



Rapport de projet OCR

William GROLLEAU (Chef de Projet)

Alexandre LEMONNIER

Jeanne MORIN

Pierre-olivier REY

Table des matières

1	Introduction	3
1.1	Equipe	4
1.1.1	William Grolleau	4
1.1.2	Jeanne Morin	4
1.1.3	Pierre-Olivier Rey	4
1.1.4	Alexandre Lemonnier	4
2	Répartition des tâches	5
2.1	William Grolleau	5
2.2	Jeanne Morin	5
2.3	Pierre-Olivier Rey	6
2.4	Alexandre Lemonnier	6
3	Pré-traitements	7
4	Segmentation	14
4.1	Segmentation des mots dans une ligne	14
4.2	Run Length Smoothing Algorithm et connected component labeling . . .	17
4.3	Algorithme final	22
5	Réseau de Neurones Artificiels	25
5.1	Introduction	25
5.2	Prémices du réseau de neurone	25
5.2.1	Réseau de Neuron entraîné au XOR	25
5.3	Réseau de Neurone final en C	26
5.3.1	Structure	26
5.3.2	Fonction d'Activation	26
5.3.3	Propagation par Lot	31
5.3.4	Entraînement	32
5.3.5	Au sein de l'OCR	34
6	Reconstruction du texte	35
7	Interface Graphique	37
7.1	Développement	37
7.2	Design	39
8	Architecture du projet	40
9	Bilan personnel	42
9.1	Pierre-Olivier	42
9.2	Jeanne Morin	42
9.3	Alexandre Lemonnier	43
9.4	William Grolleau	43
10	Conclusion	44

1 Introduction

Dans les projets comme dans la vie, on ne sait jamais vraiment ce qui nous attend. En S3 les élèves d'ÉPITA voient un projet imposé leur tomber dessus. Et pas n'importe lequel puisqu'il s'agit d'un OCR, ou ROC en français- reconnaissance optique de caractères. Ce terme désigne les procédés informatiques permettant la traduction d'images de textes imprimés ou dactylographiés en fichiers de texte.

Un OCR se sépare en 3 grandes parties :

- - la pré-analyse ou le pré-traitement, améliorant éventuellement la qualité de l'image et la transformant en noir et blanc.
- - la segmentation visant à fragmenter le texte sur l'image donnée, et retourner les caractères un à un.
- - la reconnaissance des caractères qui trouve quels sont les caractères contenus dans notre image. Par la suite, il reconstruira le texte contenu dans l'image sous un format texte.

Les groupes de projet sont composés de 4 personnes. Chacun des membres du groupe portera sa priorité lors du projet sur ces points. Les groupes ont eu trois mois pour réaliser, avec de grandes contraintes temporelles, un programme exécutable ayant une interface utilisateur. Il permet au dit utilisateur d'importer une photo contenant du texte et lui retourne dans un fichier texte, le texte reconnu sur la photo importée.

1.1 Equipe

1.1.1 William Grolleau

En tant, que chef de projet, mon devoir dans ce groupe est de donner une direction à notre travail et d'aider mes camarades pour leur faciliter la tâche. Comme dirait Rick Riordan : "Deadlines just aren't real to me until I'm staring one in the face." Afin de ne pas devoir subir cette confrontation, j'ai également dû jouer le rôle du méchant afin de rappeler que les deadlines approchent plus vite qu'on ne le pense et qu'il est important de s'y prendre tôt plutôt que tard.

1.1.2 Jeanne Morin

Ce paragraphe aurait pu être une recette de cuisine. Ce manque d'inspiration est tel mon manque d'inspiration face au projet OCR. Je n'avais aucune idée de ce qu'était la reconnaissance de caractère ou les réseaux de neurones avant le début de ce projet, et ces sujets ne m'inspiraient rien de bon ! Un mois et demi plus tard, plongé dans l'implémentation d'un réseau de neurones basique mais sournois, ma vision de l'OCR a totalement changé. Malgré mon manque d'enthousiasme au début du projet, je suis maintenant passionnée par le fonctionnement des réseaux de neurones et de l'apprentissage. Pour ce projet, j'utilise également mes qualités en présentation orale et en rédaction, pour la communication sur notre projet.

1.1.3 Pierre-Olivier Rey

J'aimerais pouvoir me considérer comme un "expert technique" car c'est ce que j'aime et ce que je sais faire. Ce projet est l'occasion pour moi de m'atteler à l'apprentissage du langage C et plus particulièrement aux manipulations plus ou moins orthodoxes de la mémoire. Travailler avec des gens ne faisant pas partie de mon cercle d'amis proche est aussi une bonne occasion de me préparer au monde professionnel où je ne travaillerais pas tout le temps avec mes amis.

1.1.4 Alexandre Lemonnier

Dans la famille des LEMONNIER, je voudrais le fils. Cet homme allie sagesse et détermination, mais n'apprécie guère les nuits courtes et les réveils de bon matin. Ce projet lui confèrera de nombreux points de caractéristiques : des points d'intelligence sur le langage C, des points de communication avec ses discussions au sein du groupe ainsi que lors des différentes soutenances, et enfin des points d'endurance mentale après les nombreuses erreurs rencontrées pendant une longue nuit. Grâce à ses nouvelles compétences, il pourra aider ses compagnons dans l'aventure afin de rétablir l'ordre dans le monde de l'OCR.

2 Répartition des tâches

2.1 William Grolleau

Dans le groupe, je suis en charge du pré-traitement d'un côté et de la segmentation de l'autre. J'ai choisi de m'occuper de ses parties, car ce sont celles restantes après le choix de chacun, cela ne me posait pas de problème. J'ai donc accepté de faire ses parties-là. Concernant le pré-traitement, j'ai essayé d'implémenter plusieurs types d'algorithmes différents afin d'améliorer ma connaissance en traitement d'images et de gérer le plus de cas possible. Bien sûr, il est impossible de gérer tous les cas existants.

J'étais également en charge d'une partie de la segmentation : je me suis occupé de la segmentation des paragraphes ainsi que celle des caractères. Les deux autres parties ont été effectuées par Alexandre. Je me suis également occupé de gérer le multicolonne ainsi que les images intégrées dans le texte.

De plus, j'ai implémenté la reconstruction du document. Une fois la segmentation effectuée il est primordial de sauvegarder le texte que notre OCR a réussi à générer dans un fichier extérieur.

Enfin, j'ai également oeuvré à la construction de l'architecture ainsi que la compréhension du projet. J'ai ajouté une documentation générée automatiquement par Doxygen. Je me suis chargé tout d'abord d'améliorer le makefile pour qu'il affiche les informations nécessaires à la compréhension des différentes règles qui le composent. Puis j'ai simplifié l'utilisation du makefile pour le rendre plus automatique.

2.2 Jeanne Morin

Pour ce projet je suis en charge, avec Pierre-Olivier, de la réalisation du réseau de neurones. Nous devons implémenter un réseau de neurones capable d'apprendre une fonction et l'entraîner. Enfin, nous devons être capable d'enregistrer les poids et les biais de ce réseau pour qu'il puisse réaliser des calculs une fois entraîné. Pour la première soutenance, nous avons implémenté un réseau de neurones, et il est possible de lui apprendre la fonction XOR. Pour cette soutenance finale, j'ai tout d'abord réalisé une étude des différentes fonctions d'activation.

Pour la classification des problèmes multi-classes (plus de deux sorties différentes), il est utile de comparer les activations utilisées pour la dernière couche du réseau. Certaines sont plus ou moins efficaces que d'autres. J'ai donc sélectionné les fonctions d'activation les plus utilisées pour les réseaux de neurones artificiels (ReLU, Sigmoid, Softmax, Tanh) et je les ai comparées en terme de rapidité et de capacité de convergence pendant l'entraînement et d'efficacité après entraînement. Pierre-Olivier et moi avons ainsi choisi la fonction la plus adaptée pour l'OCR.

Dans un second temps, j'ai réalisé les fonctions d'entraînement par lot (mini-batch), pour permettre d'optimiser l'entraînement. Le coût et la complexité de ce dernier sont ainsi amoindris. Enfin, j'ai implémenté la sauvegarde et la charge du réseau de neurone. Il nous permet de récupérer le réseau entraîné au sein du programme d'OCR. Je suis également chargée de préparer les présentations orales du groupe et de créer les supports.

2.3 Pierre-Olivier Rey

Je m'occuperai du réseau de neurones de ce projet. Dans une première partie je me suis attelé au fonctionnement principal du réseau de neurone, en particulier la création des structures, la gestion générale de la mémoire par rapport à ces structures, et à l'algorithme de propagation arrière. Après la première soutenance, je me suis occupé de la création et de la gestion d'une base d'exemples afin de finalement entraîner le réseau. Je me suis aussi occupé de faire le lien entre le réseau de neurones et la segmentation de William et Alexandre.

2.4 Alexandre Lemonnier

Mon rôle dans ce projet a été de réaliser la segmentation de l'image, avec William, pour y détecter les différentes zones de textes et leurs caractères. Au cours du projet, j'ai ainsi réalisé un algorithme pour récupérer les lignes et les mots de l'image. Cependant, cet algorithme ne fonctionne que pour certains type d'image : les images avec du texte dont les lignes des différents paragraphes sont alignées, et ne détecte pas les blocs de caractères. C'est pourquoi William et moi avons fusionné nos algorithmes. Ainsi, ma segmentation des lignes et des mots est directement utilisée dans les paragraphes récupérés par l'algorithme de William pour obtenir une segmentation la plus efficace possible.

J'ai aussi réalisé l'interface graphique pour permettre à l'utilisateur d'utiliser de façon simple et interactive notre OCR. Je me suis ainsi documenté sur la librairie GTK+ et ses différentes options. L'interface graphique est basée autour d'un éditeur de texte afin de pouvoir modifier le texte récupéré après l'application de l'OCR sur une image depuis l'interface elle-même.

3 Pré-traitements

Les pré-traitements d'une image sont tous les traitements effectués avant de pouvoir utiliser celle-ci dans différents algorithmes. Il existe de nombreux pré-traitements permettant d'améliorer la qualité de l'image, de réduire le bruit ou plus généralement de modifier l'image afin d'avoir un rendu optimal et utilisable aisément pour une segmentation. L'objectif de notre pré-traitement est de rendre l'image en noir et blanc en perdant le moins d'information et en gardant le texte le plus distinct possible. C'est William qui s'est chargé de cette partie du projet.

- **Niveau de gris de l'image :**

La première étape pour achever notre binarisation est de passer notre image en nuances de gris. En effet les couleurs de l'image ne vont pas nous servir pour déterminer le texte contenu par celle-ci. On pourrait même dire qu'ils vont nous déranger. En effet une image contient beaucoup d'informations et chaque pixel en contient beaucoup également. Les composantes rouges, bleu et vert, voire même, en fonction de la qualité de l'image les différents filtres appliqués sur l'image. Il est donc primordial de retirer les informations qui nous seront inutiles pour faciliter le traitement. La fonction est assez simple, elle prend simplement la moyenne des 3 composantes de la couleur, le rouge, le vert et le bleu. De nombreuses méthodes de niveau de gris sont applicables, ont pu notamment envisager l'utilisation de coefficients plutôt que de faire la moyenne, dans notre cas, la différence est négligeable puisque la couleur du texte ne doit pas influencer sa segmentation.

- **Augmentation de la luminosité :**

En général les images que notre pré-traitement doit gérer seront de mauvaise qualité, ainsi certaines lettres vont être étirées. Cela a pour conséquence que certains pixels qui étaient blancs vont devenir colorés, ainsi certaines lettres vont être collées alors qu'elles ne l'étaient pas. Une solution pour régler ce problème est d'augmenter la luminosité de l'image, ainsi l'égalisateur d'histogramme va accentuer le contraste, les pixels colorés créés à cause de l'étirement ou de la mauvaise qualité de l'image vont rester blancs. La binarisation n'aura donc pas de problème et ne va pas relier les différents caractères entre eux.

