Projet OCR

Rapport de soutenance

Wizard Neurons

Antoine Gonzalez
Louis Le-Gatt
Cédric Parpet
Jérémy Salfati

Dossier Projet Informatique
Info-Spé EPITA
Octobre 2018

Table des matières

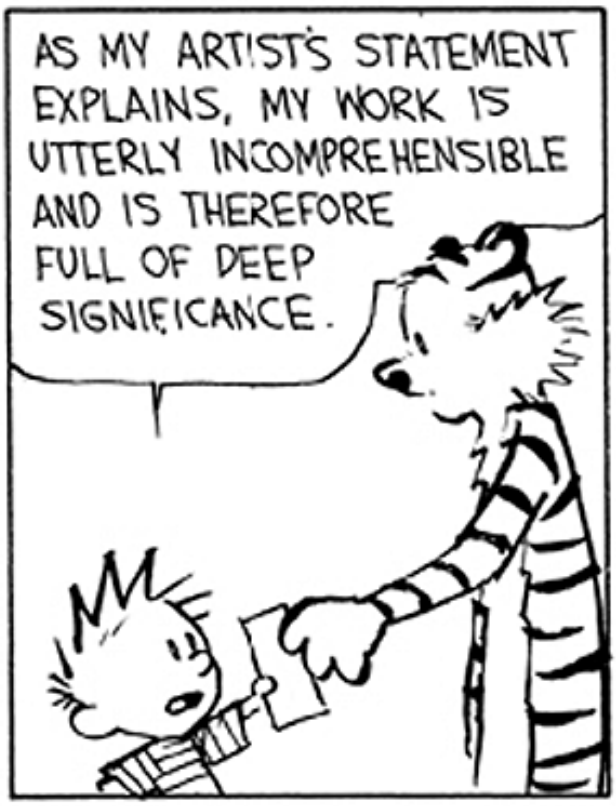
1. Introduction

Cette première moitié de projet aura été assez difficile, mais très intéressante. N'ayant eu que peu de cours de C, il nous aura fallu apprendre par nous même tout ce dont nous avions besoin. De plus, ce projet nous laisse bien moins de libertés que celui du semestre précédent, plus de limites nous sont imposées, tant sur le projet lui-même que sur la manière de l'implémenter. Ce qui nous force à chercher des solutions parfois moins évidentes.

Cependant, le groupe est motivé : un OCR est très intéressant à mettre en place et nous permet de nous essayer à beaucoup de concepts différents de la programmation : l'interface utilisateur avec GTK, le traitement d'image avec la SDL, et enfin l'intelligence artificielle (buzzword du moment) avec les réseaux de neurones. Et surtout, le langage C qui nous a été imposé est un classique de la programmation, et apprendre à l'utiliser nous permettra de mieux saisir certains concepts de la programmation avec des langages plus haut-niveaux, et de comprendre par exemple la gestion de la mémoire.

C'est donc avec soif de connaissances que nous nous sommes mis au travail, en quête du meilleur OCR réalisable. Durant ces premières semaines, chacun a pu expérimenter avec sa partie. Nous possédons désormais les connaissances et outils nécéssaires à la bonne réalisation du projet.

Dans ce rapport, nous allons présenter notre vision du projet ainsi que les avancées que nous avons réalisées, les difficultés rencontrées, et enfin nos plans pour la seconde soutenance.



Calvin and Hobbes, Bill Watterson

2. Répartition des tâches

Afin de rester organisés tout au long de ce projet, nous nous sommes réparties les tâches de la manière suivante.

	Antoine	**Louis**	**Cédric**	**Jérémy**
Chargement de l'image				X
Traitement de l'image		X		
Découpage de l'image			X	
Réseau Neuronal	X			
Sauvegarde des résultats				X
Interface Utilisateur				X

En somme, Antoine s'occupera de tout ce qui touchera de près ou de loin au réseau de neurones. Louis se chargera du traitement primaire des images (black&white ou grayscale, création d'une toolbox pour la manipulation des pixels...). Cédric devra s'occuper du découpage de l'image en lignes puis caractères, et de la reconstruction du texte reconnu par le réseau de neurones. Enfin, Jérémy sera en charge de l'interface utilisateur, afin de pouvoir utiliser tous nos systèmes de manière intuitive.

Bien entendu, chaque membre reste disponible pour aider les autres en fonction des besoins et du temps disponible. Mais permettre à chacun de se "spécialiser" dans une section permet aussi une avancée plus rapide du projet : il ne sera pas utile à tout le monde de tout savoir sur toute les parties.

3. Diagramme Fonctionnel

Afin que chaque membre comprenne parfaitement ce qui est attendu de sa partie et comment elle devra interagir avec celle des autres membres, nous avons réalisé un diagramme fonctionnel de l'OCR. On peut y voir le déroulement de son éxecution, du moment où l'utilisateur charge une image, au moment où le texte détecté est affiché et enrgistré *(voir figure ci-dessous)*.

