

Projet OCR

Soutenance Finale

Abadie Hugo
Jomier Magali
Machavoine Ethan
Poelger Jonathan



Décembre 2020
Encadrant : Rémi Vernay

Table des matières

1	Introduction	3
2	Notre logiciel de reconnaissance de caractères	4
2.1	Utilisation	4
2.2	Contraintes	4
3	Découpage du projet	5
4	Fonctionnement	6
4.1	Chargement des images et suppression des couleurs	6
4.1.1	Chargement des images	6
4.1.2	Suppression des couleurs	6
4.2	Pré-traitement	8
4.2.1	Rotation de l'image	8
4.2.2	Réduction des bruits	9
4.2.3	Binarisation	11
4.3	Détection et découpage en lignes et caractères	15
4.3.1	Découpage en lignes	15
4.3.2	Découpage en caractères	16
4.4	Extraction et mise en forme des caractères	18
4.4.1	Extraction de chaque caractère	18
4.4.2	Uniformisation du caractère	18
4.4.3	Appel au réseau de neurones	20
4.4.4	Mise en page	20
4.5	Réseau de neurones	22
4.5.1	Définition et fonctionnement	22
4.5.2	Création du réseau	25
4.5.3	Exécution du réseau	26
4.5.4	Entraînement du réseau	28
4.5.5	Création de la collection d'entraînement	31
4.5.6	Sauvegarde et chargement des poids et des biais du réseau . .	32
4.6	Interface graphique	33
5	Améliorations possibles	36
6	Impressions personnelles	37
6.1	Hugo Abadie	37
6.2	Magali Jomier	38
6.3	Ethan Machavoine	39
6.4	Jonathan Poelger	40
7	Conclusion	41

1 Introduction

Cette deuxième année de cycle préparatoire à l'EPITA a démarré en force avec l'arrivée du second projet à réaliser dans notre cursus d'ingénieur en informatique. Ce projet, un logiciel de reconnaissance optique de caractères (OCR), a été développé par une équipe qui a tout donné pour réussir.

Cette équipe est composée de 4 étudiants motivés : Abadie Hugo, Jomier Magali, Machavoine Ethan et Poelger Jonathan.

Pour arriver à la réalisation d'un OCR fiable et fonctionnel, plusieurs parties doivent être traitées. Tout d'abord, un chargement et un traitement de l'image doit être réalisé, transformant une image quelconque en une image en noir et blanc. Suite à cela, il faut reconnaître et découper les zones de textes présentes dans cette image. On découpe de même chaque caractère de chaque zone de texte. Ces caractères sont ensuite envoyés à un réseau de neurones qui s'occupe de les reconnaître. Enfin, il nous faut reconstituer le texte avec les caractères reconnus afin de le sauvegarder et de pouvoir l'utiliser. Une interface graphique vient se greffer autour de tous ces composants pour apporter un meilleur confort d'utilisation à l'utilisateur.

Nous présentons aujourd'hui, après 3 mois de travail, un rapport final de ce projet.

Ce rapport se découpe en plusieurs parties avec tout d'abord une présentation de notre application et notamment de son utilisation.

La deuxième partie se concentrera sur la répartition des tâches à réaliser pour ce projet entre les différents membres du groupe.

La troisième partie de ce rapport se focalisera sur le fonctionnement de notre OCR, en commençant par les différents traitements de l'image puis en abordant le réseau de neurones et l'interface graphique.

Enfin, ce rapport se terminera sur nos impressions personnelles à propos de ce projet.

2 Notre logiciel de reconnaissance de caractères

2.1 Utilisation

L'application se lance depuis la console avec un simple exécutable (que l'on peut construire au besoin avec *make* puis lancer avec *./main*). Plusieurs options s'offrent alors à l'utilisateur, la plus probable étant sans doute de charger une image. Pour cela, un menu permet de chercher dans les fichiers de son ordinateur et trouver l'image voulue, qui s'affiche une fois sélectionnée.

Ensuite, il est possible de tourner l'image avec plusieurs boutons prévus à cet effet, ou de lancer la détection des caractères en appuyant sur *Start*. Suite à cela, le texte trouvé par l'IA s'affiche à droite de l'image, et il est alors possible de le modifier et de sauvegarder le résultat dans un fichier texte.

Enfin, il est possible de lancer un entraînement du réseau, dont l'avancée sera affichée dans la console. A la fin de l'entraînement, un message *pop-up* annoncera la fermeture de l'application.

2.2 Contraintes

L'utilisateur doit respecter plusieurs contraintes pour utiliser notre logiciel de manière optimale :

- l'image ne doit pas être dégradée outre mesure sinon le texte sera difficilement détectable
- l'image ne doit pas être trop inclinée car la rotation dégrade un peu l'image à chaque fois
- le texte doit être écrit dans une police reconnue par le logiciel. Les polices prises en charge sont Arial et Times new roman

3 Découpage du projet

Ce tableau montre la répartition des différentes tâches entre les membres du groupe.

Répartition	Hugo	Magali	Ethan	Jonathan
Pré-traitement		X		
Découpage des caractères	X	X		
Mise en forme des caractères	X	X		
Réseau de neurones			X	X
Jeu d'images d'entraînement	X			X
Manipulation de fichiers			X	X
Interface utilisateur		X		

4 Fonctionnement

4.1 Chargement des images et suppression des couleurs

4.1.1 Chargement des images

- La bibliothèque *SDL2*

Afin de pouvoir travailler sur des images, nous avons choisi d'utiliser la librairie *SDL2*. Cette librairie contient les mêmes fonctionnalités que *SDL* mais offre la possibilité de travailler sur de nombreux formats d'images comme *JPEG* (.jpg) ou *PNG* (.png) et non uniquement sur des fichiers *Bitmap* (.bmp).

Avant toute utilisation de cette librairie, il est nécessaire de l'initialiser grâce à la commande *SDL_Init*. De même, à la fin des manipulations la bibliothèque doit être fermée avec *SDL_Quit*. Nous avons ainsi accès à la fonction *IMG_Load* qui permet de charger une image.

Cette librairie offre aussi la possibilité d'accéder et de modifier plusieurs paramètres de l'image (sa largeur et sa hauteur) ainsi que sur les pixels qui la composent (leur couleur en format *RGB* ou *RGBA*). Ces fonctions seront très utiles pour le reste du projet.



4.1.2 Suppression des couleurs

- *Greyscale*

Avant de commencer à traiter l'image, les couleurs doivent être supprimées pour permettre une meilleure analyse des pixels de celle-ci et sa binarisation.

Pour cela nous avons implémenté la fonction *greyscale*. Elle transforme tous les pixels de l'image en nuances de gris. Pour chaque pixel, nous calculons son niveau de gris grâce à sa valeur en format *RGB* dans lequel *r* correspond au rouge, *g* au vert et *b* au bleu. Ainsi le niveau de gris correspond à :

$$\text{greyscale} = 0.3 * r + 0.59 * g + 0.11 * b$$

Image de pingouin d'origine et image après suppression des couleurs



Les modifications suivantes sont opérées sur les images en nuances de gris donc avec des valeurs r , g et b identiques.

4.2 Pré-traitement

Lorsque l'utilisateur va donner une image au logiciel, elle peut ne pas être "parfaite" pour être utilisée dans le réseau de neurones. En effet, les lignes peuvent ne pas être horizontales, il peut y avoir une couleur de fond autre que blanc ou encore avoir du "bruit", c'est-à-dire des pixels dont la couleur tranche avec celle des pixels voisins et qui parasitent l'image. Différents traitements sont donc nécessaires pour corriger ces imperfections avant la suite des opérations.

Nous avons implémenté une rotation manuelle, une réduction des bruits et une binarisation de l'image.

4.2.1 Rotation de l'image

Les algorithmes que nous utilisons pour détecter les lignes de texte d'un document ont besoin d'une image source où les lignes et les paragraphes sont correctement alignés sur un axe horizontal.

Malheureusement, il est très courant que la numérisation d'un document ne se fasse pas d'une manière parfaitement horizontale par exemple, lorsque le document est pris en photo. Même si le décalage est minime, l'angle induit peut dégrader la reconnaissance du texte.

Cet angle doit donc être corrigé. Nous avons choisi de procéder par une rotation manuelle de l'image par l'utilisateur à l'aide de boutons accessibles sur l'interface graphique de notre logiciel.

L'utilisateur peut tourner l'image selon quatre angles différents : de 1° , 10° et 90° vers la gauche, ainsi que de 10° vers la droite.

Pour effectuer la rotation, nous avons choisi d'utiliser une librairie complémentaire à *SDL2*. La librairie *SDL_GFX* permet d'utiliser plusieurs fonctions de modification d'images sous le format de *SDL*. Parmi ces fonctions, nous avons utilisé *rotozoom* qui permet de pivoter une image ainsi que de faire un zoom sur celle-ci (fonctionnalité que nous utiliserons aussi plus tard dans notre logiciel.)

La fonction *rotozoom* prend quatre paramètres : l'image à modifier, l'angle de rotation, le degré de zoom (non utilisé ici donc placé à 1) et l'activation ou non de l'anti-crénelage. Elle renvoie ensuite une nouvelle image avec les modifications demandées.

Cependant, la rotation d'une image entraîne une modification de sa taille et donc l'apparition de nouveaux pixels pour combler la différence. Pour éviter toute interférence avec les prochains algorithmes, ces pixels doivent être blancs pour être

identifiés comme faisant parti du fond de l'image. Une nouvelle surface a alors été créée et remplie de pixels blancs. Puis, l'image obtenue par rotation a été copiée sur celle-ci. La taille de la nouvelle surface créée dépend de l'angle de rotation choisi par l'utilisateur.

Image de test d'origine et image après rotation par l'utilisateur

Lorem ipsum dolor sit amet, consectetur adipisciing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit
anim id est laborum.

Sed ut perspiciatis unde omnis iste natus error sit
voluptatem accusantium doloremque laudantium, totam rem
aperiam, eaque ipsa quae ab illo inventore veritatis et quasi
architecto beatae vitae dicta sunt explicabo.

Lorem ipsum dolor sit amet, consectetur adipisciing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit
anim id est laborum.

Sed ut perspiciatis unde omnis iste natus error sit
voluptatem accusantium doloremque laudantium, totam rem
aperiam, eaque ipsa quae ab illo inventore veritatis et quasi
architecto beatae vitae dicta sunt explicabo.