- **Égalisateur d'histogramme :**

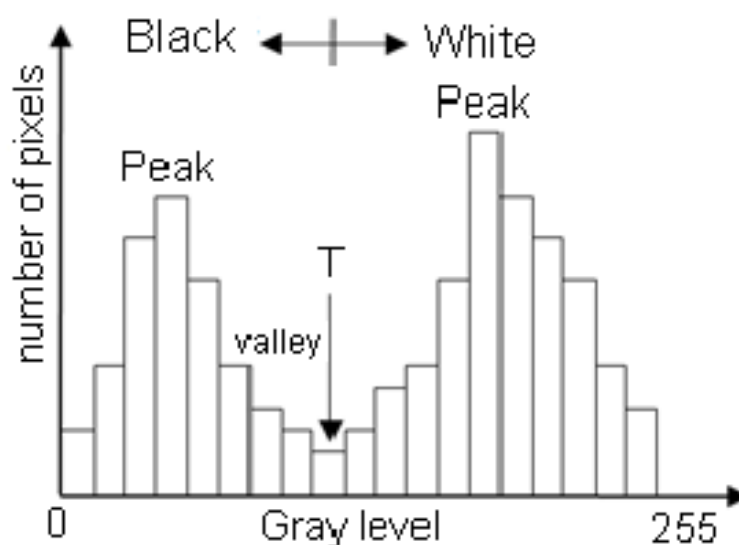
Maintenant que notre image est plus lumineuse, il est primordial d'augmenter le contraste entre les pixels sombres et clairs. En effet si l'on applique la binarisation sans augmenter le contraste, une grande partie des caractères sera coupée car ils seront trop clairs. L'algorithme que j'ai choisi d'utiliser est un égalisateur d'histogramme. Son fonctionnement est d'uniformiser la répartition des pixels sur l'intervalle $[0, 255]$. Le pixel le plus noir deviendra complètement noir, celui le plus blanc deviendra complètement blanc, les pixels seront ainsi répartis en fonction de leurs niveaux de gris. Cet algorithme va donc augmenter le contraste de l'image, pour faire simple, il rend le blanc plus blanc et le noir plus noir. Ce qui est exactement ce

qui est recherché, de cette manière les pixels grisâtres de notre texte vont devenir plus noirs et ne seront pas considérés comme blanc par la méthode d'Otsu.

● Binarisation de l'image :

La seconde étape est de transformer notre image en noir et blanc en essayant de perdre un minimum d'informations. J'ai choisi d'implémenter la méthode d'Otsu pour la binarisation puisqu'elle a l'avantage de calculer automatiquement le seuil entre le fond et la forme. Le principe de l'algorithme est de calculer pour tous les seuils possibles, donc 256 dans notre cas, l'écart type entre la classe inférieure à ce seuil et celle supérieure à ce seuil. On peut noter qu'il est possible de faire cela pour plus de 2 classes, mais dans notre cas nous n'avons besoin que de deux classes, une noire et une blanche. L'algorithme va donc chercher à minimiser la somme de ses écarts-type inter classe. L'idée derrière ses termes mathématiques est de trouver un seuil, dans lequel on peut séparer notre image en 2 classes, une claire et une foncée et ce seuil doit être placé de sorte à ce que chacune de ces classes comporte des pixels de couleurs rapprochées.

Comme le montre l'histogramme ci-dessous, le seuil se situe entre la classe claire, représentée par les pixels s'approchant des blancs et la classe foncée représentée par les pixels s'approchant du noir. L'avantage de cette méthode va d'être de calculer automatiquement notre seuil pour notre image et donc de nous donner une binarisation automatique pour toutes nos images. La réalité est bien évidemment tout autre. Les textes que nous allons chercher à rendre en noir et blanc n'auront pas un histogramme aussi avantageux, en effet ils seront majoritairement composés de nuances de blanc, le seuil sera donc très proche des noirs, cela aura pour cause de nous faire perdre une partie des informations que contient notre texte, les lettres rendues grisâtres par la mauvaise qualité de l'image seront donc perdues. C'est pour cela que j'ai choisi d'appliquer un égalisateur d'histogramme avant de passer à la méthode d'Otsu.



- **Image initial :**

En prenant comme image de base celle-ci dessous, nous allons regarder les différentes étapes du pré-traitement. En premier lieu voici l'image originale. On remarque qu'elle n'est pas des plus simples à gérer : en effet il y a plusieurs paragraphes avec du multicolonne. L'image contient également 2 éléments graphiques que nous ne souhaitons pas traiter .

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Etiam
viverra fringilla turpis id lobortis. Nunc
quis placerat justo, vitae porttitor orci.
Pellentesque habitant morbi tristique
senectus



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

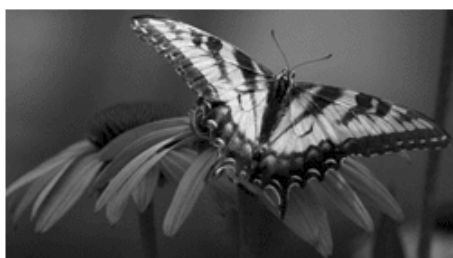
(a) Image initial

- **Image après transformation en niveau de gris :**

Une fois le niveau de gris effectué, l'image est évidemment uniquement composée de nuances de gris, cependant, on peut remarquer que l'image est très peu contrastée, les cheveux ne ressortent que très peu de l'image. Cela est dû au choix de notre fonction de calcul de l'intensité de pixel, elle prend la moyenne des trois composantes et n'applique pas les coefficients de luminescence, puisque notre but n'est pas de sublimer l'image, mais de récupérer les caractères noirs de celle-ci, notre choix de fonctions n'est donc pas problématique du moment qu'on égalise notre histogramme.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Etiam
viverra fringilla turpis id lobortis. Nunc
quis placerat justo, vitae porttitor orci.
Pellentesque habitant morbi tristique
senectus



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

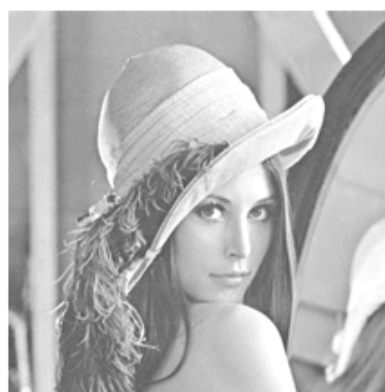
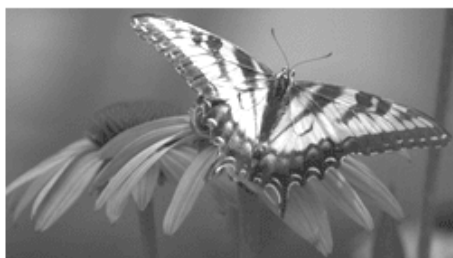
(b) Application du niveau de gris

● Image après augmentation de la luminosité :

On remarque que l'image est beaucoup plus claire, en effet on a augmenté la luminosité générale de l'image, l'image est donc plus pâle. Ceci facilitera la binarisation, la plupart des caractères ne seront donc pas fusionnés. Cependant il faut faire attention à ne pas trop augmenter la luminosité, sinon certains pixels risquent de devenir blancs alors qu'ils ne l'étaient pas.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Etiam
viverra fringilla turpis id lobortis. Nunc
quis placerat justo, vitae porttitor orci.
Pellentesque habitant morbi tristique
senectus



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

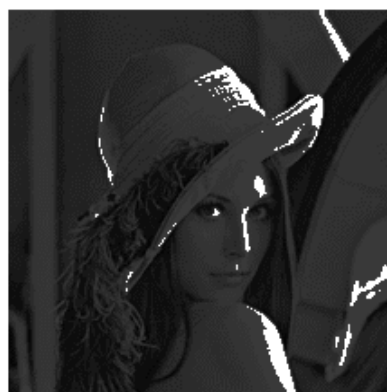
(c) Image après augmentation de la luminosité

- **Image après l'égalisation de l'histogramme :**

On constate immédiatement la différence, l'image contrastée fait ressortir les couleurs sombres. La binarisation sera beaucoup plus simple puisque la différence entre les pixels noirs et blanc est plus importante.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Etiam
viverra fringilla turpis id lobortis. Nunc
quis placerat justo, vitae porttitor orci.
Pellentesque habitant morbi tristique
senectus



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

(d) Application de l'égalisation d'histogramme

● Image binarisé :

La dernière étape est donc la binarisation de notre image. On utilisera la méthode d'Otsu, qui trouve un seuil pour déterminer les pixels qui deviendront noirs et ceux qui deviendront blancs. L'image ne change globalement pas, mais il est beaucoup plus simple de segmenter un texte sans avoir besoin de prendre en compte l'intensité de chaque pixel. Cependant, même en augmentant la luminosité, on remarque que les lettres 'se' sont 'c' du mot 'adipiscing' sont tout de même collées et auront de grandes chances d'être fusionnées dans la segmentation.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

**Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Etiam
viverra fringilla turpis id lobortis. Nunc
quis placerat justo, vitae porttitor orci.
Pellentesque habitant morbi tristique
senectus**



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

(e) Binarisation de l'image

4 Segmentation

La segmentation de l'image était l'objectif premier de William et d'Alexandre. L'idée de départ était de rechercher de son côté un algorithme qui pourrait convenir à la segmentation afin d'ensuite mettre en commun les recherches pour créer une solution optimale.

Alexandre a choisi l'alternative du XY-cut et William le rlsa combiné à un algorithme de connexion de composant. Chaque algorithme a ses avantages et ses défauts. Cependant, nous avons souhaité tous les deux mettre en place nos algorithmes, car, comme nous le verrons dans la conclusion de cette partie, les deux méthodes de segmentation se complètent très bien et nous permettront d'avoir une méthode efficace afin de fragmenter notre image en paragraphes, lignes, mots et caractères.

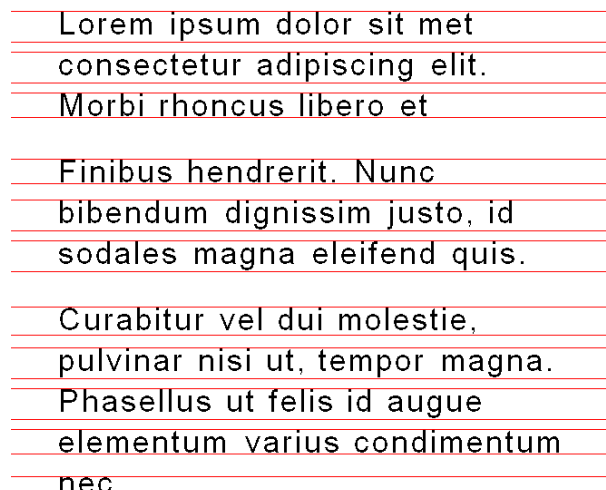
4.1 Segmentation des mots dans une ligne

L'objectif de la segmentation est de pouvoir récupérer chaque caractère d'un texte afin de le transmettre au réseau de neurones et ainsi le reconnaître.

Alexandre a décidé dans un premier temps, de concentrer la segmentation sur chaque lignes du texte et d'en extraire les caractères présents. L'inconvénient de ce principe est la détection des lignes qui dépend de l'alignement de celles-ci, ce qui n'est pas toujours le cas pour certain documents, comme les journaux. De plus, l'algorithme appliqué pour la segmentation des caractères ne permettait pas de détecter les caractères "liés" par un pixel noir (soit n'ayant pas d'espace blanc entre deux caractères), c'est pourquoi j'ai adapté mon algorithme afin de détecter chaque mot dans une ligne à la place des caractères et donnant la gestion de la segmentation des caractères à William grâce à son algorithme plus adapté.

- **Segmentation des lignes de l'image :**

Dans un premier temps, j'ai utilisé mon algorithme directement sur une image, composée uniquement de texte, afin d'observer ses résultats. En parcourant la totalité de l'image depuis le coin supérieur gauche, ligne par ligne, on regarde si le pixel correspond à un pixel blanc ou noir (0 pour blanc, 1 pour noir, après la binarisation), et si il est noir, alors cela signifie la rencontre avec un caractère. On trace alors une ligne horizontale à la ligne précédente de ce pixel, correspondant au début d'une ligne du texte. On continue de parcourir l'image, ligne par ligne, jusqu'à arriver à la fin d'une ligne de texte, c'est-à-dire une ligne où il n'y a aucun pixel noir. On trace alors une ligne représentant la fin de la ligne de texte et on recommence le même principe sur chaque nouvelle ligne de texte.



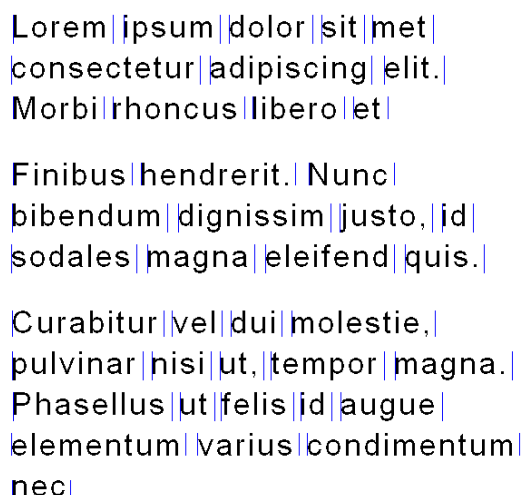
Lorem ipsum dolor sit met
 consectetur adipiscing elit.
 Morbi rhoncus libero et

 Finibus hendrerit. Nunc
 bibendum dignissim justo, id
 sodales magna eleifend quis.

