. Or cela n'était pas possible. Il a donc fallu adapter le code du réseau pour que les fonctions d'activations aient accès aux valeurs de neurones. Avec quelques recherches supplémentaires sur la fonction softmax, Nicolas a découvert qu'il existait une autre version de cette fonction. En effet, à cause de l'utilisation de la fonction exponentielle, la fonction softmax pouvait renvoyer la valeur NaN. Cette autre version de la fonction était une version stabilisée qui se présente de la manière suivante (z étant l'ensemble des valeurs des neurones de la couche avant activation) :

$$\sigma(z_i) = \frac{e^{z_i - max(z)}}{\sum_{j=1}^{n} e^{z_j - max(z)}}$$

Ensuite, il a fallu implémenter la dérivée de softmax pour la rétro-propagation du gradient. C'est à partir de ce moment que les vraies difficultés ont commencé. Tout d'abord les explications de la dérivées ne sont pas toujours claires. Par exemple, de nombreuses fois les explications présentent des formules mathématiques sans expliquer ce que représentent les variables de la formule et surtout

ce que représente leur index. Il y a certains moment où il y a des changements de variable en index sans même qu'il n'y ait d'explication sur ce que représente le nouvel index. Ces premiers problèmes ont joué un rôle important dans la difficulté à comprendre la dérivée de la fonction softmax, notamment pour comprendre la matrice jacobienne de cette dérivée. Nicolas a donc dû faire de nombreuses et revenir très souvent sur ce qu'il avait déjà trouvé afin de comprendre comment était calculé cette dérivée. A force de recherche et en mettant en commun ce qu'il avait compris avec des personnes d'autres groupes, Nicolas a finalement réussit à comprendre comment se représentait la dérivée de la fonction softmax, c'est-à-dire une matrice jacobienne contenant l'ensemble des dérivées partielles. Le problème était ensuite de savoir comment l'utiliser. Sachant que la dérivée d'une fonction est égale à la somme de ses dérivées partielles, Nicolas a tout d'abord essayé de voir si cela fonctionnait de faire la somme des dérivées partielles dérivées par rapport à la même variable. Cependant, après quelques essais et des calculs littéraux il semblerait que la somme des dérivées partielles de softmax fasse toujours zéro. Il était impossible de faire un produit matriciel entre la jacobienne et la matrice d'erreur. Ces réflexions sur l'utilisation de la jacobienne était dû au fait que Nicolas avait oublié une partie de la méthodologie de la rétro-propagation du gradient. En réalité, la situation n'était en aucun cas différente de celle de la dérivée de la fonction sigmoïde qui est utilisé pour les couches cachées et qui était la seule fonction d'activation utilisée pour la première soutenance. Les dérivées partielles sont faites par rapport aux poids qui sont connectés au neurone sur lequel on a appliqué la fonction. On les utilise pour faire remonter la correction à travers chaque neurone, il faut donc les utiliser une à une pour faire remonter correctement celle-ci. Ainsi, un simple produit matriciel d'Hadamart suffisait.

Cependant, il était possible d'améliorer tout cela. Lors de tout le processus d'entraînement et d'utilisation du réseau de neurones, de nombreuses opérations sur des matrices sont effectuées. Or ces calculs sont coûteux et prennent du temps. Il est donc préférable d'en faire le moins possible. Lors des problèmes avec la dérivée de softmax, Nicolas avait demandé de l'aide sur le serveur 2023 sur Discord et quelqu'un lui avait proposé d'utiliser l'entropie croisée qui permettait d'ignorer la dérivée de softmax. Nicolas avait tout de même continué à chercher comment résoudre son problème mais maintenant que cela était fait, il s'y est intéressé. Utiliser l'entropie croisée pour le calcul du delta de la couche de sortie apporte un grand avantage qui est donc de ne pas avoir à utiliser la dérivée de la fonction softmax, tout en permettant d'avoir une rétro-propagation du gradient fonctionnelle. Cela permet donc d'éviter les opérations matricielles nécessaires au calcul de cette dérivée, ce qui permet d'optimiser une partie de l'entraînement.