En fonction du bouton sélectionné par l'utilisateur, la fonction de rotation est appelée avec en paramètre l'angle correspondant (1° , 10° , 90° ou -10°). L'image affichée à l'écran est ensuite modifiée pour permettre à l'utilisateur de vérifier si l'angle de rotation sélectionné est suffisant.

4.2.2 Réduction des bruits

Comme dit précédemment, le but de la réduction des bruits est d'atténuer les imperfections de l'image donnée. Pour cela, plusieurs méthodes existent.

Nous avons choisi d'implémenter le filtre médian. Ce filtre permet de diminuer les bruits tout en conservant les contours des objets présents sur une image ce qui est très important pour nous car les lettres ne doivent pas être modifiées. Nous conservons ainsi les caractères.

Le principe de l'algorithme est de remplacer chaque pixel par la valeur médiane des pixels de son voisinage dans un certain rayon. Plus le rayon est grand, plus le filtre floute l'image. Nous avons choisi un rayon de 1 pour réduire les bruits sans

flouter les lettres correspondant à une matrice 3x3.

Avant application du filtre médian sur le pixel de valeur 50

5	6	7
6	50	8
7	8	9

Il faut tout d'abord trier ces valeurs par ordre croissant, afin d'avoir la valeur médiane à la case 4 (le milieu) du tableau.

Tri par ordre croissant des valeurs avoisinantes du pixel de valeur 50

5	6	6	7	7	8	8	9	50
---	---	---	---	----------	---	---	---	----

On peut ainsi en déduire que la valeur médiane est de 7. Il ne reste plus qu'à remplacer la valeur d'origine par la valeur obtenue grâce au filtre médian.

Après application du filtre médian sur le pixel de valeur 50

5	6	7
6	7	8
7	8	9

Image de lions d'origine et image après réduction des bruits



4.2.3 Binarisation

La binarisation est une étape incontournable avant la détection des lignes et des caractères. En effet, sans elle, l'image est remplie de différentes nuances de gris et les algorithmes seraient bien plus complexes sans cette binarisation. Nous avons choisi d'implémenter la méthode d'Otsu mais de nombreuses autres existent.

- Méthode classique

La méthode classique utilise un seuil fixe. Elle consiste à dire que tout pixel ayant une valeur de nuance de gris supérieure à 127 devient un pixel blanc et les autres des pixels noirs. Cette méthode implique que le texte est de couleur suffisamment foncée et le fond de l'image suffisamment clair.

$$\text{Bin(greyscale)} = \begin{cases} 0 & \text{si } \text{greyscale} > 127 \\ 255 & \text{sinon} \end{cases}$$

Image de test d'origine et image après binarisation

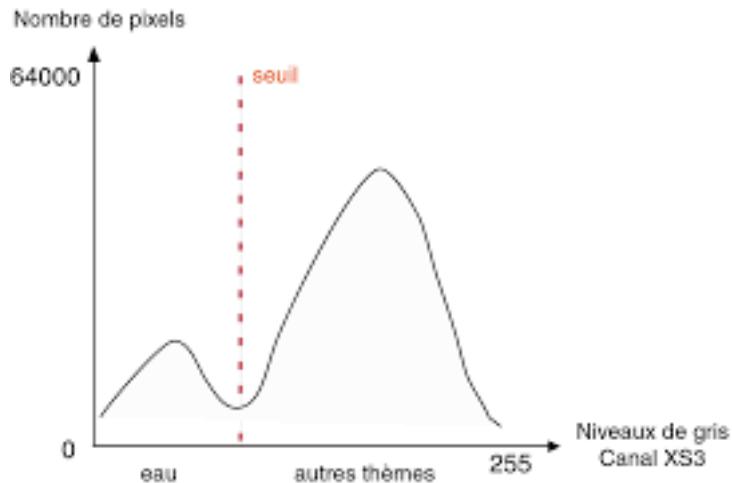


- Méthode d'Otsu

La méthode d'Otsu, au contraire, utilise un seuil automatique. Nous calculons le meilleur seuil de binarisation pour permettre de gérer les cas où le fond de l'image est sombre ou le texte clair. Cet algorithme se base sur le fait qu'une image dans la plupart des cas ne contient que deux plans, le premier plan et l'arrière plan. L'algorithme calcule le seuil optimal qui sépare les deux plans de l'image.

Les différents calculs de ce seuil se basent sur la probabilité qu'un pixel soit d'une certaine nuance de gris ou non. Pour cela, un histogramme des nuances de gris est nécessaire. Cela va nous permettre de recréer virtuellement un graphique en deux dimensions montrant à l'aide de "pics" le nombre de pixels qui ont un niveau de gris en commun. Les niveaux de gris sont classés par ordre croissant. A un niveau de gris particulier, on va pouvoir noter une très grande différence du nombre de pixels ayant une valeur de gris i et un niveau de gris $i+1$. Cette valeur est le seuil de binarisation optimal. Pour le déterminer d'autres calculs sont obligatoires.

Histogramme d'une image constituée d'eau et d'un décor



Il est ensuite nécessaire de normaliser cet histogramme pour obtenir la probabilité qu'un pixel soit d'une nuance donnée. Pour cela, nous divisons chaque valeur de l'histogramme par le nombre de pixels total de l'image. Nous obtenons ainsi un tableau de probabilités.

Ces probabilités sont ensuite divisées en deux classes : la probabilité que la nuance de gris fasse partie du fond de l'image et la probabilité qu'elle fasse partie des objets de l'image (ici les caractères). La première classe s'obtient en faisant la somme de toutes les probabilités de cette classe :

$$\omega_{C1}(i) = \sum_{k=1}^i \text{Proba}(i)$$

avec :

i : la nuance de gris de valeur i sur laquelle on travaille

$\omega_{C1}(i)$: la probabilité que i appartienne à la classe $C1$

$\text{Proba}(i)$: la probabilité qu'un pixel soit de nuance i d'après le tableau de probabilités

De plus, on remarque facilement que la deuxième classe découle de la première car la somme des probabilités vaut 1 :

$$\omega_{C2}(i) = 1 - \omega_{C1}(i)$$

avec :

i : la nuance de gris de valeur i sur laquelle on travaille

$\omega_{C1}(i)$: la probabilité que i appartienne à la classe $C1$

$\omega_{C2}(i)$: la probabilité que i appartienne à la classe $C2$

On peut ensuite calculer la moyenne de chaque classe qui correspond à la somme des probabilités multipliées par la valeur de la nuance de gris de cette probabilité :

$$Moy_{C1}(i) = \sum_{k=1}^i (k * Proba(k))$$

avec :

i : la nuance de gris de valeur i sur laquelle on travaille

$Moy_{C1}(i)$: la moyenne de la classe $C1$ de 0 à i

$Proba(i)$: la probabilité qu'un pixel soit de nuance i d'après le tableau de probabilités

La méthode d'Otsu va maintenant calculer la variance interclasse entre les classes $C1$ et $C2$ pour tous les i possibles (de 0 à 255). La variance interclasse caractérise la différence qui existe entre les pixels des deux classes. Plus elle est grande, moins les deux classes se ressemblent. La réciproque est également vraie.

Par conséquent, le seuil optimal est le i qui obtient la plus haute variance interclasse.

La variance interclasse entre les classes $C1$ et $C2$ correspond à :

$$Var_{C1interC2}(i) = \frac{(Moy * \omega_{C1}(i) - Moy_{C1}(i))^2}{\omega_{C1}(i) * \omega_{C2}(i)} = \frac{(Moy * \omega_{C1}(i) - Moy_{C1}(i))^2}{\omega_{C1}(i) * (1 - \omega_{C1}(i))}$$

avec :

i : la nuance de gris de valeur i sur laquelle on travaille

$\omega_{C1}(i)$: la probabilité que i appartienne à la classe $C1$

$\omega_{C2}(i)$: la probabilité que i appartienne à la classe $C2$

$Moy_{C1}(i)$: la moyenne de la classe $C1$ de 0 à i

Moy : la moyenne de l'image soit la moyenne de la classe $C1$ pour 255

Ainsi en calculant la variance interclasse entre les classes $C1$ et $C2$ pour tous les i de 0 à 255, on peut trouver pour quelle valeur de i cette variance est maximale. Il ne reste plus qu'à binariser l'image de la même façon qu'avec la méthode classique mais avec un seuil de binarisation valant i et non plus 127.

$$\text{Bin(greyscale)} = \begin{cases} 0 & \text{si greyscale} > i \\ 255 & \text{sinon} \end{cases}$$

Image de test d'origine et image après binarisation avec la méthode d'Otsu



4.3 Détection et découpage en lignes et caractères

Les images qui vont être traitées lors de cette partie l'ont déjà été lors du pré-traitement. Autrement dit les images traitées seront composées uniquement de pixels blancs ou noirs. Avant le traitement, toutes les valeurs de couleurs seront donc identiques pour chaque pixel.

La segmentation est une partie très importante lors de la réalisation de l'OCR. En effet, si le réseau de neurones fonctionne, mais que les images d'entrée ne sont pas bonnes, il sera alors difficile ou impossible d'arriver à un résultat correct.

Nous considérerons pour le découpage que les caractères sont des rectangles de pixels. Ainsi nous allons découper en lignes de caractères puis en colonnes entre caractères. Nous avons utilisé deux couleurs pour ce découpage, le rouge pour les interlignes et le vert pour les espaces entre caractères. Nous pourrons ainsi reconnaître les changements de paragraphe et les espaces facilement.

Pour la suite de cette partie, nous travaillerons sur l'image suivante qui évoluera au fur et à mesure des parties.

Image de texte d'origine

**Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed
 non risus. Suspendisse lectus tortor, dignissim sit amet,
 adipiscing nec, ultricies sed, dolor.**

**Cras elementum ultrices diam. Maecenas ligula massa, varius a,
 semper congue, euismod non, mi.**

4.3.1 Découpage en lignes

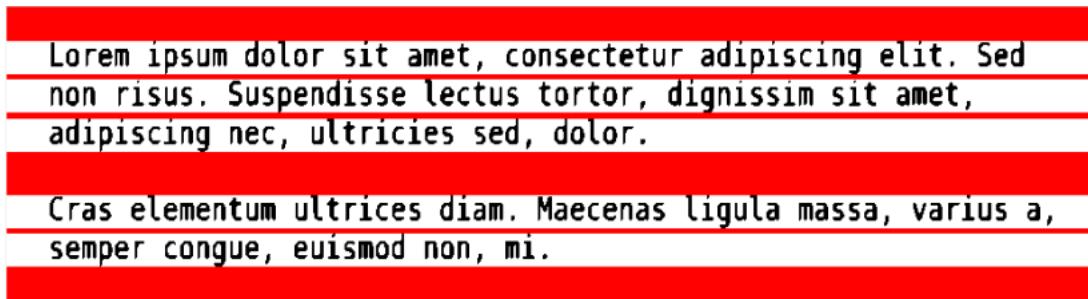
Le découpage en ligne est la première étape pour extraire des caractères. Avant de commencer, il est nécessaire de définir ce qu'est une ligne. Une ligne est "une série de choses disposées selon une direction donnée" (d'après le Dictionnaire Hachette). Ainsi pour nous, une ligne est un ensemble de caractères alignés séparés par une bande blanche grâce à la binarisation.