 Curabitur vel dui molestie,
 pulvinar nisi ut, tempor magna.
 Phasellus ut felis id augue
 elementum varius condimentum
 nec

(f) Segmentation des lignes

- **Segmentation des mots de l'image :** Après avoir récupéré les coordonnées des lignes directement supérieure et inférieure d'une ligne de texte, l'algorithme sépare alors chaque mots dans la ligne de texte. Avant de réaliser la segmentation, l'algorithme calcule la taille moyenne de l'espace blanc entre deux caractères. Cette valeur permettra de différencier l'espace entre deux caractères et celui entre deux mots. Par le même principe que pour les lignes, mais cependant en parcourant la ligne de texte colonne par colonne, on repère la première colonne contenant un pixel noir, on trace sur la ligne précédente une ligne comme repère de début du mot et on continue de parcourir le ou les caractères jusqu'à tomber sur un nombre de colonnes blanches, sans pixel noir, supérieur à celui de la moyenne précédemment calculer, pour y tracer une nouvelle ligne correspondant à la fin du mot. On continue le parcourt de la ligne de texte en faisant le même principe si on rencontre un nouveau mot.



Lorem|ipsum|dolor|sit|met|
 consectetur|adipiscing|elit|
 Morbi|rhoncus|libero|et|

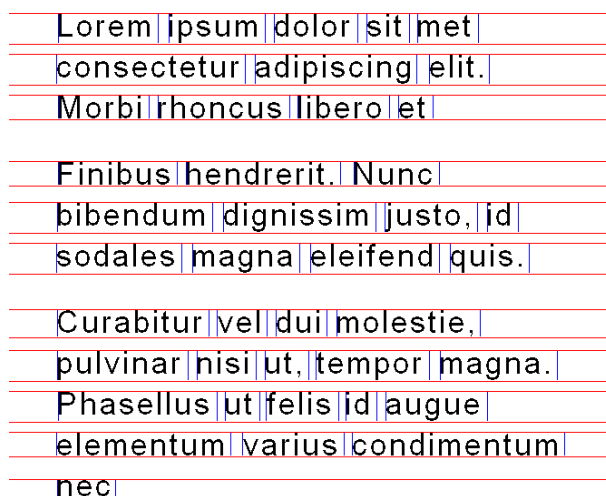
 Finibus|hendrerit.|Nunc|
 bibendum|dignissim|justo,|id|
 sodales|magna|eleifend|quis|

 Curabitur|vel|dui|molestie,|
 pulvinar|nisi|ut,|tempor|magna.|
 Phasellus|ut|felis|id|augue|
 elementum|varius|condimentum|
 nec|

(g) Segmentation des mots

● Résultat final :

Au final, on obtient une segmentation ligne/mots pour tout le texte :



```

Lorem ipsum dolor sit met
consectetur adipiscing elit.
Morbi rhoncus libero et
Finibus hendrerit. Nunc
bibendum dignissim justo, id
sodales magna eleifend quis.
Curabitur vel dui molestie,
pulvinar nisi ut, tempor magna.
Phasellus ut felis id augue
elementum varius condimentum
nec

```

(h) Segmentation des lignes et des mots

Ces résultats prouvaient que l'algorithme fonctionnait et qu'il permettaient de récupérer chaque lignes et mots de l'image. Cependant, il fallait maintenant pouvoir récupérer sous une certaine forme les lignes et les mots afin que le réseau de neurones puisse les reconnaître un par un ensuite.

Pour cela, Alexandre a cherché à implémenter une structure de matrice afin de stocker les pixels noirs des lignes ou des mots. Après plusieurs recherches sur l'implémentation des structures, des listes chaînées, de l'allocation dynamique et des pointeurs au début du projet, il a appris beaucoup de notions nécessaires à la programmation en C.

Malheureusement, il n'a pas été capable de réaliser entièrement la structure de matrice ainsi que celle d'une liste chaînée, qui aurait contenu toutes les matrices de lignes ou de mots au début du projet, afin de pouvoir y accéder, suites à de nombreuses erreurs rencontrées sur des notions nouvelles pour moi que je ne savais gérer. C'est ainsi que William, avec ses connaissances antérieures sur le langage C, a lui aussi implémenté une structure de matrice et de liste chaînée mais cette fois-ci fonctionnelle.

Ainsi, l'algorithme de segmentation à pu être utilisé, et peut à présent, à la place de tracer des lignes pour représenter la segmentation sur l'image, construire une liste chaînée contenant les différentes matrices des lignes d'un paragraphe et construire une liste chaînée contenant les différentes matrices des mots d'une ligne. Cela nous a permis de faire une reconstruction facile du texte après le traitement dans l'OCR, simplement avec des appels récursifs.

4.2 Run Length Smoothing Algorithm et connected component labeling

Cette partie a été réalisé par William. Pour la segmentation, je suis parti du principe que l'algorithme d'Alexandre allait avoir du mal à détecter les lignes d'un texte si 2 paragraphes sont l'un à coté de l'autre mais pas au même niveau, c'est-à-dire du multicolonne. Typiquement si les lignes ne sont pas alignées. Il va également rencontrer des problèmes si une image est sur le texte. C'est pour cela que je me suis tourné du côté de l'algorithme "Run Length Smoothing Algorithm" ou plus simplement rlsa.

Cependant avant d'appliquer le rlsa, je vérifie que le nombre de pixels en majorité dans mon image est blanc, s'ils sont noirs alors je considère que les lettres sont blanches et le fond noirs. Je choisis donc d'inverser le noir et le blanc. Un réseau de neurones reconnaît mieux les lettres si elles sont en noirs, il est donc important d'effectuer ce traitement.

● Run Length Smoothing Algorithm :

Le principe de cet algorithme, bien que simple reste très puissant pour détecter des blocs dans une image. L'algorithme va compter le nombre de pixels blancs compris entre deux pixels noirs consécutifs et en fonction d'un certain seuil, il va déterminer si ces pixels blancs doivent être remplacés par des pixels noirs ou laissés tels quels. Concrètement il va connecter les pixels noirs suffisamment proches entre eux afin de laisser apparaître des blocs. L'algorithme doit d'abord s'appliquer verticalement et horizontalement, on applique ensuite un "et" logique sur chaque pixel pour garder uniquement les pixels noirs présents à la fois sur le rlsa vertical et celui horizontal.

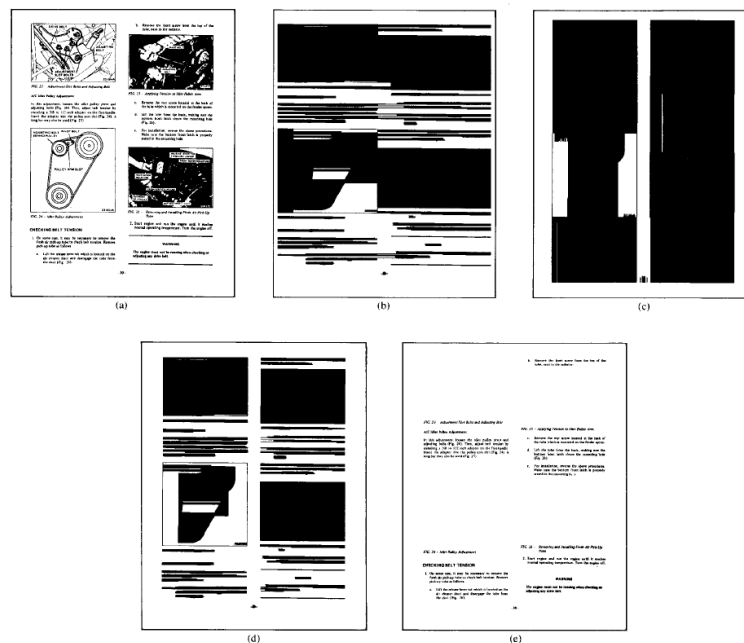
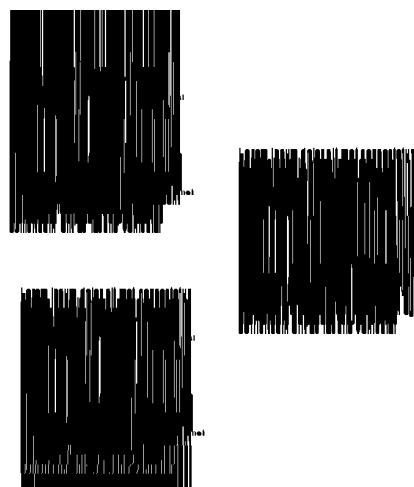


Figure 2 (a) Block segmentation example of a mixed text/image document, which here is the original digitized document. (b) and (c) Results of applying the RLSA in the horizontal and vertical directions. (d) Final result of block segmentation. (e) Results for blocks considered to be text data (class 1).

Cette image montre la capacité de l'algorithme à détecter les images et les paragraphes. Cependant l'image ci-dessus montre un cas assez avantageux pour l'algorithme. On peut remarquer que les blocs des textes sont très précis et très lisses. L'image permet à la sélection verticale d'être cohérente, car il y a un pré-traitement effectué sur celle-ci, on peut remarquer que les parties gauches et droites du document ont été encadrées au préalable pour améliorer l'algorithme. Cependant cet encadrement n'est pas généré par le *rlsa* mais par un algorithme extérieur et il a le désavantage de poser problème dans certains cas. Cependant sur une image avec seulement du texte, des lignes blanches peuvent apparaître en plein milieu d'une phrase. En effet par hasard, certains mots ou lettres peuvent être alignés et donc être ignorés par l'algorithme comme le montre l'image ci-dessous.



(i) Vertical and horizontal *rlsa*



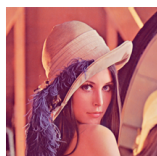
(j) Vertical *rlsa*

Ces deux images proviennent d'une application de l'algorithme *rlsa* dans notre projet sur un exemple de texte assez classique. On remarque que le texte n'est pas segmenté en paragraphe, mais plutôt en mot, et même caractère. Cela est dû principalement à l'application du *rlsa* verticalement, comme le montre la deuxième image. Certaines lignes blanches verticales existent dans notre texte, cela est dû au hasard, mais pose un problème pour détecter nos paragraphes. Il est donc primordial de trouver un moyen de régler ce problème avant d'effectuer une segmentation par paragraphe fonctionnel, qui est très importante pour la reconstruction du document et notamment la mise en page de celle-ci.

Pour régler ce problème il suffirait d'appliquer le rlsa verticalement et horizontalement sur la première image afin de lisser nos paragraphes. Pour nous convaincre que faire cela ne va pas tout simplement nous rendre n'importe quoi, il faut comprendre qu'appliquer 2 fois le rlsa avec les mêmes seuils ne va pas créer de nouveaux blocs. En l'occurrence lors de la deuxième application du rlsa nous allons appliquer un seuil bien plus faible, puisque nous souhaitons juste remplacer les quelques pixels blancs par des noirs. Voici ce que donne le résultat après avoir appliqué le rlsa verticalement et horizontalement sur la première image.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

(k) Texte original



(l) Application du rlsa

Nous avons maintenant réussi à détecter les paragraphes, cependant, nous ne sommes pas encore capables d'extraire un paragraphe de notre texte pour effectuer un traitement sur celui-ci. Nous devons également détecter les images et les retirer puisqu'elles sont inutiles. L'avantage du rlsa est de pouvoir détecter aisément des images comme celle montrée dans les exemples plus haut.

Pour déterminer si une partie détectée par le rlsa est une image, on calcule le ratio de pixel noir par rapport au nombre de pixels blancs. Si on se réfère à l'image binarisée de la partie sur le pré-traitement, on peut remarquer que l'image a un très gros ratio de pixel noir par rapport aux pixel blanc (environ 95% pour le papillon et pour Lena). Ainsi il est possible de retirer les images de notre liste de paragraphes pour pouvoir appliquer la segmentation dans de bonnes conditions.

- **Connexion de composant par étiquetage :**

Pour détecter nos paragraphes, William a utilisé un algorithme de connexion de composant par étiquetage. Le fonctionnement de ce type d'algorithme est de relier les pixels noirs directement liés par un même étiquetage afin de les placer dans un même ensemble, de cette manière tous les pixels collés entre eux appartiendront à la même classe. En appliquant cet algorithme sur l'image plus haut nous sommes donc capables de détecter les positions de chacun de nos paragraphes.

- **L'algorithme "two pass" :**

L'implémentation choisi est l'algorithme "two pass". Son principe comme son nom l'indique est d'étiqueter nos ensembles en passant 2 fois sur l'image. La première étape est donc le premier passage, dans celui-ci notre algorithme va affecter à chaque pixel, en commençant du coin haut gauche de l'image jusqu'au coin bas droit un entier. Pour chaque pixel, notre algorithme regarde si un pixel voisin directement à gauche ou au-dessus existe, s'il n'existe pas, alors nous avons découvert un nouvel ensemble et nous incrémentons notre entier et l'appliquons à notre pixel. Dans le cas où un pixel existe en haut ou en bas, on assigne la valeur de ce pixel voisin au pixel actuel, en effet les 2 pixels sont liées. Maintenant supposons que notre pixel a, à la fois un voisin à gauche et au-dessus, si les 2 valeurs de ses pixels sont les mêmes pas de problèmes, cependant si elles sont différentes alors on assigne la valeur la plus basse à notre pixel et a dit que nos 2 valeurs appartiennent aux mêmes ensembles.