Une fois la fonction softmax implémentée et la rétro-propagation du gradient adaptée, il a fallu faire des tests. Ces-derniers ont d'abord été réalisés avec la porte Xor car si le réseau n'arrivait plus à apprendre cette fonction logique, alors il n'était pas la peine d'espérer qu'il apprenne à

reconnaître des caractères. Dans un premier temps cela ne fonctionnait pas. La mise à jour des couches était calculé d'une manière qui fonctionnait avec un réseau ne possédant qu'une couche cachée et utilisant uniquement la fonction sigmoïde. La mise à jour des couches se faisait à l'aide d'une addition de matrice alors que le véritable calcul était une soustraction de matrice. Une fois ce problème réglé, le réseau réussissait à trouver les bons résultats dans la majorité des cas. Il était alors temps de l'adapter pour reconnaître des caractères. Nicolas a commencé par implémenter le système de mini-lots. A la première soutenance l'entraînement se faisait par lot, c'est-à-dire que toutes la base de données était passée d'un coup dans le réseau à chaque époque d'apprentissage. Il y a plusieurs méthodes de descente du gradient, ainsi pour justifier le choix de la descente en mini-lots, il est plus simple de présenter rapidement les 3 principales méthodes ainsi que certains de leurs avantages et inconvénients :

— **la descente stochastique** : les éléments de la base de données sont passés un à un à travers le réseau à chaque époque.
Avantages : Le modèle est mise à jour très souvent ce qui permet dans certains cas d'avoir un apprentissage plus rapide. La mise à jour fréquente entraîne l'apparition d'un bruit dans le modèle qui permet parfois d'éviter de se coincer dans un minimum local.
Inconvénients : Pour une base de données de n éléments, le modèle est mis à jour n fois par époque ce qui est coûteux en calcul. De plus le bruit dans le modèle va parfois entraîner des problèmes, comme par exemple rendre difficile la stabilisation sur le minimum.

— **la descente en lot** : les éléments de la base de données sont passés tous en même temps à travers le réseau à chaque époque.
Avantages : Les mises à jours du modèle sont moins fréquentes, ce qui est moins coûteux en calcul que la descente stochastique. De plus cela permet d'avoir un gradient d'erreur plus stable.
Inconvénients : Cette descente nécessite d'utiliser des matrices de très grande taille ce qui est coûteux en mémoire. Plus la base de données est grande, plus la taille des différentes matrices l'est. Ainsi, sur des bases de données très grande l'entraînement du réseau neuronal devient très lent. Ensuite, la convergence peut s'effectuer un peu trop tôt ce qui résulte en l'apparition d'erreurs dans la reconnaissance.

— **la descente en mini-lot** : cette descente combine les 2 précédentes. On forme des lots d'une taille constante n remplis de manière aléatoire. Ces lots sont ensuite passés un a un dans le réseau à chaque époque.
Avantages : La mise à jours du modèle est réalisé plus souvent que dans la descente en lot ce qui permet une convergence plus efficace qui évite les minimums locaux. Le coût en calcul est réduit ainsi que le coût mémoire vu que les éléments ne sont pas passés un à un et car

les matrices sont moins grandes.
Inconvénients : Des erreurs dans les calculs des delta peuvent s'accumuler. De plus une taille de lot mal choisie peut faire que le réseau ne convergera aucune valeur.

La descente en mini-lot possède de bons avantages et des inconvénients minimes. C'est pour cela que le réseau est passé d'une descente en lot à une descente en mini-lots. Cette descente n'a pas été extrêmement difficile à implémenter. La plus grosse difficulté était de réfléchir à une manière de l'implémenter de façon à pouvoir modifier le code facilement. En effet la segmentation n'était pas encore entièrement déboguer et ne possédait pas encore de composante contenant la longueur de la liste. L'implémentation à donc du être réalisée avec des constantes tout en gardant a l'esprit que plus tard il y aurait accès a des informations supplémentaires. La seule autre difficulté était de ne pas s'emmêler dans ce que représentait chaque index a cause des boucles imbriquées.