Pour le découpage en ligne, nous avons donc choisi de chercher les lignes composées uniquement de pixels blancs qui correspondent aux lignes entre les caractères.

On parcourt la ligne de pixels actuelle : si elle est entièrement composée de pixels blancs alors on remplace l'ensemble de ces pixels par des pixels rouges, soit de valeur $(r, g, b) = (255, 0, 0)$. Si un pixel noir est trouvé, on passe à la ligne suivante car la ligne actuelle correspond à une ligne de caractères.

On applique ce procédé de recherche sur toutes les lignes de l'image ce qui permet de supprimer les marges hautes et basses d'un texte en même temps que les espaces entre les lignes.

Image de texte après découpage des lignes



4.3.2 Découpage en caractères

Le découpage en caractères consiste à isoler chaque caractère d'une ligne des autres. Un caractère est défini par un ensemble variable de pixels noirs très rapprochés entourés de pixels blancs. Ainsi, chaque caractère est séparé par une colonne blanche grâce à la binarisation.

Pour le découpage en caractères, nous avons donc choisi de chercher les colonnes composées uniquement de pixels blancs qui correspondent aux espaces entre les caractères. Cette recherche doit être appliquée sur chaque ligne découpée précédemment.

Avant de chercher les espaces, il est nécessaire de déterminer les lignes de pixels sur lesquelles il faut tester. Pour cela, nous regardons le premier pixel d'une ligne :

Si, il est rouge, la ligne est un espace et ne contient pas de caractère, on passe alors à la ligne suivante.

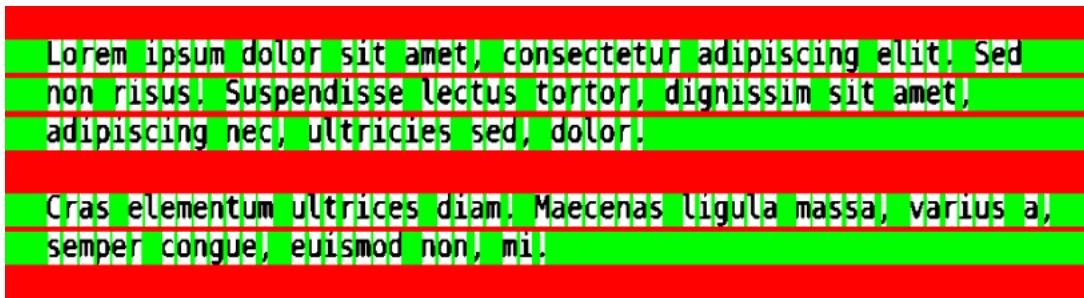
Sinon, cela veut dire que cette ligne de pixels appartient à une ligne de caractères. Dans ce cas, on note *begin* l'index de cette ligne et on commence à chercher la fin de la ligne de caractères. On cherche alors la première ligne suivante pour laquelle le premier pixel est rouge. Une fois trouvé, on note *end* l'index précédent.

Maintenant que nous connaissons les lignes de pixels sur lesquelles il faut travailler, on peut appliquer un algorithme semblable à celui des lignes entre *begin* et *end*.

Pour reconnaître les espaces entre les caractères, on regarde ensuite pour toute la longueur de la ligne si la colonne entre *begin* et *end* est entièrement composée

de pixels blancs. Cela signifie qu'aucun caractère n'est contenu dessus, on remplace l'entièreté de ces pixels par des pixels verts, soit de valeur $(r, g, b) = (0, 255, 0)$. Si un pixel noir est trouvé, on passe à la colonne suivante car la colonne actuelle correspond à un caractère.

Image de texte après découpage des caractères



4.4 Extraction et mise en forme des caractères

Les algorithmes précédents ont permis de marquer sur l'image où se trouvent les interlignes, les marges et les espaces entre caractères. Maintenant ces caractères vont être extraits de l'image, modifiés pour être uniformisés et mis en forme pour être utilisés par le réseau de neurones.

4.4.1 Extraction de chaque caractère

Un caractère est un rectangle de pixel compris entre deux lignes de pixels rouges et deux colonnes de pixels verts.

Pour extraire les caractères nous itérons sur le premier pixel de chaque ligne. Si le pixel n'est pas rouge , il s'agit d'une ligne de caractères, sinon on passe à la ligne suivante. Une fois sur une ligne de caractères, on procède de manière similaire, nous itérons sur tous les pixels de la ligne. Si le pixel n'est pas vert, il s'agit d'un caractère, sinon on passe au pixel suivant.

Une fois qu'un caractère est trouvé, une fonction d'extraction est appelée. Cette fonction détermine la hauteur et la largeur du caractère en cherchant respectivement le prochain pixel rouge présent sur la colonne et vert présent sur la ligne. Elle crée ensuite une nouvelle image de la taille du caractère et l'y copie pixel par pixel.

Extraction du caractère L



Le caractère extrait est alors renvoyé pour être utilisé dans la suite du programme.

4.4.2 Uniformisation du caractère

Une fois le caractère extrait, il doit être modifié afin d'être uniformisé. En effet, pour que le réseau de neurones détecte les caractères ils doivent être identiques. Cependant, lors de son extraction de l'image un caractère ne fait pas obligatoirement la taille attendue et des pixels blancs peuvent agrandir l'image inutilement.

- Recadrage du caractère

Des pixels blancs peuvent être présents principalement au dessus et en dessous du caractère dû aux lettres comme les p, f, l ou q. Pour supprimer ces lignes et colonnes de pixels blancs, nous appliquons nos algorithmes de détection des lignes et des caractères (Parties 4.3.1 et 4.3.2), puis une fonction semblable à celle d'extraction. En effet, nous devons aussi traiter les cas où le caractère est composé de plusieurs parties comme le i ou le !.

Pour cela, nous avons modifié la façon dont la largeur et la hauteur du caractère est calculée. Nous partons du principe que la taille du caractère correspond à la taille de l'image réduite de la taille des marges rouges et vertes. Ainsi nous déterminons le nombre de lignes rouges et de colonnes vertes présentes sur l'image et nous les soustrayons respectivement à la hauteur de l'image et à la largeur de l'image.

Recadrage du caractère L



Le caractère est ensuite copié dans une nouvelle image et retourné.

- Redimensionnement du caractère

Le caractère étant maintenant correctement découpé, il doit être redimensionné pour correspondre aux attentes du réseau de neurones. Pour cela, nous utilisons une fonction présentée plus-tôt : *rotozoom*. Le 2^{me} paramètre correspondant à l'angle de rotation est mis à 0 et le 3^{me} correspondant au degré de zoom est calculé en fonction de la taille de l'image contenant le caractère.

Le degré de zoom correspond à la taille de l'image recherchée soit 30 divisé par la caractéristique de taille la plus grande entre la largeur et la hauteur de l'image.

L'image redimensionnée créée est renvoyée pour être utilisée par le réseau de neurones.

4.4.3 Appel au réseau de neurones

Le réseau de neurones prend en entrée un tableau de *doubles* de taille 30*30 contenant uniquement des 0 et des 1. Ainsi l'image du caractère est convertie en un tableau de *doubles* de taille 30*30.

Pour cela, tous les pixels de l'image sont parcourus, lus et copiés dans un tableau préalablement créé. Si le pixel est noir, soit de valeur $(r, g, b) = (0, 0, 0)$, il sera retranscrit dans le tableau comme valant 1. Si le pixel est blanc, soit de valeur $(r, g, b) = (255, 255, 255)$, il sera retranscrit dans le tableau comme valant 0.

Le réseau de neurones est ensuite appelé sur ce tableau de valeurs. Il renvoie la lettre correspondant à ce tableau.

4.4.4 Mise en page

Les fonctions précédentes découpent le texte en lignes et en caractères sans distinction pour les paragraphes et les espaces entre les mots. Il est donc nécessaire d'implémenter une fonction permettant de les détecter.

Les paragraphes sont une "subdivision d'un texte en prose, typographiquement définie" (d'après le Dictionnaire Hachette). Ainsi un paragraphe est composé d'une typographie globalement constante qui peut se traduire par un espace plus important entre deux lignes de texte. Pour déterminer l'écart moyen entre deux lignes, nous parcourons le premier pixel de toutes les lignes (en excluant la marge haute et la marge basse qui pourraient fausser les calculs) de l'image. En divisant le nombre de pixels rouges obtenu par le nombre d'interlignes, nous obtenons une valeur moyenne de l'espacement. Si l'espacement est supérieur à cette valeur alors il s'agit d'un nouveau paragraphe sinon d'un simple interligne.

La différence entre un espace inter-caractères et un espace entre deux mots est la largeur de cet espace. En général, un espace entre deux mots est au minimum trois fois plus gros qu'un espace entre caractères. Nous utilisons un algorithme semblable à celui de la détection des paragraphes. En parcourant toutes les lignes non rouges de l'image (en excluant la marge gauche et la marge droite qui pourraient fausser les calculs), nous comptons le nombre total de pixels verts ainsi que le nombre d'espaces. En calculant la moyenne des espaces d'un texte, nous pouvons déterminer à quel type d'espace nous avons à faire pour chaque cas.

Cependant, des cas particuliers doivent être traités. Ces situations sont traitées de la même façon pour les interlignes et les espaces entre caractères.