Cependant dire que ces 2 valeurs, et donc par extension que ses 2 pixels sont du même ensemble, est plus simple à dire qu'à faire. Pour cela, nous allons devoir construire une classe d'équivalence entre nos différents ensembles. Lors du projet de l'année dernière, pour pouvoir appliquer l'algorithme de Kruskal, William avait déjà réalisé un algorithme répondant à notre besoin actuel. Il a donc choisi de l'utiliser de nouveau dans notre projet. L'implémentation choisie est donc celle de l'union-find. Cet algorithme possède 2 fonctions : une fonction de recherche dans un ensemble et une fonction d'union d'ensemble. Dans notre cas, assigner ces 2 pixels aux mêmes ensembles revient à créer un ensemble pour toutes les étiquettes et à faire l'union de 2 ensembles si le cas cité plus haut se présente.

Après notre premier passage nous allons donc avoir une matrice avec des étiquettes pour chaque pixel, certaines étiquettes seront contenues dans un même ensemble, qui est défini par la fonction Union. Le but du deuxième passage est de fusionner les pixels d'un même ensemble pour attribuer une seule étiquette par bloc d'information. Voici comment fonctionne le deuxième passage : à chaque pixel rencontré on s'intéresse à l'étiquette de notre pixel, à l'aide de la fonction find, on recherche si notre étiquette appartient à une autre ensemble ayant une étiquette plus petit que celle actuelle. Si c'est le cas, on remplace l'étiquette actuelle par la nouvelle, sinon on ne change rien. De cette manière, toutes nos étiquettes, appartenant aux mêmes ensembles, vont devenir une unique étiquette représentant notre bloc d'information. Voilà le résultat de cette fonction appliquée directement à notre texte.

Lorem ipsum
 Lorem ipsum dolor sit met consectetur
 adipiscing elit. Morbi rhoncus libero et
 finibus hendrerit. Nunc bibendum
 dignissim justo, id sodales magna
 eleifend quis.
 Curabitur vel dui molestie, pulvinar nisi
 ut, tempor magna. Phasellus ut felis id
 augue elementum varius condimentum
 nec
 Massa. Nunc faucibus dignissim tellus
 fringilla auctor. Maecenas eu vulputate
 quam, nec gravida ipsum. Nam non
 dictum elit. Suspendisse vel tortor non
 quam auctor iaculis dignissim ac velit.
 Suspendisse odio turpis, suscipit sit amet
 eleifend sed, tincidunt ac ex. Praesent
 dui ex, ultrices quis quam eget,
 elementum vulputate turpis. Donec
 sollicitudin elit ac iaculis vehicula.
 Lorem ipsum dolor sit met consectetur
 adipiscing elit. Morbi rhoncus libero et
 finibus hendrerit. Nunc bibendum
 dignissim justo, id sodales magna
 eleifend quis.
 Curabitur vel dui molestie, pulvinar nisi
 ut, tempor magna. Phasellus ut felis id
 augue elementum varius condimentum
 nec
 Massa. Nunc faucibus dignissim tellus
 fringilla auctor. Maecenas eu vulputate
 quam, nec gravida ipsum. Nam non
 dictum elit. Suspendisse vel tortor non
 quam auctor iaculis dignissim ac velit.
 Suspendisse odio turpis, suscipit sit amet
 eleifend sed, tincidunt ac ex. Praesent
 dui ex, ultrices quis quam eget,
 elementum vulputate turpis. Donec
 sollicitudin elit ac iaculis vehicula.

(m) Représentation de two pass sur un texte

Nous sommes donc capable de découper le texte en paragraphes, d'extraire chaque paragraphe de l'image pour continuer mon traitement. La détection de ligne est trivial, il suffit d'appliquer un rlsa horizontalement avec un seuil maximal pour détecter nos lignes. Nous appliquons ensuite le two pass pour séparer les lignes et recommençons pour les mots. Ici, appliquer le rlsa verticalement n'est pas une bonne solution, en effet la distance entre 2 mots varie grandement en fonction de la police et de la taille de police. Mon algorithme n'est donc pas compétent pour détecter les mots en toutes circonstances. Pour les caractères, il suffit d'appliquer directement le two pass, on remarque qu'on peut directement extraire nos caractères de notre image sans passer par la segmentation en paragraphe et ligne, cependant cela rendrait la reconstruction du document plus délicate.

4.3 Algorithme final

Pour l'algorithme final, il n'est pas possible de trancher entre celui fait par Alexandre et celui de William, en effet aucun des deux ne permet une segmentation cohérente en toute circonstance. Cependant, il est très intéressant de combiner ces algorithmes.

En effet, celui d'Alexandre va avoir du mal à détecter les images et les paragraphes d'un texte, là où l'algorithme de William n'a aucun problème. Par conséquent la détection de paragraphe sera du ressort de l'algorithme de William. La détection de ligne est aisément faite par nos deux algorithmes, nous choisiront celui avec la plus faible complexité et donc le plus faible temps de calcul.

Pour la segmentation des mots, le rlsa est inefficace à cause du seuil qui serait difficile à définir. Face à ce problème, l'algorithme d'Alexandre devient intéressant. Il n'a aucune difficulté à trouver les mots puisque chaque mot sera forcément séparé d'un espace, la segmentation en mot lui revient donc.

Pour la segmentation en caractères, le two pass va avoir un avantage important sur le XY-cut. En effet dans le cas de certaines polices de caractères, il est possible qu'une lettre passe par-dessus ou par dessous une autre. Cela rend la détection des lettres impossible par l'algorithme d'Alexandre. Cependant le two pass n'aura aucun problème à détecter les caractères, c'est donc l'algorithme de William qui sera en charge de cette étape.

Pour résumer, la solution est de combiner les deux algorithmes. C'est donc ce que nous avons fait. Tout d'abord le rlsa détecte les différents paragraphes et images dans notre texte. Ensuite on retire toutes les images qui ont été détectées par le rlsa. On segmente le texte en ligne puis en mots et enfin en caractère. Ainsi nous avons une segmentation qui nous permet de reconstruire le document beaucoup plus facilement puisque que l'on a réussi à détecter les 4 composants du texte. En voici les résultats sur l'image donnée en exemple pour le prétraitement :

Dans un premier temps nous séparons l'image en plusieurs paragraphes, nous retirons les images et nous créons notre liste comportant les différents paragraphes. Dans l'image du pré-traitement, le premier paragraphe est celui ci-dessous.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id lobortis. Nunc quis placerat justo, vitae porttitor orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis venenatis tortor quis tincidunt pellentesque. Aenean semper nulla vitae luctus euismod. Mauris a nulla tristique, viverra odio et, maximus libero. Nulla at hendrerit quam, a tincidunt tortor.

(n) paragraphe

Nous appliquons l'algorithme pour séparer les différentes lignes et récupérer la première. Il est possible que l'algorithme garde des parties blanches inutiles sur les lignes, notamment si la ligne est plus petite que le paragraphe. Cela ne pose aucun problème puisque l'algorithme de séparation de caractère va retirer toutes les parties blanches inutiles.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam viverra fringilla turpis id

(o) ligne

Nous appliquons l'algorithme de séparation des mots, afin de séparer les différents mots de notre image. L'algorithme fonctionnant sur une moyenne, il est possible que certains caractères soient séparés et qu'à l'inverse certains mots ne le soient pas. Cela reste assez rare et pose réellement des problèmes quand le texte est dans une très petite police.

L o r e m

(p) mot

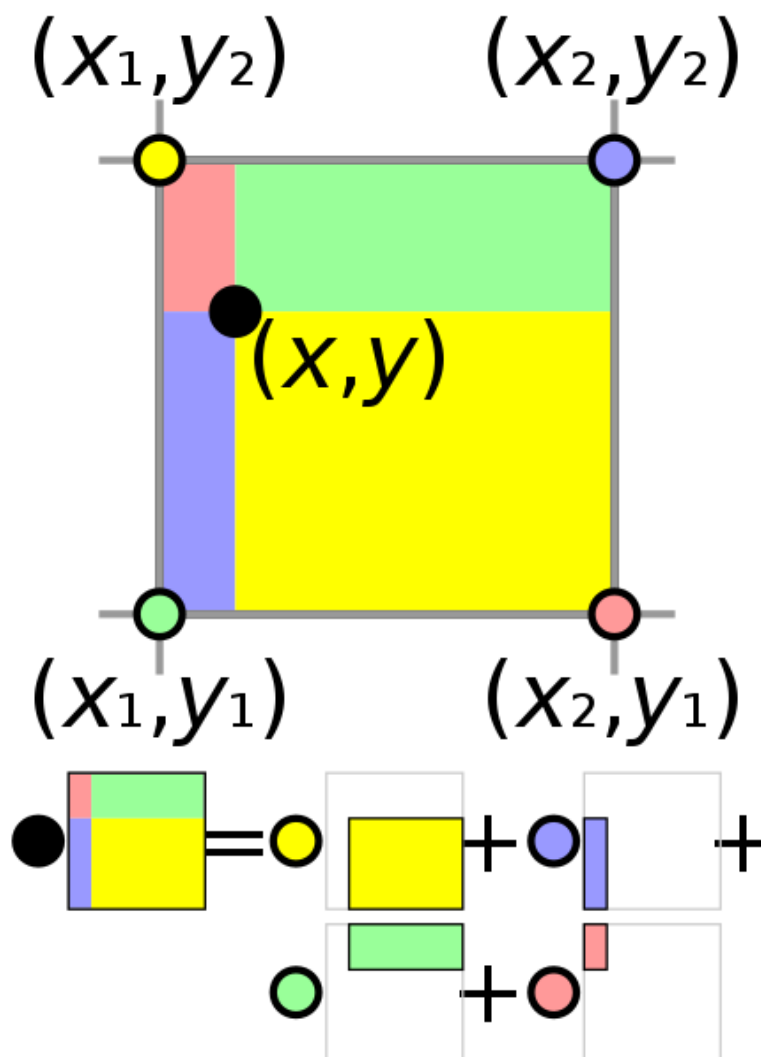
Enfin nous appliquons le dernier algorithme pour séparer les caractères entre eux. Comme expliquer plus haut les caractères sont contenu dans la plus petite matrice possible, le blanc inutile est donc complètement retiré. Un problème peut se présenter si la police est trop petite ou que la qualité est trop mauvaise. Dans ces cas-là, il est possible que certains caractères soient fusionnés, mais une grande majorité ne devrait pas l'être car seulement certaines combinaisons de caractères vont poser problème. Dans l'exemple ci-dessous certains caractères peuvent sembler flous : c'est le cas car nous avons forcé la taille de ceux-ci afin qu'ils soient visibles sur le rapport de soutenance. Cependant les caractères ont une taille différente, le "L" par exemple est plus grand que les autres lettres car c'est une majuscule, dans notre projet, une modification de la taille du caractère est effectuée pour qu'ils soient tous normalisés.

L o r e m

Cependant, la taille des caractères récupéré par l'algorithme de segmentation n'était pas la même selon la résolution de l'image et la police des caractères, rendant la détection des caractères plus complexes par le réseau de neurones.

Alexandre a alors cherché et implémenté une méthode pour redimensionner les matrices des caractères afin qu'elles fassent la même taille (28*28), tout en ayant des caractères qui remplissent la matrice complètement peu importe leur police initiale. La méthode implémenté est la méthode d'interpolation bilinéaire, qui consiste à déterminer la nouvelle position de chaque éléments de la matrice initiale dans la matrice redimensionnée. On projette l'élément de la matrice initiale dans la matrice redimensionnée en calculant la somme des produits de la valeur des coins autour du point projeté et de la surface "partielle" diagonalement opposée au point crée par ces points. Si cette somme est supérieur à 0.5, alors l'élément projeté est un pixel noir, sinon c'est un pixel blanc.

Voici un schéma plus représentatif de la situation :



(v) Visualisation geometric de l'interpolation bilinéaire

5 Réseau de Neurones Artificiels

Le réseau de neurone a été créé par Pierre-Olivier et Jeanne. Un réseau de neurone entraîné au XOR puis un réseau de neurone entraîné à la reconnaissance de caractère ont été implémentés.

5.1 Introduction

Les réseaux de neurones artificiels sont inspirés du fonctionnement des réseaux de neurones de notre cerveau. Chaque neurone est connecté aux autres grâce à ses dendrites et ses axones. Ses dendrites vont lui permettre de recevoir de l'information par des potentiel d'action (signal électrique). Par ses axones, il pourra lui même propager ce signal. Pour évaluer l'importance de l'information, le neurone va prendre en compte l'amplitude du signal et le nombre de ces influx. Ainsi un signal peu "intense", composé d'un seul influx avec une petite amplitude, ne sera pas transmis.

Pour retenir une information, nos neurones s'organisent en réseau. Ces réseaux pourront persister un certains temps, si un signal passe à nouveau dans ce réseau.

De ce modèle découle les réseau neuronaux artificiels. On peut aujourd'hui programmer, en s'inspirant des réseaux de neurones réels, un algorithme capable d'apprendre. Cette structure est composé de plusieurs couche de neurones, connecté entre elle. Nous allons programmer cette dernière pour apprendre. Suite à cela, nous lui "apprendront" des choses, qu'elle retiendra grâce aux calculs complexe qui la compose. Impossible de savoir exactement par quelle "réflexion" le réseau passe. Mais, s'il est bien entraîné à résoudre un problème, il sera capable de le faire avec n'importe quelle situation donnée.