Comme on peut le voir, chaque partie interagit de manière simple avec le reste, et possède un but bien défini. Chaque morceau de l'OCR n'appelle qu'une unique autre partie, ce qui limite les erreurs potentielles d'appels croisés entre les multiples fonctions. Chaque membre du groupe n'a donc qu'à penser à une seule autre partie. Il lui suffit de spécifier quel type de valeur il attend en retour, et quel type de valeur il est capable d'envoyer. La partie suivante n'a qu'à s'adapter à ces règles, et faire de même.

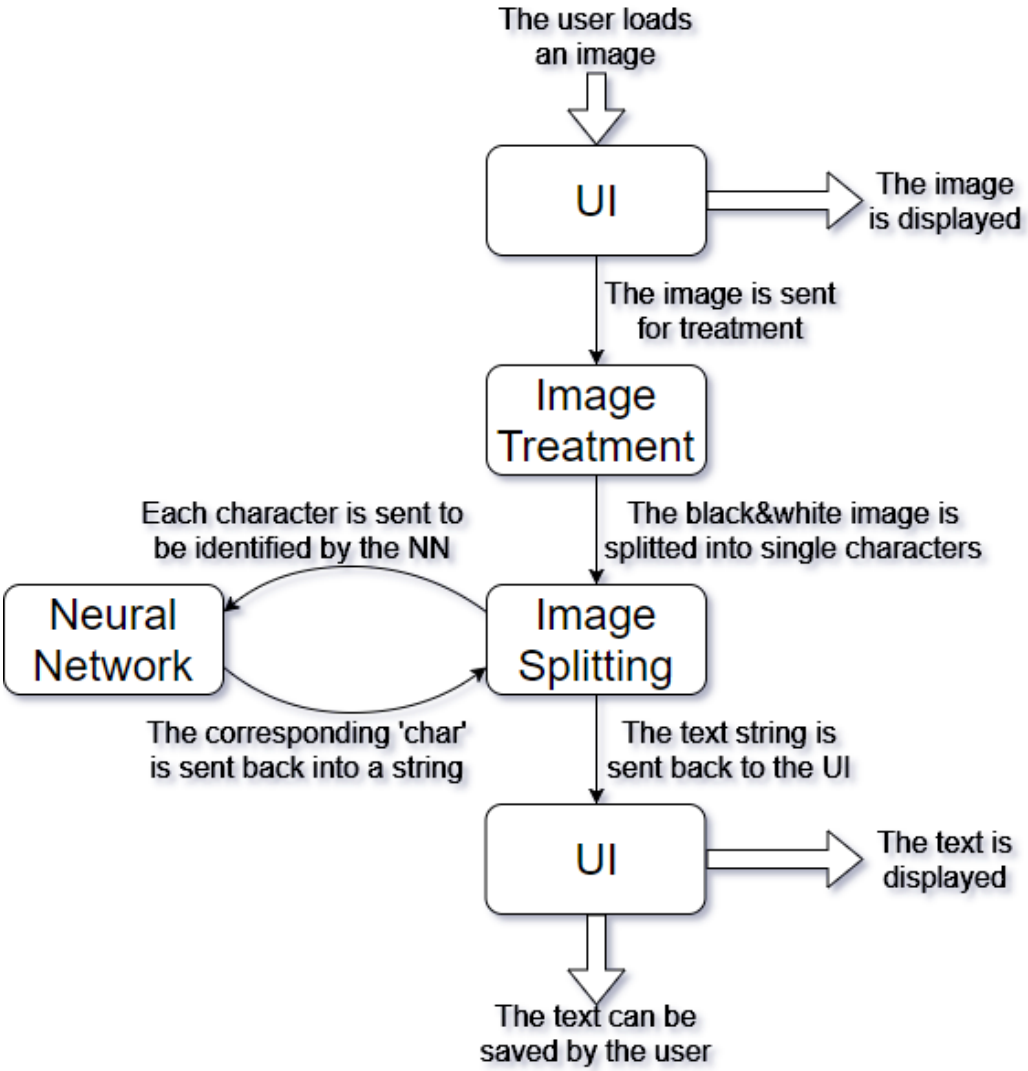


FIGURE 3.1 – Functional diagram

4. Avancement

4.1 Traitement de l'image (Louis)

J'ai eu à ma charge la partie sur le traitement de l'image. J'ai ainsi décidé de commencer par travailler sur le chargement de l'image et sa sauvegarde. Pour cela je me suis renseigné sur comment traiter une image en C, et quels outils permettent de faciliter le travail sur les images en C. J'ai très vite compris que j'allais devoir travailler avec la bibliothèque SDL, qui permet le traitement des images de manière simple.

Nous avons ensuite décidé des différentes fonctions utiles pour les différents membres du groupe. J'ai enfin développé un programme permettant de tester mes fonctions en affichant l'image puis en l'actualisant après application des différentes fonctions. Tout ceci toujours grâce à la bibliothèque SDL qui permet d'afficher une image sur une fenêtre en quelques lignes seulement.