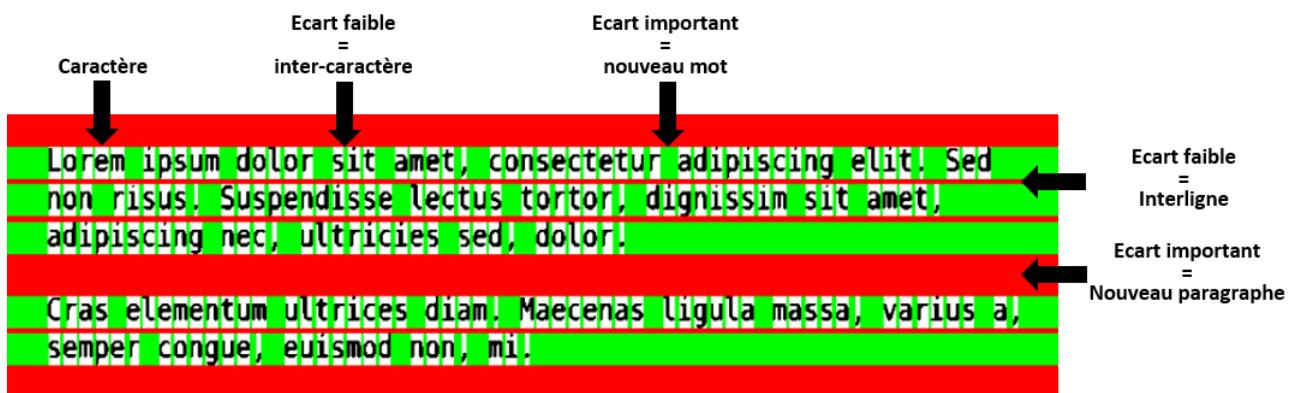
- Si le texte contient plusieurs lignes mais aucun paragraphe, la moitié des lignes seront traitées comme des nouveaux paragraphes. Nous multiplions la taille moyenne d'un interligne par 1,25 pour éviter cette confusion.

- Si le texte ne contient qu'une seule ligne, il n'y a aucun interligne donc nous renvoyons une valeur sans importance (le nombre de pixels rouges trouvés).
- Si le texte contient moins de quatre lignes, aucune moyenne significative ne peut être calculée. Nous multiplions le résultat obtenu par 10 pour traiter tous les espaces comme des interlignes.

Le calcul de ces moyennes se fait en amont du parcours de l'image pour le traitement des caractères. Ainsi, lors de ce parcours, nous analysons les caractères, les espaces et les interlignes en même temps afin de retourner le plus fidèlement possible la mise en page du texte de l'image.

En effet, pour retourner le texte nous écrivons dans le fichier "*text.txt*" tous les caractères juste après leur détection. Ainsi lorsqu'un espace entre deux caractères est analysé comme étant un espace entre deux mots, un espace est écrit dans le fichier avant la détection des lettres suivantes.

Détection des différents éléments sur une image



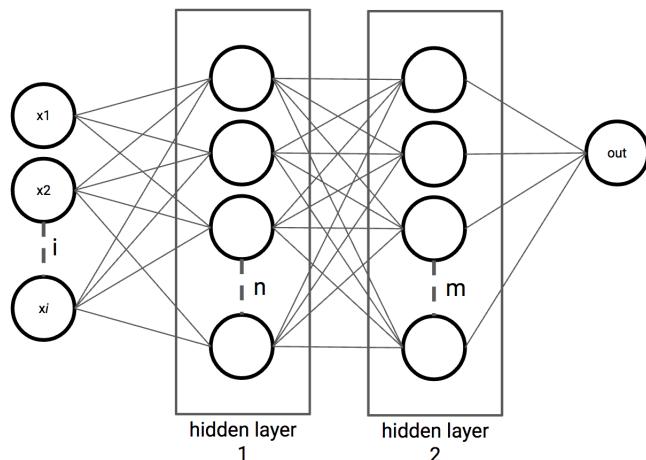
4.5 Réseau de neurones

4.5.1 Définition et fonctionnement

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

En informatique, un neurone est simplement une case mémoire qui contient un chiffre, très souvent entre 0 et 1, qui est sa valeur d'activation. Lorsqu'on agence des neurones en couches et que nous les relions entre elles, nous obtenons un réseau de neurones. On appellera la première couche (celle où l'information à traiter est entrée) couche d'entrée, la dernière (celle qui contiendra la réponse du réseau) couche de sortie, et toutes les autres entre ces deux-là seront nommées couches cachées. Pour les relier entre elles, il faut mettre en place une série de poids qui déterminera l'influence de chaque neurone sur chaque neurone de la couche suivante. En plus de cela, il faut mettre en place pour chaque neurone un biais, qui est un indicateur de sa tendance à être activé (proche de 1) ou éteint (proche de 0).

Exemple de représentation d'un réseau de neurones avec ses poids



- Comment fonctionne un réseau de neurones ?

Pour le calcul de la valeur d'activation d'un neurone d'une couche, il faut effectuer la somme pondérée des activations de la couche précédente multipliée par les poids reliant chaque neurone de la couche précédente à celui que l'on souhaite calculer. Reste ensuite à ajouter le biais du neurone au résultat. Il est possible et pratiquement obligatoire dans les cas complexes d'utiliser une fonction d'activation qui forcera le résultat des calculs à rester dans l'intervalle d'activation des neurones. Les trois possibilités pour ce travail, qui sont les plus utilisées dans les réseaux de neurones sont :

- La fonction *Sigmoïd* : $f(x) = \frac{1}{1 - e^x}$
- La fonction *ReLU* : $f(x) = \max(0, x)$
- La fonction *Softmax* : $\delta(z)_j = \frac{e_j^z}{\sum_{k=1}^K e_k^z}$ pour tout j de $\{1, \dots, K\}$.

Au final, on peut condenser le passage d'une couche $i-1$ à la suivante ainsi :

$$a_i = f(W * a_{i-1} + B)$$

a_i : vecteur contenant la couche i .

a_{i-1} : vecteur contenant la couche $i-1$

W : matrice des poids de dimension taille de a_i par taille de a_{i-1}

B : vecteur contenant les biais de la couche a_i

f : fonction d'activation appliquée à chaque élément de a_i

Il faut ensuite répéter l'opération de la première à la dernière couche.

- Comment entraîner un réseau de neurones ?

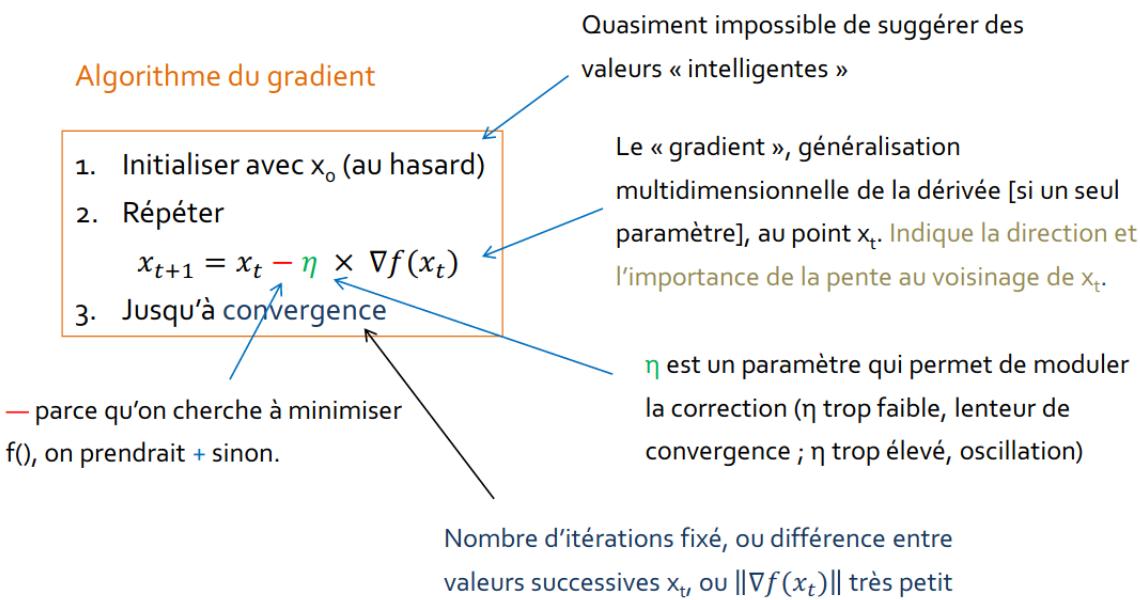
L'entraînement du réseau consistera à calibrer tous les poids et biais jusqu'à ce que la réponse soit au plus proche de ce que l'on souhaite.

Pour commencer, il faut savoir à quel point le réseau est loin de ce que l'on attend de lui. Pour cela on utilise une fonction, appelée fonction de coût. Cette fonction prend en paramètre un vecteur contenant les sorties du réseau et un autre vecteur contenant les réponses attendues pour chaque neurone de la dernière couche pour une entrée donnée. La fonction fera la somme des différences au carré entre ce que l'on a et ce que l'on veut pour chaque neurone de la couche de sortie, donc la somme des distances au carré entre notre modèle et celui vers lequel on voudrait tendre.

Le but de l'entraînement sera de minimiser cette fonction de coût, donc d'arriver à un réseau qui pour chaque entrée donne des réponses très proches de ce que l'on souhaite.

En mathématiques, pour minimiser une fonction, une méthode efficace est la descente de gradient :

Algorithme du gradient (Source : univ-lyon2)



Pour entraîner un réseau de neurones il faudra réaliser l'algorithme du gradient pour chaque poids et chaque biais, jusqu'à minimisation de la fonction de coût.

4.5.2 Création du réseau

Le réseau est composé de trois couches distinctes :

- La couche d'entrée est composée de 900 neurones (les pixels d'une image de 30*30)
- La couche cachée comporte 30 neurones, qui font la jonction entre l'entrée et la sortie.
- La couche de sortie possède 70 ($=2*26+10+8$) neurones, représentant les lettres de l'alphabet en majuscule et minuscule, les chiffres de 0 à 9 ainsi que les caractères suivants : ., ?, !, -, (,)' .

De plus, nous avons décidé d'utiliser une représentation matricielle pour ce réseau de neurones. Chaque couche est donc représentée par un tableau contenant les valeurs d'activation des différents neurones de cette couche. Les biais sont aussi représentés dans un tableau, tandis que les poids sont dans une représentation en une dimension d'une matrice.

Comme notre réseau est composé de trois couches, notre implémentation est donc composée de trois tableaux pour les valeurs d'activation des neurones des trois couches, deux tableaux de poids et deux tableaux de biais. Pour chaque couple de tableaux de poids et de biais, le premier relie la première couche à la deuxième et le deuxième tableau relie la deuxième couche à la troisième couche. Tous ces tableaux font partie d'une même structure afin de grandement faciliter les appels de fonctions.