Une fois tout cela implémenté, il était temps de tester le réseau sur des caractères. Il a donc fallu commencer par réaliser une base de donnée. La segmentation était toujours entrain d'être déboguée. Nicolas a donc réalisé a la main une petite base de données contenant la moitié des lettres de l'alphabet, chaque image de caractère (16px par 16px) ayant été découpée a la main. Le réseau arrivait a apprendre Xor mais après ces changements le réseau n'arrivait pas a apprendre a reconnaître la moitié de l'alphabet. Pour trouver la raison pour laquelle cela ne fonctionnait, Nicolas a commencé par afficher les valeurs en sortie avant entraînement et après entraînement. Il a alors observé que les valeurs convergeaient presque toutes pour un seul et même caractère quelque soit l'image. Les calculs avaient déjà été vérifiés auparavant, l'erreur ne pouvait donc pas venir de la. Nicolas a donc fait des recherches sur ce qui pouvait causer des problèmes pour l'apprentissage des réseaux de neurones. Parmi les informations trouvées, une d'entre elle a attiré son attention. Une mauvaise initialisation des poids pouvait être la cause. Jusque la, Nicolas avait écouté les conseils d'un épitéen de Paris qui avait déjà aider un bon nombre de personne pour le réseau de neurone et qui avait conseillé d'initialiser les poids entre -0.75 et 0.75 pour les couches cachées et entre -0.6 et 0.6 pour la couche sortie. Nicolas décida donc de faire ses propres recherches. Il découvrit alors les fonctions de He, de Xavier (aussi appelée de Glorot) et de LeCun. Ces fonctions sont des fonctions faites pour initialiser les poids et qui sont très utilisées. Tout d'abord Nicolas a utilisé l'initialisation de LeCun mais après des recherches ultérieures, il a appris que l'initialisation de Xavier était était plus efficace avec les couches utilisant la fonction d'activation sigmoïde. Vu que la majorité qu'il peut y avoir plusieurs couches cachées mais une seule couche sortie et que les couches cachées utilisent sigmoïde, utiliser Xavier est plus pertinent.

L'implémentation de l'initialisation de Xavier a permis de résoudre les problèmes de convergence des valeurs a la sortie du réseau a ce stade. Peu de temps après, la segmentation et sauvola était

finalement débogué. Nicolas pu alors changer la fonction de création des mini-lots afin que celle-ci utilise la structure Liste de William plutôt que des constantes. La modification n'a pas causé de problèmes plus tard. Vu que ces parties étaient terminées il étai désormais possible de tester le réseau sur des textes segmentés et surtout de pouvoir utiliser une unique image pour la base de donnée qui est segmentée pour l'entraînement. Nicolas a donc modifié le main de test pour le réseau afin que tout cela se fasse. Le principale problème n'a pas été de modifier le main mais de changer le Makefile vu qu'il fallait générer de nouveaux fichiers objets pour que tous fonctionne. A force de temps il s'est rappelé comment il avait fait pour que cela fonctionne bien pour les autres fichiers et a donc pu résoudre les problèmes d'utilisation de fonction indéfinie. Ensuite, le réseau ne trouvait a nouveau plus les bons résultats. Le problème était le même qu'avant, c'est-a-dire une convergence pour une même valeur quel que soit les matrices passées au réseau. Vu qu'il avait déjà implémenté l'initialisation des poids de Xavier, Nicolas savait que le problème ne venait plus de la. Il ne voyait pas du tout comment faire, puis s'est fait la réflexion que le problème pouvait venir des paramètres du réseau comme par exemple le nombre de neurones par couche cachée et le nombre de ces couches. A ce moment le réseau contenait 3 couches cachées de 300 neurones chacune. Nicolas a alors diminué le nombre a 100 neurones et 2 couches cachées et le réseau fonctionnait de nouveau. Le nombre excessif de neurones provoquait un surentraînement qui faussait les résultats. Le réseau une bonne partie de l'alphabet mais a force de test en changeant les paramètres du réseau Nicolas a réussit a faire en sorte qu'il reconnaisse tout l'alphabet presque a chaque fois.