5.2 Prémices du réseau de neurone

Les concepts de réseau de neurone et d'apprentissage sont des notions particulièrement compliquées à assimiler. Jeanne et Pierre-Olivier ont donc décidé, dans un premier temps, de créer chacun de leur côté un XOR en python. Ces deux programmes ont été réalisés avec des structures et des principes différents. Ainsi, Jeanne et Pierre-Olivier ont pu enrichir leurs connaissances sur les réseau de neurones en comparant leurs travaux. Jeanne a de son coté cherché à comprendre le coté purement concret et à appris à implémenter les réseaux neuronaux artificiels à travers des implémentations en plusieurs langages, disponible sur Internet. Pierre-Olivier à lui eu une toute autre approche. Il s'est confronté à des documents d'explication purement mathématiques pour comprendre la théorie derrière ces concepts.

5.2.1 Réseau de Neuron entraîné au XOR

Pour créer notre réseau de neurones en C, nous avons dans un premier temps créé des structures qui nous seraient nécessaires par la suite. Des matrices, des Layers, représentants les couches du réseau et des neuNet, des réseaux de neurones composés de couche. Pour l'entraîner au XOR nous avons choisi la fonction d'activation sigmoïde.

Jeanne a implémenté l'initialisation du réseau et des Layers ainsi que la fonction forward propagation. De son coté, Pierre-Olivier a réalisé la fonction de propagation arrière.

Notre but était de développer le réseau le plus flexible possible pour faciliter notre travail lors de la suite du projet. Ainsi, aucun éléments de notre réseau n'était hard codé. Nous

pouvions ainsi créer un réseau avec autant de couche et de neurone par couche que nous voulions.

5.3 Réseau de Neurone final en C

5.3.1 Structure

Nous avons choisis de structurer notre réseau comme suit : Une structure `neuNet` composé d'une liste de layer et d'un nombre de Layer (longueur de la liste de Layer). Chaque structure Layer est composé de 6 matrices : poids, biais, poidsbatch, biaisbatch, values et output. Enfin, elle possède un nombre de neurone. Ainsi, les opérations durant la propagation avant et arrière sont des opérations matricielles, ce qui simplifie nos calculs. La force de notre réseau de neurone réside dans sa flexibilité. Lors de la première phase, nous avons implémentés un réseau ayant un nombre de couche et un nombre de neurone par couche modulables.

5.3.2 Fonction d'Activation

La fonction d'activation a été inspirée du "potentiel d'action", un phénomène électrique entre deux neurones biologiques. Un neurone réel (comme ceux de notre cerveau) reçoit des signaux d'autres neurones à travers les dendrites. Le poids associé à une dendrite, appelé poids synaptique, est multiplié par le signal entrant. Les signaux des dendrites sont accumulés dans le corps cellulaire et si la force du signal résultant dépasse un certain seuil, le neurone transmet le message à l'axone. Sinon, le signal est tué par le neurone et ne se propage pas davantage. Le potentiel d'action est donc la variation de la force du signal indiquant si la communication doit se faire ou non. La fonction d'activation prend la décision de transmettre ou non le signal. Dans ce cas, il s'agit d'une fonction simple avec un seul paramètre : le seuil. Maintenant, lorsque nous apprenons quelque chose de nouveau, le seuil et la probabilité de connexion (appelée poids synaptique) de certains neurones changent. Cela crée de nouvelles connexions entre les neurones, ce qui permet au cerveau d'apprendre de nouvelles choses.

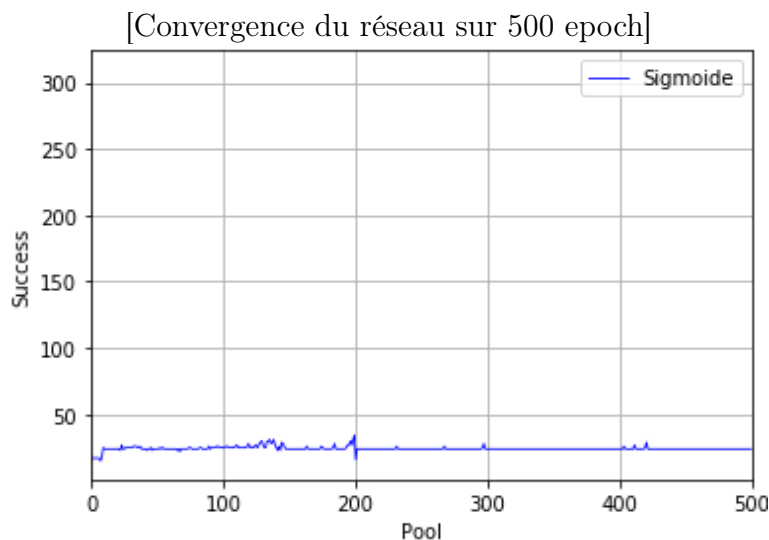
L'activation a pour but de transformer le signal de manière à obtenir une valeur de sortie à partir de transformations complexes entre les entrées. Les fonctions d'activations utilisés dans les réseaux de neurones artificiels sont généralement non linéaires : au-delà d'une couche de neurones, l'application récurrente d'une même fonction d'activation linéaire n'aura plus aucun impact sur le résultat. Ainsi, pour résoudre des problèmes complexes tels que la reconnaissance de caractère, l'utilisation de fonctions non linéaires est obligatoire. Autre utilité des fonctions d'activation non linéaires : elles réduisent la valeur de sortie d'un neurone, pouvant atteindre de grandes valeurs. L'utilisation d'une fonction linéaire, ne modifiant pas la sortie, les valeurs des données transmises de neurones en neurones peuvent devenir de plus en plus grandes et rendant les calculs beaucoup plus complexes. Cela peut mener à des phénomènes comme l'explosion du gradient, qui vont faire diverger le réseau pendant l'entraînement. Les fonctions d'activation non linéaires vont, elles, transformer les valeurs de sortie d'un neurone en une valeur généralement entre 0 et 1, voire en une distribution de probabilité.

Habituellement pour la reconnaissance de caractère, on choisira la fonction Softmax comme fonction d'activation. Nous avons quand même souhaitez, au cours de notre projet, comparer différentes fonction d'activation pour constater la plus efficace pour notre problème. Jeanne a réalisé l'étude de plusieurs fonction d'activation non linéaires, présentée ci dessous.

Sigmoïde :

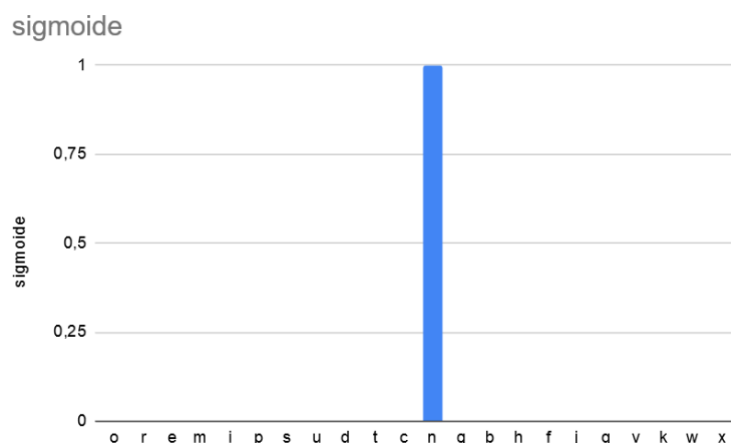
$$h(x) = \frac{1}{1 + e^{-x}}$$

Sigmoïde est la fonction d'activation d'un réseau de neurone par excellence. Le but premier de la fonction est de réduire la valeur d'entrée dans un intervalle de 0 à 1. Si la valeur en entrée est un très grand nombre positif, la fonction convertira cette valeur en une valeur de 1. A l'inverse, si la valeur en entrée est un très grand nombre négatif, la fonction convertira cette valeur en un 0. Sigmoïde est une fonction très efficace dans la classification de problème binaire (est-ce un a, oui ou non ? Les valeurs correspondent telles à celles d'un XOR ?..). Malheureusement, comme nous allons le voir dans les données présentées, elle n'est pas efficace sur les problèmes de classification plus complexes, comme une reconnaissance de caractère.



Ci-dessus, la courbe moyenne des succès par epoch, sur 500 epochs. On peut constater que le réseau converge autour de 30 succès, ce qui est très mauvais, et montre qu'il n'aura sûrement aucune qualités pour reconnaître des caractères.

[Lettres reconnues par le réseau précédemment entraîné]



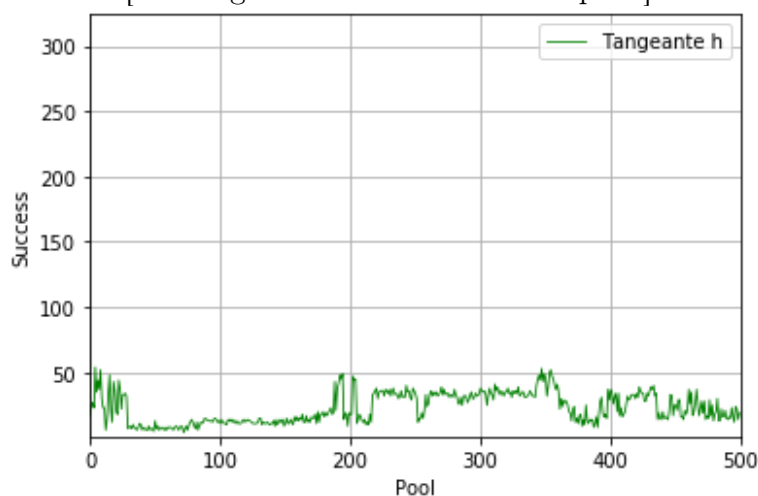
Suite à cette entraînement, une phrase contenant 24 caractères différents ont été envoyés au réseau. Ci-dessus, un histogramme représentant les lettres reconnus par le réseau. On s'aperçoit que le réseau n'a reconnu qu'une seule lettre : le n.

Tangente hyperbolique :

$$h(x) = \frac{\sinh}{\cosh}$$

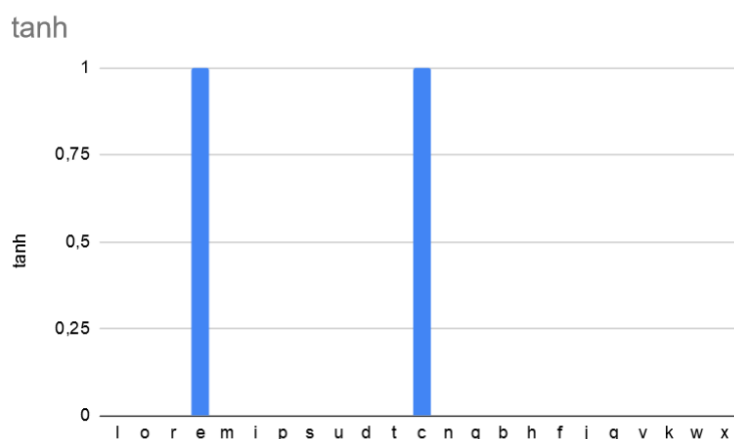
Tangente hyperbolique ou Tanh ressemble à la fonction Sigmoidé. Mais Tanh diffère de Sigmoidé en un point : elle produit un résultat compris entre -1 et 1. eAinsi, elle présente un avantage sur Sigmoidé : Tanh est centrée sur 0. Les grandes entrées négatives tendent vers -1 et les grandes entrées positives tendent vers 1. Ainsi, on pâlie au risque de saturation de neurone que peut causer Sigmoidé.

[Convergence du réseau sur 500 epoch]



Avec la fonction tangente hyperbolique, on constate que le réseau converge plus vite mais son apprentissage reste très laborieux.

[Lettres reconnues par le réseau précédemment entraînés]

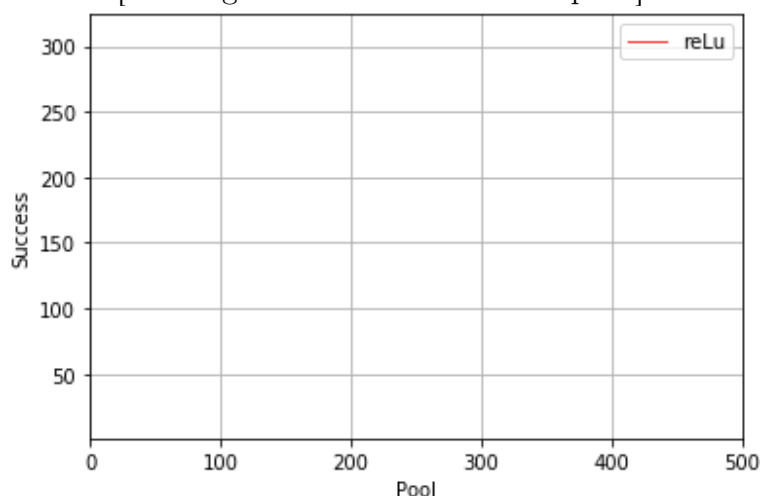


Cette fois le réseau est capable de reconnaître 2 lettres : le e et le c.