De plus, les tailles des couches sont définies comme constantes globales afin d'être facilement accessibles dans tout le code. Cela permet aussi de retrouver facilement le nombre de lignes et de colonnes dans la matrice de poids représentée sous forme de tableau. Voici la définition de la structure "*network*" :

```
// different lengths for the neural network
#define INPUT_LEN 30*30
#define HIDDEN_LEN 30
#define OUTPUT_LEN 2*26+18

// struct for the neural network
// contains all the 3 layers: input, hidden and output
// and all weights and bias
typedef struct network
{
    //layers of the network, lengthes always matches the network's
    double input[INPUT_LEN];
    double hidden[HIDDEN_LEN];
    double output[OUTPUT_LEN];

    //weights and biases of the network, lengthes always matches the network's
    double weights01[INPUT_LEN*HIDDEN_LEN];
    double weights12[HIDDEN_LEN*OUTPUT_LEN];
    double bias01[HIDDEN_LEN];
    double bias12[OUTPUT_LEN];

} network;
```

Ainsi pour une couche n , le poids w^n_{ij} relie le $i^{i\text{eme}}$ neurone de la $n^{i\text{eme}}$ couche au $j^{i\text{eme}}$ neurone de la $n + 1^{i\text{eme}}$ couche, et de même pour le biais b^n_j .

Il a ensuite fallu initialiser cette structure. Pour cela, trois choix sont possibles :

- Initialiser tous les poids et biais à 0 pour une initialisation rapide
- Initialiser de façon aléatoire les poids et biais grâce à une fonction dédiée qui itère sur chaque élément de chaque tableau de poids et de biais et y met un nombre à virgule flottante aléatoire entre 0 et 1
- Charger un réseau déjà existant avec une autre fonction qui lit les quelques 29200 éléments des tableaux de biais et poids du réseau pour les placer au bon endroit.

4.5.3 Exécution du réseau

Afin de pouvoir exécuter notre réseau de neurones, nous avons d'abord écrit les fonctions d'addition et de multiplication de matrices. Ensuite, pour l'exécution en elle-même, comme décrit plus haut, pour activer un neurone d'une couche n , nous sommes le produit des valeurs d'activation des neurones de la couche $n - 1$ avec les poids les reliant au neurone à activer et enfin nous ajoutons un biais à cette somme. Les résultats de chaque couche sont ensuite placés dans un tableau auquel nous appliquons la fonction *SoftMax*. La fonction *SoftMax* est une fonction qui prend en entrée un vecteur et qui retourne un vecteur de nombres réels strictement positifs et de somme 1.

Nous pouvons résumer cette exécution à l'aide de l'équation suivante :

$$\begin{bmatrix} a_0^n \\ \vdots \\ a_j^n \end{bmatrix} = \text{SoftMax} \left(\begin{bmatrix} w^{n-1}_{00} & \cdots & w^{n-1}_{i0} \\ \vdots & \ddots & \vdots \\ w^{n-1}_{0j} & \cdots & w^{n-1}_{ij} \end{bmatrix} \cdot \begin{bmatrix} a^{n-1}_0 \\ \vdots \\ a^{n-1}_i \end{bmatrix} + \begin{bmatrix} b^{n-1}_0 \\ \vdots \\ b^{n-1}_j \end{bmatrix} \right)$$

Pour plus de précision, on peut écrire :

$$\forall k \in \llbracket 0 ; j \rrbracket : a^n = \text{SoftMax}(w^{n-1} \cdot a^{n-1} + b^n)$$

avec :

a^n : tableau d'activation de la $n^{i\text{eme}}$ couche.

w^{n-1} : matrice poids reliant le la $n - 1^{i\text{eme}}$ couche à la $n^{i\text{eme}}$ couche.

b^n : tableau de biais pour l'activation de la $n^{i\text{eme}}$ couche

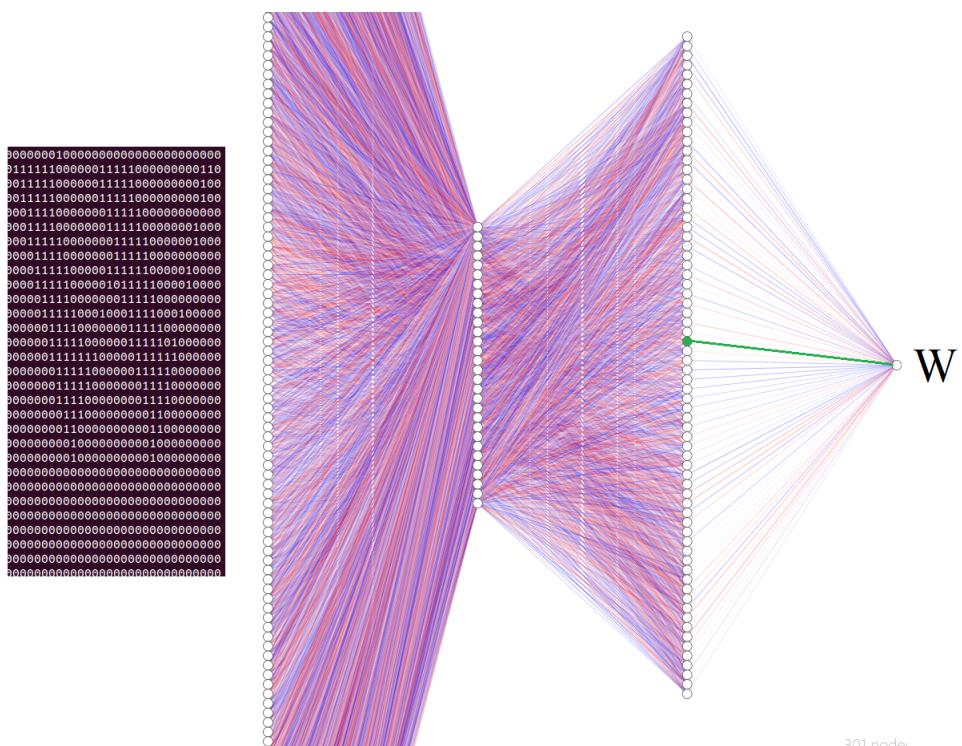
Intéressons-nous maintenant au cas de notre réseau cherchant à reconnaître des caractères.

Comme précisé plus haut, les différentes matrices sont représentées par des tableaux de flottants, et les valeurs d'activation des neurones de la première couche correspondent à la couleur des pixels de l'image (noir = 1, blanc = 0). Cela est possible grâce aux différents traitements que nous faisons subir à l'image et qui la rendent utilisable.

Contrairement au réseau de la preuve de concept, une fonction d'activation était obligatoire, et nous avons opté pour *SoftMax*, qui donnait de meilleurs résultats que *Sigmoïd* et *ReLU*. La différence principale est le fait que *SoftMax* lisse au maximum les résultats des activations sans en changer l'ordre. Enfin, dans la dernière couche nous considérons que le neurone avec la plus grande activation est celle qui représente le caractère qui a été reconnu par le réseau.

Une fonction permet, en prenant un réseau et la matrice 30*30 d'une image en entrée, de renvoyer un seul caractère, celui qu'elle aura trouvé comme étant le plus probablement celui présent sur l'image. Pour cela elle exécutera les formules vues ci-dessus pour calculer l'activation de la couche cachée en fonction de l'entrée, puis la couche de sortie en fonction de la couche cachée, et enfin parcourra la couche de sortie pour trouver la plus grande activation.

Représentation de l'exécution du réseau pour l'image d'un W (Times new roman)



4.5.4 Entraînement du réseau

Tout d'abord, il faut charger la collection d'entraînement en entier. Cela est fait en début de fonction et avant la boucle principale afin de ne pas avoir besoin de la recharger à chaque itération. Chaque élément de la collection sera représenté par une structure comportant trois éléments : la matrice 30*30 de l'image, le résultat (un *char*) attendu, ainsi que la couche de sortie idéale pour cette image.

Afin que le réseau évolue plus rapidement, les éléments de la collection sont organisés en "mini lots". A chaque itération de l'entraînement un des mini lots sera utilisé pour entraîner un peu plus le réseau. Cette technique est utilisée pour entraîner un réseau sur une large collection d'entraînement tout en ayant une vitesse d'entraînement plus efficace.

Pour charger la collection d'entraînement, on remplit donc une matrice de taille (nombre de mini lots)*(nombre d'éléments d'entraînement par lot) avec des structures d'éléments d'entraînement. On va donc itérer sur chaque case de cette matrice, charger un élément d'entraînement et l'y mettre.

Afin de charger les éléments de façon efficace et automatique, des matrices déjà préparées sont rangées dans un dossier à part, et leur nom correspond au symbole qu'elle doit représenter. Par exemple "*arial/5.txt*" contiendra la matrice d'un 5 en police Arial. Une fonction permet de charger une matrice à partir du nom du fichier dans laquelle elle est contenue et s'exécute donc pour chaque image de la collection d'entraînement.

Pour calculer le coût d'une exécution du réseau, il a fallu implémenter une fonction qui prend en argument deux tableaux (une sortie et la réponse attendue), et qui calcule la somme des écarts au carré pour chaque indice du tableau.

$$C(\text{réseau}) = \frac{1}{n} \sum_{i=0}^{n-1} (output[i] - wanted[i])^2$$

Cette fonction sert surtout à garder la trace de l'avancement de l'entraînement et de l'évolution de réseau.