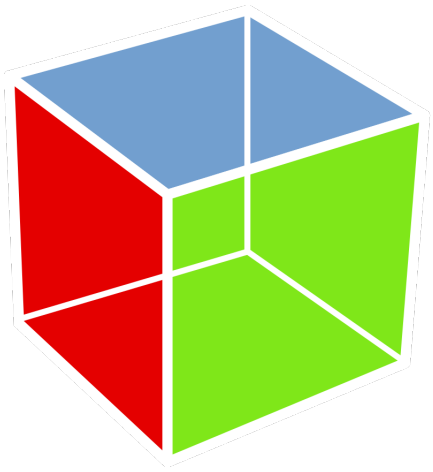
5.4 Interface utilisateur

5.4.1 Soutenance 1

Pour la première soutenance, nous ne sommes pas du tout occupé de l'interface graphique. Nous avons préféré nous concentré sur les parties les plus importantes, celles qui font tout le processus de reconnaissance de caractères. Nous comptons réaliser deux interfaces utilisateur différentes. L'une sera une interface graphique avec laquelle l'utilisateur pourra interagir à l'aide de boutons et de champs de texte, l'autre sera une interface console où l'utilisateur utilisera le logiciel par l'intermédiaire de commandes.

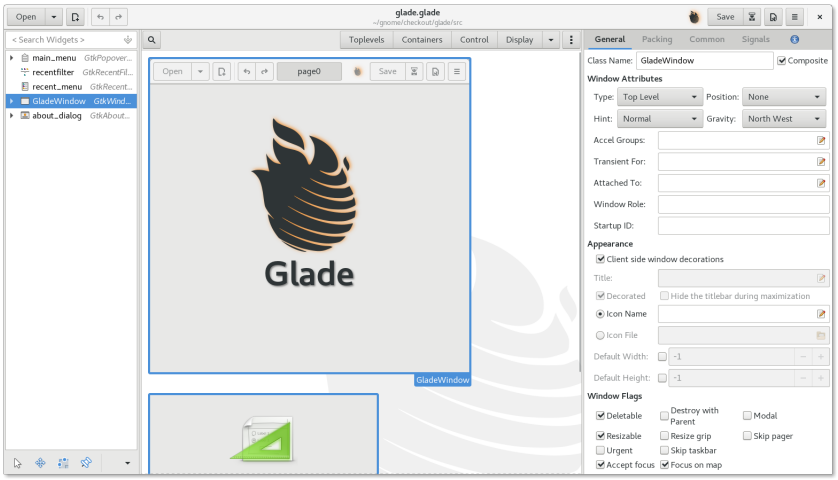
5.4.2 Soutenance 2

Cédric c'est donc chargé de l'interface pour la deuxième soutenance. Pour faire une interface graphique, Gtk est une boite à outils essentiels. Elle est composé de fonctions permettant la réalisation d'un tel projet.



Pour faire l'interface graphique, il a d'abord fallut comprendre le fonctionnement de Gtk ainsi que découvrir d'autres outils qui permettront d'atteindre nos fins. L'outil principal de la création de l'interface fut Glade.

Glade est un outil RAD (Rapid Application Development) permettant de faciliter le développement des interfaces utilisateur pour la boîte à outils GTK+ et l'environnement de bureau GNOME. Les interfaces utilisateur conçues dans Glade sont sauvegardées au format XML(langage de balisage extensible), et en utilisant l'objet GtkBuilder GTK+, elles peuvent être chargées dynamiquement par des applications si nécessaire.



Travailler avec ces deux outils permet donc de faciliter grandement l'implémentions d'interface graphique. En effet la construction de la structure du programme consistera en :

— Créer un document XML grâce à Glade.
— Récupérer les objets provenant de ce document dans la main.
— Créer des signaux connectant les évènements des objets à des fonctions

— Implémenter les fonctions des objets.

Cédric à commencer par réfléchir à la récupération de l'image à analyser et l'enregistrement du documents final. Pour cela il décida de laisser le choix à l'utilisateur entre :

— Taper l'adresse de l'image et de l'enregistrement.
— Choisir dans un gestionnaire de dossier.

Pour cela il fallut faire fonctionner deux objets ensemble :

— GtkEntry
— GtkFileChooserButton

Le premier permettant à l'utilisateur d'écrire un texte, le deuxième permettant d'ouvrir un gestionnaire de dossier. L'implémentions de certains évènement de ces deux objet à permit de faire un sorte que le FileChooserButton s'actualise en même temps que l'utilisateur tape l'adresse, et que l'adresse choisie dans le gestionnaire de dossier s'affiche dans l'Entry.