ReLU : $h(x) = x$ si $x > 0$ sinon $h(x) = 0$

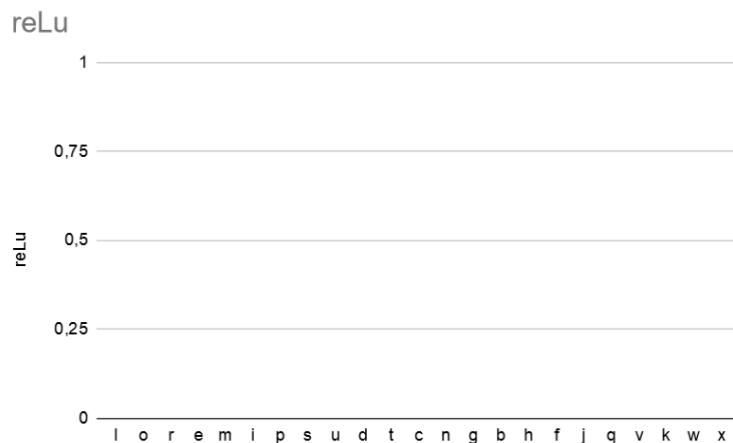
La fonction ReLU est aujourd'hui la fonction d'activation la plus populaire. Elle permet un entraînement plus rapide par rapport aux fonctions sigmoïde et tangente hyperbolique. Point négatif : la fonction ramenant les valeurs entre 0 et plus infini, un phénomène de "Dying neuron" peut apparaître. En effet, les neurones ne donnant que des valeurs de pré-activation inférieur à 0 auront une valeur de sortie, suite à son activation, égale à 0. Ces derniers n'auront donc plus aucun impact sur les calculs du réseau. Dans certains cas, la presque totalité du réseau peut mourir pendant l'entraînement. Il existe de nombreuses variantes de cette fonction, notamment Leaky ReLu, qui permettent d'éviter le plus possible ce phénomène de neurone mourant.

[Convergence du réseau sur 500 epoch]



Lors des tests, la fonction reLU s'est montrée inefficace, pour une moyenne de 0 succès sur 500 epoch. Ces résultats semblaient étrange, mais Jeanne a quand même décidé de les présenter.

[Lettres reconnues par le réseau précédemment entraînés]-

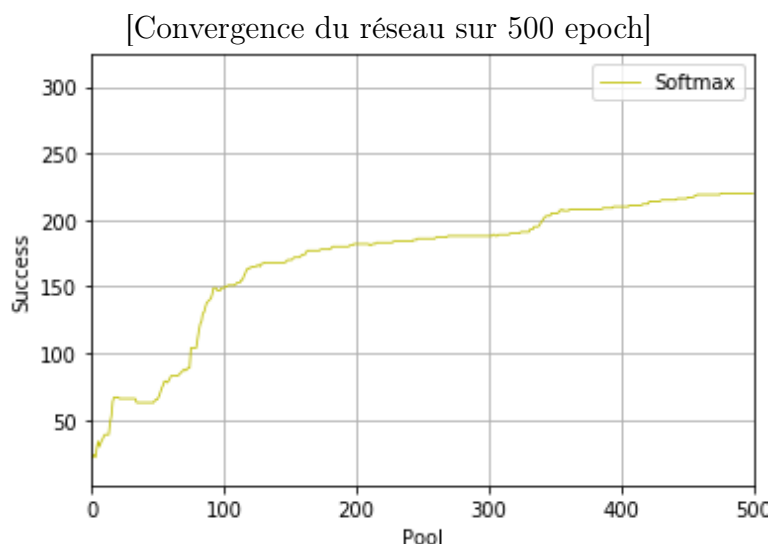


Ainsi, aucunes lettres n'a été apprise par le réseau.

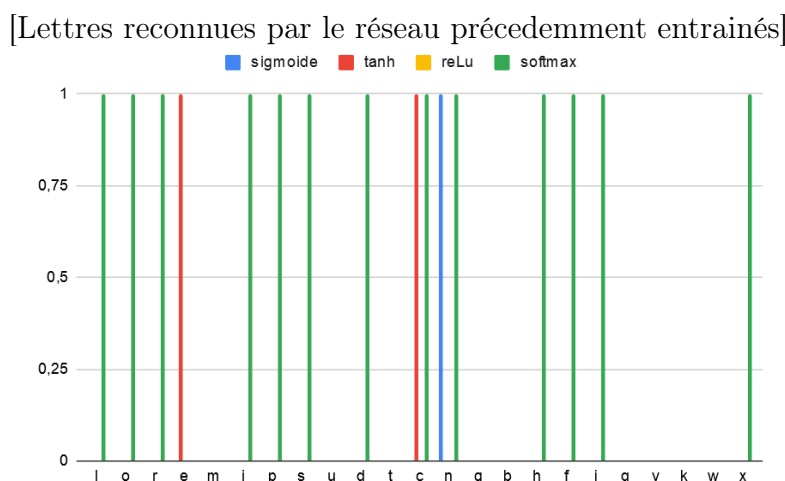
Softmax :

$$h(x) = \frac{e^x}{\text{sum}(e^z)}$$

Softmax est utilisée dans la classification de problème multi-classes. C'est-à-dire que Softmax attribue des probabilités décimales à chaque classe d'un problème à plusieurs classes. La somme de ces probabilités décimales doit être égale à 1. Cette contrainte supplémentaire permet de faire converger l'apprentissage plus rapidement qu'il ne le ferait autrement. L'apprentissage est encore accéléré si on utilise la cross-entropy pour calculer l'erreur du Layer softmax. Dans ces tests, elle n'est pas utilisé. Deplus, Softmax est une fonction qui traite des données sous forme de vecteur ou de matrice colonne. Cela est donc très avantageux dans les structures de réseau neuronaux qui utilisent des opérations sur les matrices.



Grâce à ces test, nous pouvons constater que la fonction softmax est de loin celle qui converge le mieux dans le cas présent. Le réseau converge de façon très satisfaisant au bout de seulement 500 epochs.



Enfin, on compare l'efficacité de chacun des réseaux entraînés ci dessus pour reconnaître les 24 lettres de la même phrases. La fonction d'activation softmax est la plus efficace car elle reconnaît, avec seulement 500 epochs d'entraînement, 13 lettres.

Après comparaison des fonctions d'activation les plus utilisées dans les réseaux de neurones artificiels, nous avons choisi d'utiliser la fonction softmax. Ainsi, notre réseau apporte des résultats satisfaisant avec peu d'entraînement.

5.3.3 Propagation par Lot

Pour entraîner notre réseau le plus efficacement possible, nous avons choisit d'implémenter une propagation arrière dite par lot (ou batch propagation). Cette évolution a été réalisé par Jeanne. Le principe est simple : pour un nombre n de propagation arrière, on somme les n matrices erreurs d'une layer donné et on soustraira cette somme aux matrices de poids et de biais du layer après les n propagations. Les matrices poids et biais sont ainsi mises à jour qu'à la fin. Leur calcul est d'autre part grandement simplifiés. Cette méthode permet d'amoinrir le coût de la propagation arrière lors de l'entraînement du réseau et de rendre l'apprentissage plus rapide.

Dans un premier temps, Jeanne a modifié la fonction la fonction back propagation, déjà implémentée, pour empêcher la mise à jour des matrices poids et biais de chaque layer à chaque appel de la fonction. Elle a également modifié notre structure Layer pour ajouter des matrices `weights_batch` et `biases_batch`, qui contiendront la somme des matrices erreurs du Layer. Enfin, elle a implémenté une fonction Final Update qui permet de mettre à jour les matrices biais et poids de chaque couche. Cette fonction est appelée à chaque fin d'entraînement du réseau avec un lot d'exemple complet. Elle permet de grandement amoindrir le coût des calculs de la back propagation durement l'entraînement et ainsi réduire la durée de ce dernier.

5.3.4 Entraînement

L'objectif de cette partie est de créer et maintenir un ensemble d'exemples qui seront utilisés pour entraîner le réseau de neurones. Le but est de pouvoir facilement lancer l'entraînement du réseau sur une grande base d'exemples afin qu'il puisse finalement reconnaître un document complet.

Il faut aussi prendre en compte que l'entraînement ne pouvait se faire d'un seul coup avant l'échéance finale. Il était au coeur du processus de test des du fonctionnement du réseau de neurones. De plus, il a fallu que nos algorithmes soient facilement modifiables pour intégrer de plus en plus de type de caractères pour que le réseau puisse les reconnaître.

Ainsi, la gestion de cette base d'exemples est réalisée grâce à 3 fichiers situés dans le dossier `neuralNetwork_data/` :

- `size.data` : contient sur sa première ligne le nombre d'exemples qui seront contenus dans la base.
- `names.data` : contient le chemin et le texte d'une image ;
- `examples.data` : contient les matrices et les caractères représentés sur ces matrices.

L'entraînement : Lors du lancement de l'entraînement, si le fichier `examples.data` n'existe pas, alors la lecture et la création des exemples sera mise en place pour lire le fichier `names.data` afin de créer la base. Enfin, le fichier `examples.data` est lu afin de créer le lot principal d'exemple.

La partie principale de l'entraînement peut enfin commencer après cette étape de création du lot d'exemple. On considère que le réseau est entraîné lorsque le taux de succès est supérieur à 0.9 ou 90%. On fera donc l'entraînement tant que ce taux n'est pas atteint. De manière à sécuriser le tout, le réseau est sauvegardé tous les 100 lots passés, pour prévenir une session d'entraînement qui n'aurait pas pu aboutir, à cause d'un problème de batterie par exemple.

Afin de réaliser un entraînement efficace, nous avons choisi de le réaliser par lot. Ainsi, le grand lot d'exemple va être divisé en plus petits lots qui passeront ensuite dans les fonctions de propagation par lot. Il est important de noter que cet entraînement par lot n'est efficace si et seulement si la base d'exemple est suffisamment importante pour qu'un petit échantillon puisse être considéré représentatif du lot principal. De manière à ce que cette uniformité soit respectée, le lot d'exemple principal est mélangé entre chaque itération de l'entraînement.

L'entraînement par lots d'exemples diffère assez peu de l'entraînement classique, dit *online*. Là où on faisait les étapes toujours les unes à la suite des autres, on décidera dans la propagation par lot de réaliser certaines étapes en boucle en stockant le résultat, puis d'effectuer l'étape coûteuse d'un seul bloc sur tous ces résultats. En détail, l'entraînement se réalise comme suit :

- Création de minis-lots à partir du lot principal d'exemples mélangé.
- La propagation avant pour tous les exemples d'un mini-lot : On propage les signaux d'entrées jusqu'à la couche finale.

- La propagation arrière pour tous les exemples d'un mini-lot : L'erreur est propagée de l'arrière du réseau vers l'avant en prenant soin de stocker toutes les valeurs d'erreurs.
- La mise à jour des poids et biais lorsque tous les exemples d'un mini-lot sont passés.
- Calcul de la précision de l'entraînement. Si il ne respecte pas la condition fixée, alors recommencer.

Il est à noter que dans un soucis d'optimisation de la mémoire, les exemples ne sont pas lu à nouveau à chaque fois qu'il faut recommencer le processus d'entraînement. Ils sont gardés en mémoire et les seules opérations sont donc des libérations et créations de pointeurs. De plus, l'écriture et la lecture de ces fichiers est réalisé avec les fonctions `fread` et `fwrite` de la bibliothèque `stdio` dans le même soucis d'optimisation.

Comme il a été soulevé pendant la première soutenance, l'entraînement du réseau de neurone peut prendre beaucoup de temps, aussi bien au niveau du temps qu'il nous faut pour laisser le réseau s'entraîner puis constater ou non son efficacité, qu'au niveau de la quantité de paramètres à changer.

Il a été assez compliqué de réaliser l'entraînement de notre réseau. Malgré les algorithmes que nous avons mis en place pour générer la base d'exemple, il a fallu trouver et créer des fichiers étant parfaitement reconnus par la segmentation, d'une part, et ne pas se montrer trop insatiables par rapport à la quantité d'exemple dont nous allons avoir besoin. En effet, nous avons commencé par un entraînement sur 12 alphabets dans des polices différentes, et nous nous sommes rendus compte que ce n'était pas efficace, et ce malgré des changement sur le nombre de neurones dans la couche cachée et le taux d'apprentissage.

Il a d'ailleurs paru évident, à cette phase qui était proche de la fin du projet, de rendre l'entraînement plus facile à réaliser. Nous avons à cette étape le besoin de changer à chaque itération de notre séance d'entraînement le nombre de neurones de la couche cachée et le taux d'apprentissage. Or, cela nous faisait perdre beaucoup de temps et n'était pas pratique. C'est pour cela que nous avons développé une mini-CLI pour rendre l'entraînement plus flexible. Lorsque le programme est lancé avec un argument (autre que le nom du programme), il sera demandé le nombre de neurones de la couche cachée et le taux d'apprentissage.