Ensuite, vient le calcul du gradient. Pour ce faire, on itère sur chaque élément des différents tableaux de poids et biais, puis la dérivée du coût par rapport à cet élément est calculée de la façon suivante : (on nomme a l'activation d'un neurone à la couche i , qui est égal à la somme pondérée des activations de la couche précédente)

Si l'élément est entre la $2^{i\text{eme}}$ et la $3^{i\text{eme}}$ couche :

$$\frac{\partial C}{\partial \text{Element}} = \frac{\partial C}{\partial a} \frac{\partial a}{\partial Z} \frac{\partial Z}{\partial \text{Element}}$$

Si l'élément est entre la première et la $2^{i\text{eme}}$ couche :

$$\frac{\partial C}{\partial \text{Element}} = \sum_{i=0}^{n-1} \frac{\partial C}{\partial a_i} \frac{\partial a_i}{\partial Z_i} \frac{\partial Z_i}{\partial a_{i-1}} \frac{\partial a_{i-1}}{\partial Z_{i-1}} \frac{\partial Z_{i-1}}{\partial \text{Element}}$$

avec :

Le coût par rapport à une activation :

$$\frac{\partial C}{\partial a_i} = (\text{output}[i] - \text{wanted}[i])$$

La dérivée de *SoftMax* :

$$\frac{\partial a_i}{\partial Z_j} = a_i(\delta_{ij} - a_j); (i = j : \delta_{ij} = 1; i \neq j : \delta_{ij} = 0)$$

La dérivée d'une somme pondérée par rapport à un élément de la somme :

$$\frac{\partial Z}{\partial \text{Element}} =$$

- Si l'élément est un poids : l'activation multipliée par ce poids
- Si l'élément est une activation (pour $\frac{\partial Z_i}{\partial a_i}$) : le poids multiplié par cette activation
- Si l'élément est un biais : 1

Dans les faits, nous avons découpé le problème en trois fonctions de calculs de dérivées :

- La première calcule la dérivée partielle du coût par rapport à un élément entre les deux dernières couches
- La seconde calcule la partie "fixe" de la somme dans la formule entre les deux premières couches

- La dernière calcule la dérivée partielle du coût par rapport à un élément entre la couche d'entrée et la couche cachée

Le processus d'entraînement est donc le suivant :

Dans une fonction chapeau, on crée un nouveau réseau et on y charge les biais et poids sauvegardés préalablement dans un fichier texte, puis on l'envoie à la fonction principale. Dans cette fonction, on commence par récupérer les collections d'entraînements. Lorsqu'on les a chargés, on commence une boucle finie dans laquelle on commence par initialiser un vecteur pour y mettre toutes les dérivées partielles (le gradient total).

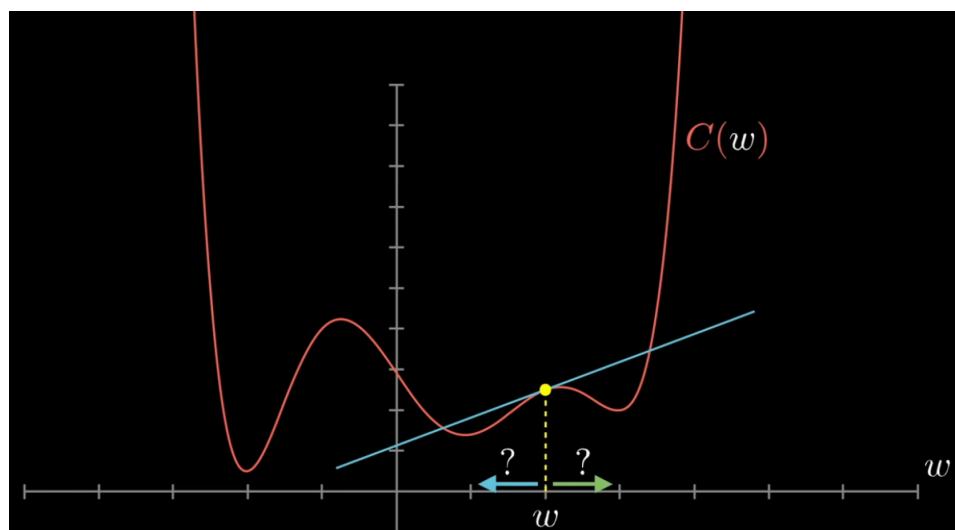
Ensuite on vérifie si le numéro de "mini lot" est valide, puis on exécute le réseau sur chaque élément du "mini lot", afin de récupérer les valeurs des trois couches qui résulteront de l'exécution du réseau. Grâce à ces informations, on peut calculer le coût de cette exécution, ainsi que son gradient. Le gradient actuel est ajouté au gradient total initialisé plus tôt et le coût est ajouté au coût total du mini lot.

Le coût total ainsi obtenu nous aide à savoir l'avancement de l'entraînement, et le gradient est rétro-propagé sur le réseau : on itère sur chaque élément des tableaux de poids et de biais et on enlève l'élément du gradient qui lui correspond : la dérivée partielle du coût par rapport à cet élément.

Enfin on incrémente l'indice du "mini lot" et le compteur de boucle, et une nouvelle itération de la boucle principale commence.

Lorsque le nombre d'itérations de l'entraînement voulu est atteint, on sort de la fonction d'entraînement.

Exemple du calcul du gradient du coût en fonction d'un paramètre



L'exécution du réseau, sa fabrication et la rétro-propagation du gradient sont fonctionnels.

Le calcul de ce dernier en revanche pose un problème. En effet, lorsque nous entraînons le réseau, la fonction de coût commence à baisser lentement mais sûrement. Au bout d'un moment, le coût commence à osciller autour de ce qui semble être un minimum local, qui est pourtant trop haut pour que le réseau soit efficace. Si l'on continue l'entraînement à partir de là, la fonction de coût peut, suivant les cas, osciller autour d'un minimum ou commence à diverger et le réseau se remplit alors de "nan".

Nous sommes rapidement arrivés à cet état du réseau. En rencontrant ce problème, nous avons essayé de nombreuses solutions différentes mais rien n'a pu régler le problème. Par exemple nous avons essayé avec les différentes possibilités de fonction d'activation (*SoftMax*, *Sigmoid*, *ReLU*) et sans, ce qui oblige à réécrire le calcul du gradient à chaque fois, mais *SoftMax* que nous avions utilisé en premier donnait les meilleurs résultats. Nous avons aussi essayé diverses tailles d'images et de couche cachée, mais aucun modèle ne prévalait sur un autre. Après avoir repensé et réécrit la majorité des fonctions une par une sans que les résultats n'évoluent, nous avons essayé de modifier les collections d'entraînement, de revoir en détail les points théoriques et repasser encore une fois chacune des fonctions en revue, mais nous n'avons au final pas réussi à obtenir le résultat escompté, la dernière partie du réseau n'est de ce fait pas fonctionnelle.

4.5.5 Création de la collection d'entraînement

La collection d'entraînement n'est pas, comme on pourrait pourtant s'y attendre, une collection d'images. Il s'agit en fait de matrices de 0 et de 1 représentant les pixels d'une image. Il composé des caractères les plus usuels (lettres, chiffres et quelques symboles, comme décrit plus tôt) de deux polices : Arial et Times new roman.

Pour créer ces matrices, nous avons utilisé le retour de la fonction de segmentation et de traitement de l'image, afin de s'entraîner sur des données les plus proches possibles de à quoi le réseau pourrait être confronté dans son utilisation. Les valeurs de ces matrices sont forcément 0 ou 1 car les images sont binarisées. De plus, les images sont redimensionnées pour faire exactement 30*30 pixels, malgré une taille de chaque lettre différente à la base.

Une fonction a permis d'automatiser la création de collections. Cette fonction découpe l'image ci-dessous et pour chaque lettre, affiche dans la console la matrice 30*30 correspondante où l'on reconnaît un caractère de l'image. Il suffit d'entrer le symbole correspondant pour enregistrer la matrice dans "*/(police)/(caractère).txt*". (à noter que "*..txt*" ne peut pas exister et à donc été remplacé par "*&.txt*". De plus, pour les tests sous Windows, deux fichiers "*A.txt*" et "*a.txt*" ne peuvent pas exister dans le même dossier, les majuscules ont donc la forme suivante : "*Aa.txt*").

a b c d e f g h i j k l m n o p q
r s t u v w x y z
A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 . ? ! , - ' :
()
a b c d e f g h i j k l m n o p q r s
t u v w x y z
A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 . ? ! , - ' : ()

4.5.6 Sauvegarde et chargement des poids et des biais du réseau

Pour sauvegarder les poids et les biais du réseau de neurones, nous utilisons la manipulation de fichiers. Nous ouvrons un fichier puis, pour chaque poids et biais du réseau de neurones, nous l'écrivons dans ce fichier. Tous les poids et biais écrivant dans ce fichier sont séparés les uns des autres par une virgule, afin de faciliter le chargement de ces données. Enfin, nous fermons ce fichier.

Pour charger les poids et les biais du réseau, nous ouvrons le fichier où sont sauvegardés ces derniers. Puis, nous récupérons chaque donnée dans un tableau et nous chargeons chacune de ces données dans le bon tableau de poids ou de biais. Enfin nous fermons le fichier.

4.6 Interface graphique

L'interface graphique est un élément essentiel de tout projet destiné à une utilisation grand public. En effet, elle permet l'accès aux fonctionnalités d'un programme de façon graphique. Ainsi l'utilisateur n'a pas besoin d'entrer des lignes de commandes dans un terminal pour utiliser les différentes options de notre logiciel.

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

- *GTK et Glade*

Nous avons choisi d'utiliser la librairie *GTK* pour notre projet car elle supporte le langage C. Après plusieurs recherches, elle s'est révélé être une des bibliothèques les plus utilisées et les plus documentées pour créer des interfaces graphiques en C. Elle possède une grande variété de contenus pour afficher, modifier et interagir avec des éléments de notre programme.



De plus, cela nous a permis d'utiliser *Glade*, un logiciel qui permet de créer des interfaces graphiques. Celui-ci permet en effet de créer, dans une interface graphique, les éléments d'une interface graphique *GTK*. L'utilisateur peut donc placer à la souris les différents éléments, tels qu'une image, une boîte de texte ou des boutons. Le logiciel créé ensuite un fichier *.glade* qu'il suffit d'importer dans notre code pour interagir avec les éléments programmés.