Ensuite il a fallut implémenter le button qui lance le programme principale. Pour cela, quand le button start est enclenché un signal permet le commencement du processus d'OCR. Les étapes sont les suivantes :

— Vérification de l'existence et du format des adresses.
— Binarisation de l'image reconnu
— Segmentation de l'image
— Reconnaissance de chaque caractères par le réseau de neurone
— Reconstitution du texte dans un fichier éditable.

Pour la réalisation de ces tâches, un minutieux état des lieux s'imposait, en effet la compréhension des fonctions de ses camarades fut essentiels pour leurs bonnes utilisation.

La segmentation va renvoyer une liste dynamique ou chaque élément contient la matrice d'un caractère ainsi que ses coordonnées dans l'image, ce format va être très utile lors de la reconstruction du texte. En effet après reconnaissance de chaque lettre par le réseau de neurone, si l'écart entre deux lettre (position en x) est supérieur à la taille de la police de la lettre, il faut sauter un espace, si la position en y change, il faut sauter une ligne.
Par la suite, il fut imaginer de pouvoir visualiser l'image sélectionner. Pour cela un gros problème rentre en jeu : l'actualisation de l'image. Pour cela, il fut penser de recréer un builder (un builder est

un objet auxiliaire qui lit les descriptions textuelles d'une interface utilisateur et instancie les objets décrits) à l'ouverture de la fenêtre, après l'initialisation de l'image. Cependant l'adresse de l'image est fixe, il fallut donc initialiser l'image de base avec une adresse que l'on connaissait, et copier l'image récupérer à cet emplacement. Pour cela l'intervention de SDL fut nécessaire, cependant, l'utilisation de SDL crée des conflit avec le fichier contenant le code de la segmentation.
De plus, le bouton exit n'est pas implémenté par défaut lors de l'ouverture d'une fenêtre avec gtk, c'est pourquoi ce bouton dut également être implémenter à la main. Pour fermer un fenêtre il suffit d'appeler la fonction gtk_main_quit() quand le bouton est pressé.

6 Avis personnels

6.1 Nicolas

Cela faisait quelques temps que j'avais commencé à m'intéresser aux réseaux de neurones et j'avais commencer à m'informer à leur propos durant les vacances ces d'été (seulement sur leur utilisation et les différents problèmes qui restreignent leur évolution néanmoins). M'occuper de cette partie dans le projet m'a permit d'apprendre de nombreuses choses, notamment comment ils fonctionnent. Cela n'a pas toujours été facile. Plusieurs fois je me suis retrouvé face à des problèmes que je n'arrivais pas à résoudre. Déboguer le réseau neuronal était souvent difficile car refaire les calculs à la main pour vérifier que le problème ne vient pas d'eux prend beaucoup de temps et n'est pas toujours évident à cause du nombre de valeurs qui peut parfois être important. Je trouve que le travail de recherche était plus important et plus profond que pour le projet de S2. De plus j'ai trouvé que la gestion du groupe était elle aussi plus présente. Là où l'année dernière ce n'était pas forcément grave pour les autres s'il manquait des morceaux, ce n'était pas aussi simple cette année. De nombreuses fois nous avons chacun eu besoin du travail des autres. Il fallait donc que j'organise le travail et que je donne des objectifs aux membres de mon groupe pour que nous progressions de manière efficace. Au début cela n'a pas été facile car tous le monde n'était pas forcément réactif et cela était un peu frustrant mais cela c'est très vite arrangé. A la fin la communication des différents problèmes allait très vite et cela permettait de mieux nous réorganiser. A mes yeux ce projet aura ainsi été plus formateur et intense que celui du S2.

6.2 Marius

Je ne m'étais jamais intéressé au traitement d'image avant ce projet. M'informer a propos de cet aspect du projet a été très intéressant. J'ai découvert et appris beaucoup de choses lors de mes recherches sur les méthodes de binarisation. Certaines plus obscures que d'autres mais il était