De cette manière, nous pouvons présenter cette version du réseau de neurones capable de reconnaître une majorité de caractères.

5.3.5 Au sein de l'OCR

Le réseau de neurone est utilisé dans l'OCR pour reconnaître les caractères lus sur une image. Pour cela, le réseau a besoin d'être entraîné à la reconnaissance de caractères. Après cette étape, il pourra être correctement utilisé dans le programme complet.

Ainsi, lorsque la segmentation a récupéré l'ensemble des caractères présents sur le fichier image, elle les envoie sous forme de matrice de zéros et de uns au réseau. Chaque matrice va alors être propagée dans le réseau, qui va estimer de quel caractère il s'agit. Il renverra un entier i entre 0 et 26 représentant la i -ème lettre de l'alphabet. Le réseau est actuellement entraîné pour reconnaître l'alphabet minuscule. Il serait néanmoins possible de l'entraîner pour reconnaître plus de caractères, comme la ponctuation ou les majuscules ou encore les chiffres. Mais pour sécuriser le rendu d'un projet fonctionnel nous avons décidé de nous concentrer sur les minuscules, l'entraînement étant déjà suffisamment chronophage. Cette estimation va alors être renvoyé à la reconstruction de texte pour placer le caractère.

6 Reconstruction du texte

La dernière étape est la reconstruction du texte. Elle a été faite par William. Une fois la segmentation fini et le réseau suffisamment fonctionnel. Il faut arriver à renvoyer un fichier avec le texte que nous avons reconnu dans l'image. La manière dont nous avons stocké nos paragraphes, lignes, mots et caractère va nous permettre de reconstruire facilement le document. En effet pour savoir quand est-ce qu'il faut ajouter un espace ou un retour à la ligne, nous avons besoin de savoir quand est-ce qu'un nouveau mot est construit pour y ajouter un espace à la fin. La même chose doit être faite pour la fin de chaque ligne. Pour sauvegarder cette information nous avons une liste de paragraphes qui contient à la fois la position du paragraphe dans le document ainsi que la liste des lignes contenues dans ce paragraphe. Ce même système est implémenté pour les listes de ligne, qui contiennent une liste de mots et enfin la liste de mots qui contient une liste de caractère.

Nous pouvons donc appliquer un traitement récursif sur nos listes afin de reconstruire le document en insérant des espaces et des retours à la ligne aux bons endroits. Il faut cependant faire attention à ne pas ajouter un espace à la fin du dernier mot, il faut donc gérer le dernier mot différemment des autres.

Une amélioration qui aurait pu être intéressante aurait été d'ajouter de la mise en page. En effet puisque nous sauvegardons la position des paragraphes et des images, nous arrivons à détecter les images. Il aurait été intéressant de remettre cela en forme dans un fichier PDF par exemple. Cependant les outils disponibles à l'école pour la construction du PDF avec mise en page sont assez réduits. Des outils comme HanaPDF ou PDFCreator ne sont pas sur les ordinateurs de l'école ce qui réduit nos options. LaTeX était une des solutions possibles, mais il n'est pas possible de placer exactement comme on souhaite les différents paragraphes. En effet LaTeX s'occupe de faire ça pour nous, ce qui rend la mise en page beaucoup plus compliquée. Bien que certains paquets nous permettent d'organiser au pixel près la disposition des paragraphes. Cependant ils ne sont pas disponibles sur les ordinateurs de l'école, cette solution n'était donc pas non plus envisageable. Ils auraient fallu se pencher du côté du langage HTML ou du XML. Cependant, n'ayant aucune notion dans ses 2 langages il était compliqué de les utiliser pour reconstruire proprement notre document. Cependant les informations nécessaires pour reconstruire notre document avec la mise en page sont déjà disponibles. Mais dû à notre manque de connaissances en XML et en HTML nous avons préféré créer un fichier texte avec les paragraphes séparés par des retours à la ligne, sans mise en page ni intégration des images détectées par le réseau. L'exemple ci-dessous est effectué avec un réseau non entraîné, les lettres n'ont donc pas de sens, mais on reconnaît bien les différentes longueurs de mots, lignes et paragraphes que compose notre texte.

On reconnaît en effet les quatre paragraphes qui composent notre texte, le deuxième possèdent des lignes plus courtes car il était également plus court que les deux autres, les différents mots sont espacés et chaque ligne est séparée d'un retour à la ligne. les paragraphes eux, sont séparés de deux retours à la ligne ce qui permet de les différencier beaucoup plus aisément.

```

hhhhhhh hhhhhhhhhhhh hhhhhhhhhhhhhh hhhh hhhhhh hhhh hhhhhhhhhhhhhh
hhhhhh hh hh hh
hh hhh hh hhhhhhhh hhhh hhh hh hhhhhhhh hh hh hh hhh hh hh hhhhhh hh
  h hhhhhhhh hh
hhhh hh hh hh hh hh hh h hhh hhh hhhhhh hhhh hhhh hhh hh h hhhhhhhh hh
hhh hhhhhh
hhhhhhhhh hhh h hhh hhhh h hhhhhhhh hhhhhhhhhh hhh hhhhhh hhhh hhhh
h hh hhhh hhhh
hhh hh hhhh hhhh hhh hh h hh hhhh hh hhh h hhhh hhhh hh hhh hhhh
h hhh hhhhhhhh
hhh h hhh hhhh

```

```

hhhhhhh hhhhhh hh h hhh hhh hhh hhh
hh hh hhhhhhhhhh hhhhhhhh
hh hhh h hhhh hhhhhhhhhh hhhh
hhhhhhhhhhhhh hhh hhhh hhhh
hhh h h hhhh hhhh hhhh hhh hhhh
hhhhhhh h hhhh hhhh hhhh hhh hhhh
hhh hhh hhhhhhhhhhhhhh h

```

```

hhhhhhh hhhhhhhhhhhh hhhhhhhhhh hhhh hhhh hhhhhh hhhh hhhhhhhhhhhhhh
hh
hhhhh hh hh hh hhhhhh hhhhhhhh hhhhhh hh hhhhhh hh hh hh hhh hh hh
  hh
hhhhh hh h hhhhhhhhhhhh hh hh hhhh hh h hhh hhh hhhh hhhhhhhh
hhhhh h hhhh hhhhhhhhhhhhhh hhhhhhhh hhh h hhh hhhh h hhhhhhhh hhh
hh
hhhh hhh hhhh hhhh hhhh h h h h hhhh hhhh

```

```

hhhhhhh hhhhhhhhhh hhhhhhhhhh hhhh hhhh hhhh hhhh hhhhhhhhhhhhhh
hh
hhhhh hh hh hh hhhhhh hhhhhhhh hhhhhh hh hhhhhh hh hh hh hhh hh hh
  hh
hhhhh hh h hhhhhhhhhhhh hh hh hhhh hh h hhh hhh hhhh hhhhhhhh
hhhhh h hhhh hhhhhhhhhhhhhh hhhhhhhh hhh h hhh hhhh h hhhhhhhh hhh
hh
hhhh hhh hhhh hhhh hhhh h h h h hhhh hhhh

```

(w) Exemple de texte en sortie

7 Interface Graphique

L'interface graphique de notre projet a été développée par Alexandre en utilisant la librairie GTK+ (version 3.0) et son design a été réalisé avec Jeanne. Elle permet à l'utilisateur à la fois d'appliquer notre OCR sur une image, mais aussi d'éditer un fichier texte.

7.1 Développement

L'interface graphique étant le seul élément visuel sur lequel l'utilisateur peut se repérer pour utiliser notre OCR, il fallait qu'elle soit simple d'utilisation tout en étant assez complète. Ainsi, après avoir fait plusieurs recherches sur les différents outils disponibles avec la librairie GTK+, Alexandre a décidé de baser l'interface graphique autour d'un éditeur de texte.

En effet, l'interface graphique possède deux fenêtres principales :

- **Un menu principal :**

Au lancement du programme, l'utilisateur peut choisir de lancer l'OCR sur une image choisie, ou de simplement éditer un texte et ainsi accéder à notre seconde fenêtre. Il peut aussi tout simplement quitter l'application avec le bouton correspondant.

- **Un éditeur de texte :**

Que l'utilisateur ait choisi de lancer l'OCR ou d'éditer un texte, cette fenêtre s'ouvre et offre à l'utilisateur de nombreuses options, dont principalement, comme son nom l'indique, d'éditer un fichier texte.

Pour décrire les différentes options que propose cette seconde fenêtre de l'interface graphique, on peut la séparer en 3 parties :

- **Un conteneur pour le texte :**

La plus grande surface est occupée par le widget `GtkTextView`, qui correspond à l'espace où le texte va pouvoir être affiché et édité. Si le texte ne rentre pas dans l'espace initial, des barres de défilement horizontale et/ou verticale apparaissent pour pouvoir accéder aux parties cachées.

- **Une barre d'outils :**

Le widget `GtkToolbar` inclut dans la librairie GTK+ a permis de regrouper les différents boutons que l'utilisateur va pouvoir utiliser pour accéder aux options que cette fenêtre propose. Chaque bouton a une icône propre représentant l'option qu'il va activer. Voici les différents boutons que la barre d'outils possède :

- OCR : Ce bouton est le même que celui sur le menu principal, et permet de choisir une image, puis de la confirmer avant d'appliquer l'algorithme de l'OCR. Le texte provenant de l'algorithme de reconstruction du texte sera ainsi affiché dans le widget `GtkTextView` présenté précédemment afin de pouvoir vérifier le résultat et/ou le modifier.
- Nouveau fichier : Si l'utilisateur souhaite créer un nouveau fichier, ce bouton permet de fermer le fichier ouvert (s'il existe) et de vider le `GtkTextView` de son contenu afin d'obtenir un nouveau fichier vide.

- Ouvrir un fichier : Dans le cas inverse où l'utilisateur veut ouvrir un fichier contenant du texte, une fenêtre s'ouvre afin de le sélectionner et d'afficher son contenu dans le `GtkTextView`, permettant de l'éditer.
 - Sauvegarder : Bien évidemment, l'utilisateur souhaite pouvoir sauvegarder les modifications qu'il a apporté au texte affiché, pour cela, un bouton "Sauvegarder" est disponible pour sauvegarder le fichier. Si le fichier a déjà été enregistré à un emplacement, le fichier est sauvegardé à celui-ci. Sinon, une fenêtre s'ouvre permettant à l'utilisateur de choisir le nom et l'emplacement du fichier texte qu'il veut enregistrer.
 - Sauvegarder sous : Même principe que le bouton "Sauvegarder", mais demande directement le nom et l'emplacement du fichier texte même si il en possède déjà un.
 - Fermer : Permet de fermer le fichier texte sans fermer l'application. Si l'utilisateur a apporté des modifications au fichier texte mais ne l'a pas sauvegardé, propose à l'utilisateur de sauvegarder les modifications.
 - Quitter : Comme son nom l'indique, ferme l'application et le fichier texte ouvert, proposant à l'utilisateur de sauvegarder les modifications qui n'ont pas été enregistré.
- **Une barre menu** : Grâce au widget `GtkMenu`, l'utilisateur peut retrouver tous les boutons disponibles dans la barre d'outils dans un premier onglet sous forme d'une liste portant le nom de chaque bouton à la place de leur icône. Cependant, un second onglet permet d'accéder à de nouvelles fonctionnalités :
- Readme : Affiche un fichier README afin de retrouver l'explication des différentes fonctionnalités de l'interface graphique.
 - Doc : Lance le programme pour accéder à la documentation Doxygen complète du projet sur internet.
 - A propos de : Ouvre une fenêtre "A propos" indiquant le nom, la version, les auteurs du projet ainsi que le lien du site web d'EPITA, à qui le projet appartient.

Pour stocker les informations concernant le document ouvert ainsi que l'interface en elle-même, deux structures ont été ajoutées :

- Une structure Document : Cette structure permet de stocker les informations relatives au fichier texte ouvert, soit l'emplacement où il est enregistré (s'il existe), le status actuel de sa sauvegarde, c'est à dire "est-ce que le fichier a été sauvegardé depuis sa dernière modification ?", ainsi que le widget `GtkTextView` dans lequel il se trouve, utile pour les différentes options de l'éditeur de texte.
- Une structure UI : Afin d'accéder à des éléments comme le document ouvert, la fenêtre du menu principal ainsi que la fenêtre de l'éditeur de texte, cette structure nous est très utile dans l'exécution du programme.

Alexandre n'a pas utilisé de logiciel comme Glade pour réaliser son interface, ce qui aurait pu faciliter son développement. Il souhaitait découvrir complètement les différentes syntaxes des widgets disponibles sur la librairie GTK+ afin de les utiliser dans de futures projets.

7.2 Design

Le visuel de l'interface est autant important que l'interface doit être complète. C'est pourquoi le design a été réfléchi et réalisé par Alexandre et Jeanne, afin de rendre le plus agréable possible pour l'utilisation de notre OCR.