- Notre interface

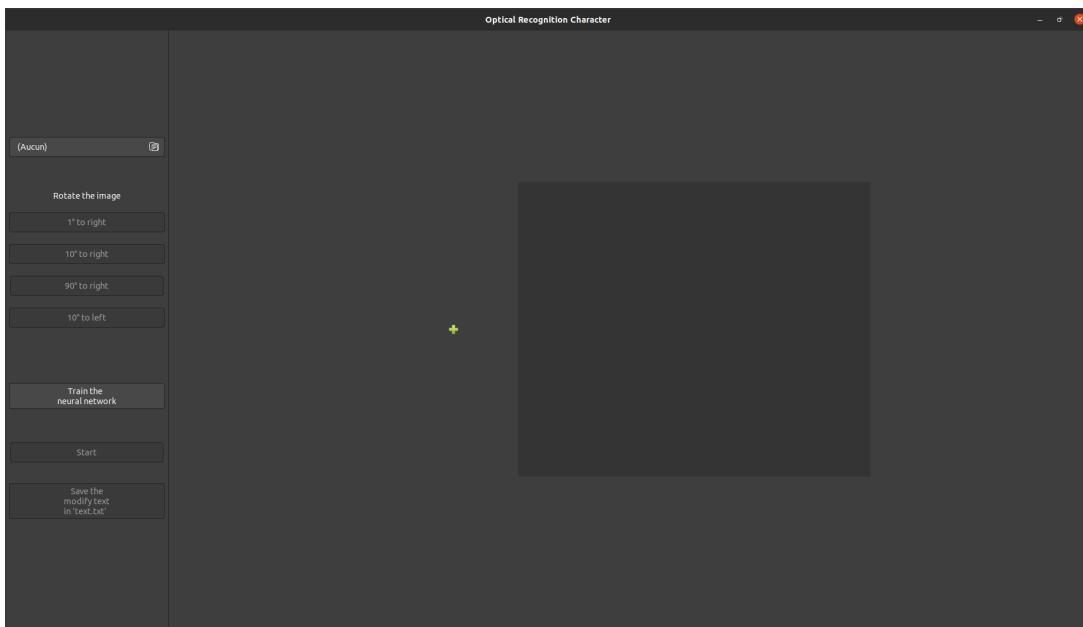
Notre interface est composée d'une seule fenêtre regroupant toutes les fonctionnalités accessibles à l'utilisateur. Elle est divisée en deux parties : la barre d'outils à gauche et la zone d'affichage à droite.

La barre d'outils regroupe tous les boutons correspondant aux possibilités de l'utilisateur. Au lancement du logiciel, seuls deux boutons sont activés :

- un bouton *GtkFileChooserButton* qui permet en cliquant dessus de naviguer dans l'explorateur de fichier. Lorsqu'une image est ouverte, elle est enregistrée dans le fichier "*tmp.bmp*" afin d'être affichée dans la zone d'affichage. De plus, cinq autres boutons sont activés.

- un bouton classique nommé *Train the neural network* qui permet de lancer l'entraînement du réseau de neurones sur 200 itérations. Le coût sera affiché dans le terminal toutes les cinq itérations. De plus une fenêtre de dialogue s'affichera à l'écran pour prévenir l'utilisateur que le logiciel se fermera à la fin de l'entraînement pour recharger le nouveau réseau de neurones.

Notre interface au lancement du logiciel

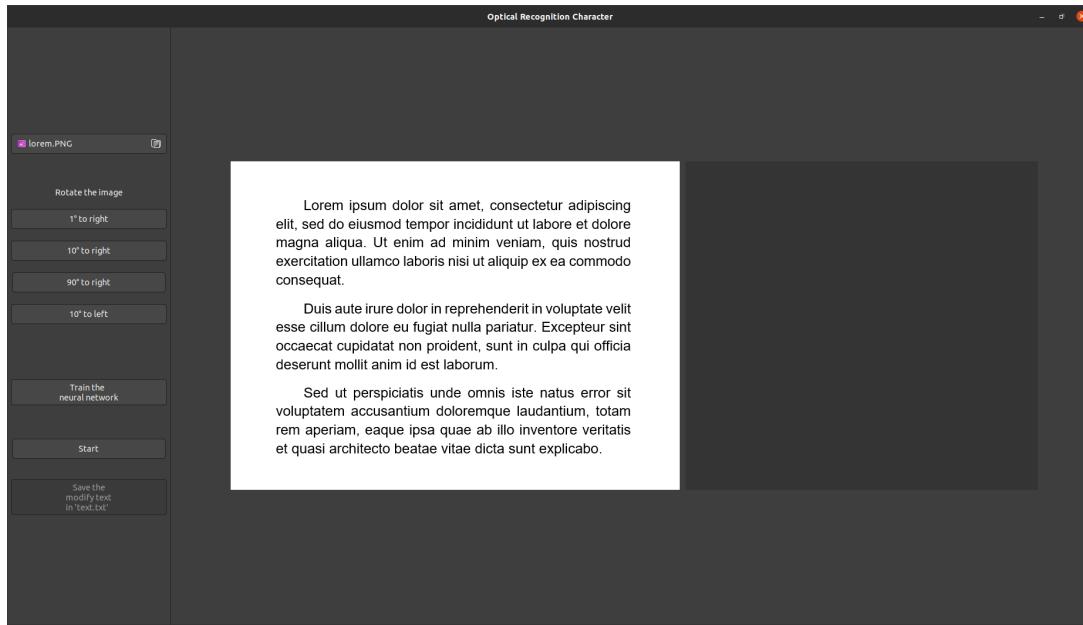


Après l'ouverture d'une image, d'autres boutons deviennent accessibles :

- quatre boutons présentés par un *Label* permettant d'appliquer une rotation sur l'image afin d'obtenir un texte horizontal. Chaque fois qu'une rotation est appliquée, la nouvelle image est sauvegardée dans le fichier "*tmp.bmp*" afin de mettre à jour l'image affichée dans la zone d'affichage.

- un bouton nommé *Start* qui permet de lancer la reconnaissance de caractères sur l'image choisie. A la fin du traitement de l'image, le texte détecté sera affiché dans la zone d'affichage.

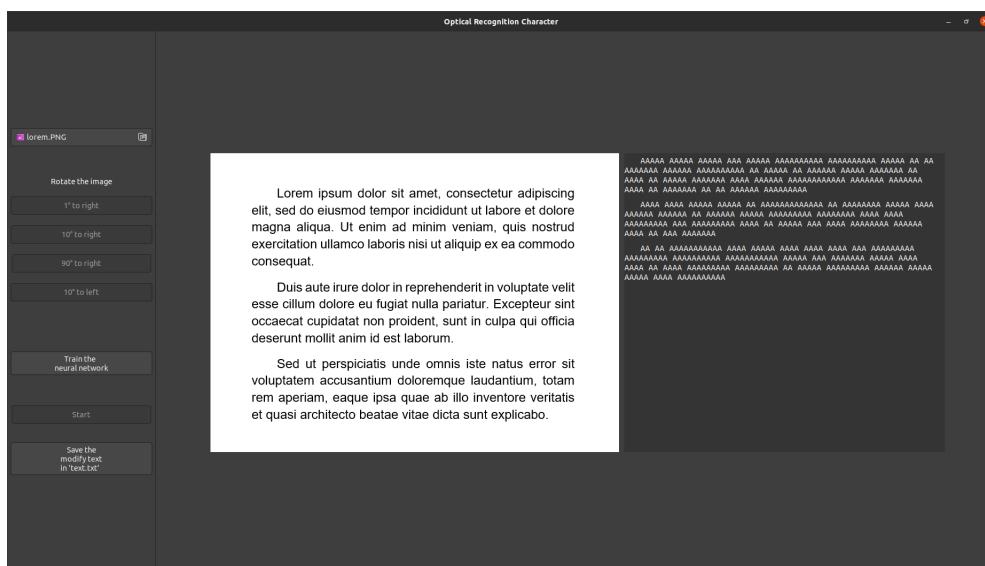
- un bouton nommé *Save the modified text in "text.txt"* qui permet d'enregistrer les modifications apportées au texte affiché dans la zone d'affichage. Ce bouton n'est activé qu'après le premier lancement de la reconnaissance de caractères. En effet, aucun texte n'a besoin d'être enregistré si aucune image n'a été analysée.

Notre interface après sélection d'une image

La zone d'affichage comporte deux espaces qui permettent à l'utilisateur de suivre ses actions dans le logiciel :

- à gauche se situe une zone permettant d'afficher l'image choisie avec le bouton correspondant. L'image se met à jour à chaque activation d'un bouton de rotation. Si aucune image n'a encore été choisie, un signe plus est affiché.

- à droite se situe une zone de texte permettant de charger le texte contenu dans dans le fichier "*text.txt*" qui contient le texte détecté par notre logiciel. Ce texte peut être modifié directement en sélectionnant la zone de texte.

Notre interface après lancement de la reconnaissance de caractères

5 Améliorations possibles

Notre logiciel n'étant pas parfait à la fin de ces trois mois de développement, nous avons pensé à plusieurs améliorations qui peuvent lui être apportées :

- Améliorer la descente de gradient du réseau de neurones pour qu'il puisse réellement apprendre à reconnaître des caractères
- Apprendre au réseau de neurones à reconnaître davantage de polices d'écriture en diversifiant la collection d'entraînement afin d'utiliser notre logiciel dans d'autres situations
- Entraîner le réseau de neurones sur de plus grandes collections d'entraînement pour lui permettre de reconnaître plus efficacement les caractères afin de minimiser les erreurs
- Retourner un texte dans la même mise en forme que l'image en conservant par exemple la police d'écriture ou les possibles images présentes à côté du texte.
- Améliorer le traitement de l'image pour être en mesure de traiter des images plus dégradées ou des textes en couleur afin d'utiliser le logiciel dans toujours plus de situations

6 Impressions personnelles

6.1 Hugo Abadie

Ce projet nous a permis de travailler sur des sujets que l'on ne connaissait pas, j'ai personnellement trouvé le sujet très intéressant et prenant. Cependant il a été difficile d'avancer sur ce projet avec très peu de documentation ou de connaissance fournie. Il a donc été nécessaire de se structurer très efficacement lors des recherches afin de pouvoir compléter les objectifs du projet. Cependant lorsque des problèmes de compréhension et/ou d'application ont été rencontrés aucun regard extérieur n'a pu nous indiquer là où nous nous étions trompés, il a donc fallu apprendre à prendre du recul sur le travail que nous effectuions. Cette absence de soutien a été fatale au déroulement du projet puisque en appliquant ce que l'on avait compris cela ne rendait pas le résultat escompté. Ce projet m'a également appris à manipuler de façon plus efficace le langage C et de pouvoir utiliser plusieurs structures différentes représentant une même chose et de pouvoir passer facilement de l'une à l'autre. Ce projet m'a également appris à pouvoir choisir la méthode optimale lorsqu'il possible de remplir des objectifs de plusieurs manières différentes.

Cependant malgré l'échec de n'avoir pu atteindre les objectifs initiaux demandés je reste satisfait du déroulement de ce projet. En effet je suis content d'avoir pu travailler avec des membres de groupe motivés et intéressés également par le sujet. Lors de la création du groupe la répartition des tâches a été rapidement effectuée ce qui nous a permis de nous focaliser et de nous imprégner rapidement du sujet, et ce malgré les contraintes sanitaires plus ou moins fortes auxquelles nous avons dû faire face. De plus, le groupe a été très soudé et personne ne travaillait vraiment dans son coin ou tout du moins en adéquation avec des idées que nous avions évoquées ensemble. Ainsi lors de l'emboîtement des parties aucun problème majeur n'est apparu grâce aux règles que nous avions établies entre nous.