très intéressant de voir les nombreuses recherches qui ont été faites pour comparer les différentes méthodes et chercher la plus puissante. La découverte du C en même temps que ce projet et les nombreux travaux pratiques avec les devoirs d'algorithme ont été source de problèmes et de pression pendant la réalisation de ce projet. Le problème étant que la multitude de travaux en parallèle et le fait de se heurter a des difficultés sur certains n'aidait pas à la focalisation du à la facilite de bifurquer sur d'autres. J'ai personnellement apprécié ce projet et le groupe avec qui j'ai travaillé, j'ai fait la découverte de personnes que je ne connaissais jusqu'alors que de loin. Les nombreuses soirées à travailler avec William en salle machine peuvent en témoigner. Par rapport au projet de l'année dernière qui a été intéressant pour sa part de travail artistique ce projet a été beaucoup plus technique (le fait que les résultats soit visibles et aussi très appréciable pour la motivation). De plus nous avons travaille plus sur nos propres parties sur des choses très différentes pendant ce projet, a la différence du jeu ou nous faisions beaucoup plus les choses ensemble mêlant nos parties beaucoup plus dès le début. Ce projet a été très formateur sur le point de vue technique et par rapport aux recherches faites pour pouvoir trouver des méthodes performantes pour le traitement de l'image.

6.3 William

Avant de commencer ce projet, je n'étais pas au courant que ce genre de logiciel existait, il m'aurait bien été utile quelquefois dans ma vie. Et je n'avais aucune connaissance en C avant de commencer ce semestre, donc le début a était assez compliquer pour comprendre comment ce langage fonctionnait. Mais grâce autant qu'on nous donne chaque semaine, j'ai alors pu commencer ma partie doucement mais sûrement. Lorsque l'on m'a confié cette tâche j'ai d'abord cru que ça allait être assez facile ayant déjà ma petite idée, afin de segmenter tout le texte. J'ai très vite compris que ça n'allait pas être si facile que ça, ayant rencontré de nombreux problèmes qui m'ont mis dans tous les états. Contrairement à l'année dernière où j'ai pu travailler chez moi quasiment tout le temps, ici, je n'ai pas réussi à installer la SDL sur mon ordinateur, je ne pouvais pas tester mon code sur une vraie image pendant un moment. Je suis donc souvent resté après les cours à l'école afin de pouvoir continuer la segmentation et pouvoir voir ce que mes fonctions faisaient sur une image. Même si chacun avait leurs parties et qu'au début on travaillait surtout chacun de notre côté, on s'est très vite rendu compte que pour continuer dans nos tâches, il nous fallait partager et communiquer aux autres membres ce dont nous avions besoins et ce que nous avons fait afin que tout le monde puisse continuer leurs parties. Ce projet m'a permis tout d'abord d'améliorer mes notions de C, mais aussi énormément le travail d'équipe, partager les informations et être réactif aux demandes.J'ai préféré réaliser l'OCR, contrairement au projet de S2, qui consistait à créer un jeu vidéo. Car celui-ci est un logiciel que pourrait potentiellement utiliser dans la vie quotidienne

et qu'il pourrait être utile pour d'autres personnes.

6.4 Cédric

Avant le commencement du projet, j'étais assez anxieux, car l'OCR est un projet compliqué que nous avions a réaliser dans un temps limité et dans un langage de programmation que nous ne connaissions pas. Quelques problèmes de compréhension ont fait que l'interface graphique ne fût commencé qu'à partir de la première soutenance et que deux réseaux de neurones firent implémenté. En effet j'ai pris du plaisir à implémenter le réseau de neurone jusqu'au bout, mais je suis un peu déçu que mon travail ne sera finalement pas utilisé. Cependant, l'implémentation de l'interface graphique avec Gtk fut aussi très intéressante malgré de nombreux problème rencontré du à la recense des connaissances mis en pratique. D'un point de vue expérimental, je trouve que ce projet m'a apporté beaucoup. Il m'a permit de me rendre compte de la difficulté de la réalisation d'un tel projet ainsi que l'importance d'une organisation irréprochable dans le temps qui nous est impartie. En effet dans l'avenir nous n'aurons pas le choix des gens avec qui nous allons travailler, cependant les projets seront tout aussi compliqué à réaliser, c'est pourquoi il ne faut pas perdre en efficacité et donc mettre à l'oeuvre un bonne communication avec ses collègues.

7 Conclusion

Ce projet aura été pour nous une expérience enrichissante même si cela n'a pas toujours été facile. Nous n'avons certes pas réussi a obtenir un OCR parfaitement au point (il reste des warnings et des fuites de mémoires) mais nous avons fait de notre mieux pour avoir une chaîne de production complète pour la dernière soutenance.