Le design du menu principal et de ses différents fonds disponibles ont été imaginés et réalisés par Jeanne. Les fonds, au nombre de 3, ont été créés sur Canva, un éditeur de graphisme pour les particuliers. Ils apparaissent de façon aléatoire comme fond du menu principal, ce qui crée du dynamisme entre chaque utilisation de notre OCR, donnant un effet de renouvellement.



(x) Menu principal

Le design de l'interface de l'éditeur de texte est simple car la zone de texte prend le plus d'espace, mais est aussi colorée grâce aux icônes de la barre d'outils donnant de la modernité à l'interface.



(y) Barre d'outils

8 Architecture du projet

Cette partie résume les changements et améliorations qui ne touchent pas directement les fichiers sources. Nous allons expliquer ce qui a été mis en place pour simplifier l'avancement ainsi que la compréhension du projet.

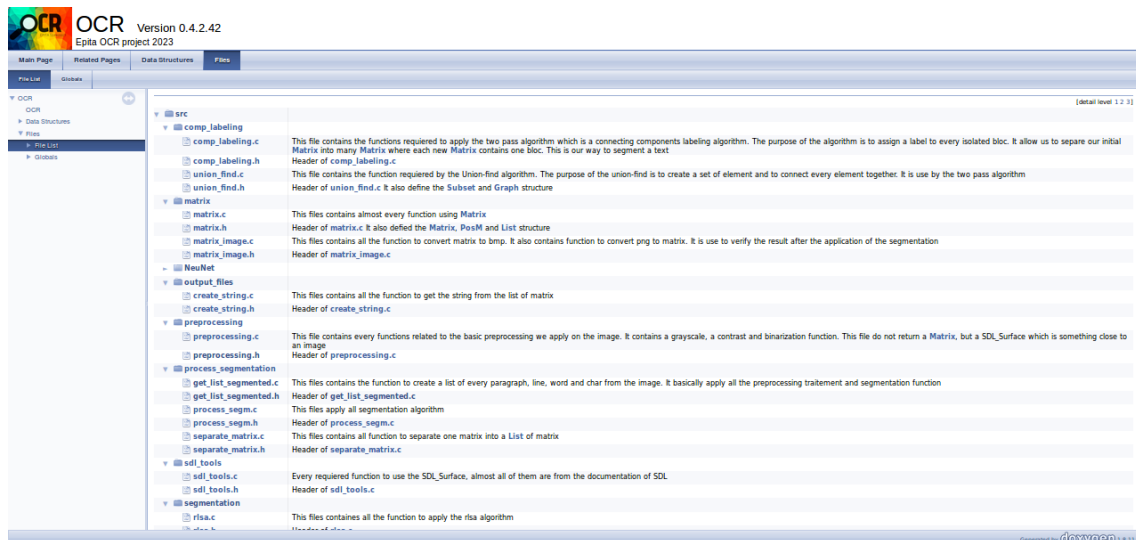
Dans un premier temps, un des problèmes du makefile pendant le développement du projet était l'ajout des nouveaux fichiers sources dans la compilation. Il fallait les rajouter à la main ce qui avait pour conséquence une perte de temps importante et un makefile très peu automatisé et assez laids. Dans un premier temps William souhaitait ajouter un makefile récursif, cela avait pour avantage de pouvoir compiler indépendamment les différentes parties de notre projet. Mais cette possibilité avait 2 défauts, la première c'est qu'il fallait refaire l'architecture pour créer un programme main dans chacun de nos partis. La deuxième qui est également la plus problématique, est l'ordre dans lequel les fichiers sont compilés. Le désavantage du makefile récursif c'est qu'il est délicat d'ordonner quelle partie doit être compilée avant une autre. Cela n'est pas problématique si nos différentes parties sont complètement indépendantes. Cependant si une partie A a besoin d'un fichier de la partie B, mais que la partie B n'est pas déjà compilée, alors la partie A ne pourra pas compiler. C'est la raison pour laquelle William a abandonné le makefile récursif pour envisager une autre solution.

Ainsi, il a utilisé une commande shell pour récupérer tous les fichiers sources de notre projet, et nous n'avons plus besoin de modifier le makefile à chaque ajout d'un nouveau fichier source. Notre makefile commençait à comporter beaucoup de règles, il était donc primordial d'expliquer leur utilité. C'est pour cela que William a ajouté un système permettant d'expliquer rapidement ce que fait chacune des règles directement dans le makefile. Une commande shell va récupérer cette explication et l'afficher ensuite dans le terminal. Vous pouvez voir le résultat plus bas :

```
turlesmoke@turlesmoke-VirtualBox:~/Desktop/OCR$ make
clean          Clean every .o and .d.
doc            Generate html documentation and open it in your default browser.
doxygen        Generate html documentation and shortcut doc.html.
mrproper       Clean every .o and .d as well as all generated files.
OCR            Generate the executable, use ./OCR to launch the project.
```

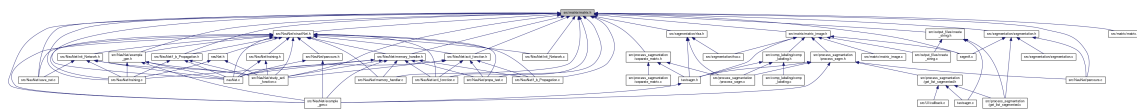
(z) Aide du makefile

Comme vous pouvez le constater plus haut, la règle doxygen permet de créer une documentation html. En effet, dans un second temps, puisque tout le monde utilisait les fonctions des uns et des autres. Il était important d'avoir un moyen de comprendre ce que chacun a fait. C'est pour cela qu'il était primordial de commenter nos fonctions. Cependant William ne trouvait pas cette solution suffisamment bonne et c'est la raison pour laquelle il a implémenté une documentation générée automatiquement avec l'aide de Doxygen. Cet outil nous permet de créer une documentation HTML assez aisément, il suffit de placer des commentaires avec les bonnes balises dans notre code et Doxygen s'occupe de faire le reste. Il faut cependant configurer Doxygen pour avoir un bon rendu, ce qui ne s'est pas forcément avéré facile. Voici un exemple de ce que peut faire Doxygen :



() Doxygen

On peut voir sur cette image les dossiers ainsi que les fichiers qui sont documentés. Cela nous permet de naviguer facilement et de comprendre ce que chacun a fait. Il est possible de regarder directement chaque fichier, chaque fonction et chaque variable. La documentation offre aussi la possibilité d'étudier les différentes structures implémentées dans le projet. Un autre exemple de ce que peut faire doxygen :



() Graph d'inclusion

Bien que l'image soit très petite, on peut voir au-dessus les différents fichiers qui utilisent celui le plus haut. Cela nous permet de voir quels fichiers utilisent, en l'occurrence, la structure de Matrix. Cela permet de savoir quels fichiers vont possiblement changer si on modifie certaines fonctions de la structure Matrix.

9 Bilan personnel

9.1 Pierre-Olivier

CE projet s'est finalement révélé beaucoup plus difficile que prévu. Premièrement par la complexité technique, nous sommes tombés dans de nombreux problèmes bloquants tout au long du projet et il a parfois été vraiment compliqué de comprendre et debugger nos programmes.

Le travail de groupe n'as aussi pas été aussi simple que ce à quoi je m'attendais. La communication a été un réel problème et nous a fait perdre beaucoup de temps et de lucidité par rapport au projet. J'en retire néanmoins de plus grandes connaissances en C particulièrement au nouveau des structures et de la gestion de la mémoire, et sûrement de plus grandes habilités au travail en équipe.

9.2 Jeanne Morin

Pour ma part, j'ai trouvé que notre groupe à eu beaucoup de difficultés à travailler ensemble. Nous avons réalisé la quasi totalité de nos tâches individuellement, même les tâches réparties à un groupe de personnes.

J'ai eu beaucoup de mal à trouver ma place dans le groupe car je ne partageais pas la même façon de travailler que mes trois coéquipiers. La communication a été très complexe, pendant les trois quarts du projet. J'ai été laissé de côté pendant cette période. Tous ces éléments, ajouté à ma note de soutenance très inférieur à celles de mon groupe, mon grandement démoralisé. Je n'avais jamais vécu cela avant mais j'ai parfois eu l'impression de gêner mes coéquipiers et d'être sous estimé dans mes capacités de travail. Je n'avais également jamais connu des difficultés à m'imposer et à m'impliquer dans un groupe. J'ai donc rencontré des situations inédites qui me seront bénéfique pour l'avenir.

D'un point de vue technique, j'ai beaucoup appris sur le langage C et mon niveau en programmation a progressé. De plus, les sujets des réseaux de neurones artificiels, dont je n'avais jamais entendu parlé avant, m'ont passionné. J'ai ainsi découvert une dimension de l'informatique que je n'avais jamais envisagé.

Pour finir sur une note positive, nous avons la chance d'être un groupe travailleur. Tous les membres sont très impliqués dans le projet, ce qui est un vrai atout. De plus, nous avons avancé de manière régulière et assidus dans notre travail. Je suis persuadé que nous présentons un travail de qualité.

9.3 Alexandre Lemonnier

Ce projet fût pour moi une très bonne expérience à la fois techniquement que humainement. L'apprentissage des différentes notions du langage C et leurs applications dans le projet m'ont permis de me familiariser avec ce langage qui était nouveau pour moi. Ce langage, étant complexe à la fois par la gestion de la mémoire et l'implémentation des structures, me sera utile dans mes futures projets. La réalisation de la segmentation de l'image était un problème que je n'avais pas encore rencontré, le rendant très enrichissant et intéressant. Afin de développer l'interface graphique, je me suis orienté vers la librairie GTK+. Pour créer une interface la plus complète possible, j'ai dû me renseigner sur les différentes options disponibles grâce à cette librairie. Au cours de son développement, j'ai rencontré plusieurs difficultés que j'ai apprises à gérer au fil du temps. Le développement d'une interface graphique demande d'imaginer les différentes envies de l'utilisateur lors de son utilisation, et d'ainsi corriger les différents bugs qu'il pourrait rencontrer si l'interface ne gère pas ce cas là.

Au niveau du groupe, je trouve que le manque de communication entre les membres du groupe a posé de nombreux problèmes alors que la communication est un élément élémentaire de tout projet de groupe, et aurait pu être gérée facilement si chacun avait exprimé clairement ses idées. Pour rétablir cette communication, j'ai essayé d'apporter au groupe de la motivation ainsi que de la bonne humeur pour continuer le projet dans de bonnes conditions. Cette expérience sociale est pour moi une situation qui pourra sûrement se réaliser de nouveau lors de mes futurs projets.

9.4 William Grolleau

De mon côté, ce projet m'a apporté une connaissance importante dans le langage C, ce langage est très différent de ceux que nous avons étudiés jusqu'ici, puisqu'il demande une réflexion particulière lors de l'implémentation de certains algorithmes ou de certaines structures de données. D'un côté le traitement de l'image m'a permis de comprendre les enjeux actuels car il est en effet très délicat d'effectuer des opérations sur des images. Il existe beaucoup de cas compliquant le traitement de celles-ci. De l'autre la segmentation de l'image m'a fait découvrir de nombreux algorithmes très intéressants, tel que l'algorithme de two pass qui me sera très certainement utile dans certains de mes futurs projets. Pour ce qui n'est pas relié au langage C, j'ai appris à utiliser certains outils pour faciliter la mise en place et l'architecture d'un projet. Bien qu'il me manque encore beaucoup de choses à apprendre, comme par exemple l'utilisation de test unitaire pour assurer un fonctionnement général à chaque ajout de fonction. Je suis satisfait des outils que j'ai mis en place dans l'OCR et je pense les réutiliser dans des projets futurs. Ce projet m'a également permis d'apprendre à organiser le travail du groupe et à gérer une équipe. J'ai pu apprendre de mes erreurs et ainsi m'améliorer. J'espère que dans mes futurs projets je pourrais appliquer ce que je n'ai pas pu ici pour éviter que les problèmes que nous avons rencontrés se reproduisent dans de futurs projets. Dans tous les cas, ce projet restait très intéressant aussi bien humainement que techniquement. Il est vrai que certains points étaient très frustrants et pouvaient même me dégoûter. Si je devais résumer mon avis quant au projet en général, ce serait certainement :

Ce projet, bien que très intéressant n'était pas toujours agréable. Cependant il reste une expérience nécessaire d'avoir au moins une fois dans sa vie afin de ne plus jamais devoir y être confrontée à l'avenir.

10 Conclusion

Pour conclure, nous avons tous les quatre été très impliqués dans ce projet passionnant et avons avancé à bon rythme. La première partie de projet nous a permis en généralité d'apprendre le langage C et ses différentes difficultés. Par conséquent, nous avons pu être beaucoup plus efficace, que ce soit dans la répartition des tâches, ou dans leur exécution, pour la seconde partie du projet. Nous sommes parvenu à avoir un OCR fonctionnel pour la soutenance finale et nous en sommes très fiers.