De manière plus spécifique, j'ai beaucoup découvert sur les différents traitements d'images et filtres que nous pouvons appliquer et à quoi ils servent. Cela m'a ainsi donné une vision plus "concrète" de tout ce qui est appliqué sur les images grâce à des logiciels de retouche et de modification d'images. Je n'ai malheureusement pas réussi à implémenter toutes les fonctions que je voulais. Cependant cela m'a permis d'apprendre à traiter des données pouvant être impressionnantes comme en fait de nombreuses petites données simples afin de travailler dessus de manière efficace et rapide. Ce projet m'a également appris avec le traitement de l'image et en particulier les sous-traitements à résoudre des problèmes complexes en découplant ce problème en plusieurs étapes afin d'avoir quelque chose de plus facile à implémenter et à comprendre.

6.2 Magali Jomier

Ce projet m'a beaucoup apporté dans de nombreux domaines tant techniques que sociaux. Il m'a permis de découvrir de nombreux aspects du langage C que j'ai découvert tout au long du développement ainsi que plusieurs outils qui permettent de rendre ce langage plus performant. En effet, j'ai énormément appris sur la puissance du langage dans sa capacité à gérer la mémoire malgré les nombreuses réécritures successives de données. De plus, j'ai découvert pendant le développement le débogueur *gdb* qui m'a grandement aidé à régler les problèmes de gestion de la mémoire même s'il a été difficile à utiliser au début. J'ai aussi découvert le logiciel *Glade* qui m'a grandement aidé pour la réalisation de l'interface graphique.

Le projet m'a aussi énormément appris sur les innombrables traitements d'images qui existent. Même si je suis loin d'en avoir fait le tour, je me rends compte des progrès que j'ai fait sur ce domaine que je ne connaissais pratiquement pas au début de l'année. Le traitement d'image quel que soit l'objectif recherché se base sur des algorithmes complexes pour retourner le plus fidèlement l'image attendue. J'ai passé beaucoup de temps sur la segmentation des caractères et la reconstruction du texte et j'ai compris qu'il existait un grand nombre de situations auquel notre algorithme n'est pas toujours prêt à faire face comme des écritures en script où les caractères sont liés les uns aux autres rendant leur détection impossible. Cependant, je suis assez satisfaite du travail accompli dans ce laps de temps.

Ce projet m'a aussi permis de mettre en pratique les leçons tirées du projet de l'année dernière. Nous avons dès le début réparti les tâches entre les membres du groupe. Ainsi chacun a pu travailler sur un domaine qu'il souhaitait découvrir tout en aidant les autres. Nous avons très tôt commencé à travailler chacun de notre côté et la mise en commun des différentes parties a été très fluide. Nous sommes restés très organisés durant tout le projet. De plus, nous avons constamment gardé contact pour être à jour sur les avancées de chacun, nous soutenir et nous aider mutuellement. J'espère pouvoir retrouver une même cohésion de groupe sur mes prochains projets.

Malgré le bon déroulement de ce projet, j'ai quelques regrets. Je regrette beaucoup l'impossibilité de se rencontrer pendant la deuxième période de développement due au confinement. Chacun s'est retrouvé seul chez soi et cela a été parfois difficile lorsqu'on rencontrait des problèmes. Toutefois, nous avons pris l'habitude de régulièrement nous regrouper pour travailler en vocal ensemble. Je regrette aussi l'impossibilité que j'ai eu de vraiment approfondir l'étude d'un réseau de neurones due à la grande quantité de travail nécessaire sur mes parties. J'espère avoir le temps plus tard d'étudier un peu plus le sujet.

6.3 Ethan Machavoine

Ce projet a été pour moi très enrichissant sur plusieurs points.

Tout d'abord, l'intelligence artificielle est un sujet qui m'intéresse beaucoup, découvrir les réseaux de neurones et comprendre comment ils sont créés et entraînés a été une expérience qui m'a vraiment plu. Pour moi qui est attiré autant par les mathématiques que par l'informatique, la découverte du domaine de l'informatique a été enrichissant et intéressant et m'a permis de me rapprocher de ma décision finale concernant le choix de la majeure à prendre en deuxième année de cycle ingénieur. Cet aspect du projet m'a montré que lier mathématiques et informatique n'était pas impossible et m'a permis d'envisager une possibilité de doctorat après l'EPITA.

Au début de ce projet, je pensais que le temps nous manquerait du fait de la complexité de ce dernier. J'ai donc rapidement commencé à me renseigner sur le réseau de neurones. En commençant à créer la preuve de concept pour pouvoir reconnaître la fonction XOR et à modéliser cette preuve de concept, je me suis très vite rendu compte que la façon de créer un réseau de neurones en langage C est très différente de celle dans un langage orienté objet comme le C# (langage où un exemple de réseau de neurones apprenant à jouer au jeu *Snake* nous avait été présenté en TP durant notre première année de cycle préparatoire).

Après la première soutenance, il nous a paru important de commencer à réaliser le réseau de neurones final car le temps allait très vite. En commençant la réalisation de ce dernier, j'ai pu me rendre compte que créer un réseau de neurones avec 4 neurones pour une couche au maximum n'était pas la même chose que de créer un réseau de neurones avec près de 900 neurones pour la couche d'entrée. Avec beaucoup de travail, ce réseau a finalement vu le jour et il a fallu ensuite l'entraîner. Le processus et les fonctions créés pour la preuve de concept étaient une bonne base à cet entraînement mais nous avons quand même dû les adapter. Lorsqu'au final, après beaucoup de tentatives de résolution, l'entraînement du réseau ne fonctionnait pas, j'ai été déçu. J'ai tenté, en solo et avec l'aide des membres de mon groupe de résoudre ce problème, mais sans y parvenir.

Finalement, au terme de ce projet, j'ai compris que réaliser un projet imposé est très différent de la réalisation d'un projet choisi. De plus, le passage du présentiel au distanciel a été, comme pour l'année dernière, un passage difficile car il est compliqué de séparer les périodes de cours, le temps en projet, et la vie personnelle, car l'environnement où nous nous trouvons reste le même peu importe si nous sommes en cours ou non. Finalement, malgré une fin de projet difficile, son rendu global est pour moi satisfaisant dans les autres parties de ce projet.

6.4 Jonathan Poelger

Dès la création des groupes, nous avons vite découpé le projet en grandes parties et commencé à se renseigner sur le sujet. J'avais personnellement déjà une bonne idée de ce qu'est un réseau de neurones car je m'intéresse depuis un certain temps à la thématique des IA. Cependant c'est dans les phases d'entraînement que j'ai pu découvrir beaucoup de choses. J'étais déjà familier avec les entraînements sous forme de sélections naturelles et de mutations et j'ai donc pu découvrir les algorithmes de descente de gradient, beaucoup plus efficaces pour ce genre de problèmes. Cela participe un peu à la démystification des IA, qui au final ressemblent davantage aux mathématiques qu'à de l'intelligence à proprement parler.

Après la phase de recherche, est venue la phase de développement de la preuve de concept, le réseau XOR. Mettre en pratique les nombreuses notions nouvelles dont nous disposions (rétro-propagation et descente de gradient notamment), à fortiori dans un langage de programmation que nous étions en train de découvrir, a été un challenge difficile à relever mais nous a surtout beaucoup appris et a permis d'ancrer de manière efficace toutes ces nouvelles connaissances. Grâce à une très bonne cohésion d'équipe, la première soutenance est arrivée sans poser de problèmes majeurs. Nous étions, je pense, à peu près dans les temps pour les différentes parties du projet à ce moment là.

Suite à cela, nous savions qu'il n'y aurait pas beaucoup de temps jusqu'à la soutenance finale et avons donc rapidement commencé à modifier notre réseau pour qu'il devienne un réseau de reconnaissance de caractères. Les principaux changements sont la présence d'une fonction d'activation dans l'exécution et les dérivées partielles, ainsi que la création d'une vraie collection d'entraînement (le XOR a en tout et pour tout 4 motifs possibles alors que la reconnaissance de caractères en aurait un nombre trop grand pour entraîner le réseau sur une liste exhaustive). Créer une collection d'entraînement complète s'avérant long et fastidieux, j'ai donc décidé d'automatiser cette étape en les créant de façon procédurale. Cela n'a pas été particulièrement complexe car la segmentation était déjà opérationnelle, ce qui a permis de l'utiliser pour créer les matrices d'entraînement de chaque caractère à apprendre.

Au niveau du réseau en lui-même, nous sommes rapidement arrivés à un résultat où la fonction de coût commençait à réduire, mais nous nous sommes à ce moment heurtés au plus gros problème que nous avons pu rencontrer dans ce projet : la fonction de coût semblait s'arrêter à un minimum local bien trop haut pour être efficace. Après avoir revu et repensé le code un certain nombre de fois, rien y faisait et nous sommes arrivés à court de temps avant d'avoir pu trouver ce qui faisait défaut à notre réseau. Malgré cela, nous avons pu mettre en commun les différentes parties du projet, et finalement, faire en sorte que ce projet s'unifie.

Au final, ce projet m'aura beaucoup apporté au niveau de la compréhension du C et des réseaux de neurones. Même si nous avons un peu manqué de temps pour en venir réellement à bout, cela a renforcé mon intérêt pour les IA.

7 Conclusion

Pour conclure ce rapport, nous pouvons dire que le projet de S3 nous a beaucoup apporté tant sur le plan technique que sur le plan humain. Nous avons pu découvrir et approfondir des sujets qu'il aurait été difficile d'aborder autrement. Même si nous n'avons pas pu tout à fait terminer ce qui était demandé, nous avons énormément appris sur des sujets tels que les réseaux de neurones, la retouche complexe d'images ou la création d'applications par exemple.

La cohésion d'équipe a été excellente durant ce projet, avec beaucoup d'entraide, d'efficacité et de communication entre les membres du groupe, et donc de ce fait une idée précise des avancements de chaque tâche à tout moment du projet. En effet, nous nous sommes réunis de façon très régulière pour travailler ensemble en s'épaulant.

Nous avons aussi pour la première fois pu réaliser un projet de relativement grande envergure avec un cahier des charges et des dates butoirs imposés. Cette situation est proche de ce que nous pourrons réaliser dans le futur et constitue donc un bon entraînement au milieu professionnel.