Ainsi en accord avec les membres du groupe et le livret, j'ai décidé de réaliser ces fonctions :

- Une fonction qui supprime les couleurs d'une image afin d'obtenir une image en noir et blanc. Cette fonction utilise 2 autres fonctions vue en travaux pratiques : getpixel et putpixel qui permettent pour la première de récupérer la valeur d'un pixel donné et pour l'autre de placer un pixel sur une surface à une coordonnée x,y choisie. J'ai ainsi décidé de placer ces fonctions dans un fichier à part, hors de celui regroupant les fonctions de traitement de l'image, car les opérations sur les pixels peuvent s'avérer utiles pour les autres membres du groupe. Ces deux fonctions m'ont permis de récupérer la valeur du pixel et de la changer en noir ou en blanc en fonction de la couleur du pixel. Cependant après quelques discussions avec les autres membres du groupe nous avons décidé que l'image en noir et blanc n'était peut-être pas la meilleure solution pour le découpage et le réseau de neurones. Nous avons cherché une autre manière de supprimer les couleurs de l'image. La solution qui semble la mieux adapté pour les traitements à réaliser fut d'obtenir une image en teinte de gris (dite grayscale).

FIGURE 4.1 – Original image

- La deuxième fonction que j'ai réalisé fut donc une fonction permettant encore une fois supprimer les couleurs, mais de cette fois-ci les remplacer par leur teinte de gris associée. Il a donc fallu faire plusieurs recherches sur la teinte de gris la plus adapté pour notre travail. J'ai donc opté pour 3 coefficients appliqués aux valeurs du rouge, du bleu et du vert de chacun des pixels. Puis il suffit d'additionner ces trois valeurs (après multiplication par leur coefficient respectif) afin d'obtenir notre valeur de gris propre au pixel actuel.
- J'ai ensuite réalisé une fonction de sauvegarde d'image permettant de sauvegarder les modifications de l'image. Puis une fonction de copie d'une image. Celle-ci sera très utile afin de ne pas réaliser tous ces traitements sur l'image que l'utilisateur va nous fournir. En effet nous souhaitons restituer à l'utilisateur son image sans aucun changement de nom ou de structure.

FIGURE 4.2 – Grayscale

- J'ai finalement décidé d'ajouter une fonctionnalité supplémentaire au traitement de l'image. En effet, en testant mon code avec plusieurs images différentes je me suis très vite rendu compte de plusieurs défauts pouvant apparaitre sur l'image. J'ai fait quelques recherches qui m'ont conduit conclure qu'il s'agissait de "bruit". J'ai alors décidé d'essayer de réduire au maximum ce bruit sur notre image afin de faciliter les étapes suivantes de notre OCR. Après de nombreuse recherche à travers des manières plus ou moins complexe de corriger ce bruit j'ai décidé de le réduire en utilisant un filtre médian. Ce filtre médian fonctionne de manière très simple. Il parcourt l'image pixel par pixel et observe pour chacun des pixels, ses pixels voisins. Il réalise ensuite une liste de ces différents pixels en y intégrant leur valeur numérique correspondant à leur couleur. Finalement cet algorithme trie cette liste en ordre croissant. La valeur médiane de cette liste devient alors la nouvelle valeur du pixel. Ainsi, chaque pixel est une moyenne de ses pixels voisins ce qui permet de supprimer le bruit de l'image.

FIGURE 4.3 – Grayscale without noise

Ma partie sur le traitement de l'image est ainsi à ce jour constituée de 5 fonctions principales réalisant des taches indispensables pour la reconnaissance de caractère.

4.2 Découpage de l'image (Cédric)

La première version du découpage de caractère est une version itérative. Elle consiste à parcourir les lignes de l'image jusqu'à trouver un pixel noir indiquant un morceau de caractère (cela nécessite donc que l'image soit traitée au préalable afin d'éliminer le bruit ou bien de choisir des images déjà adaptées). On garde l'index de la ligne en mémoire, puis on continue de parcourir les lignes de l'image jusqu'à ce qu'on tombe sur une ligne sans pixels noirs. On prend l'indice de la ligne précédente qui contenait des pixels en mémoire, puis on parcourt les colonnes entre l'intervalle formé par les deux indices mémorisés. La méthode est la même : lorsque l'on trouve une colonne contenant un pixel noir, on garde l'index de cette colonne en mémoire puis on parcourt les autres jusqu'à trouver une colonne sans pixels noirs, on prend alors l'index de la colonne précédente et on obtient ainsi 4 bornes qui nous permettent de déterminer un rectangle dans lequel se situe un caractère. On continue ensuite de la même façon à déterminer les caractères en parcourant les autres colonnes. Une fois cela fait on reprend le parcours des lignes en répétant le même processus.

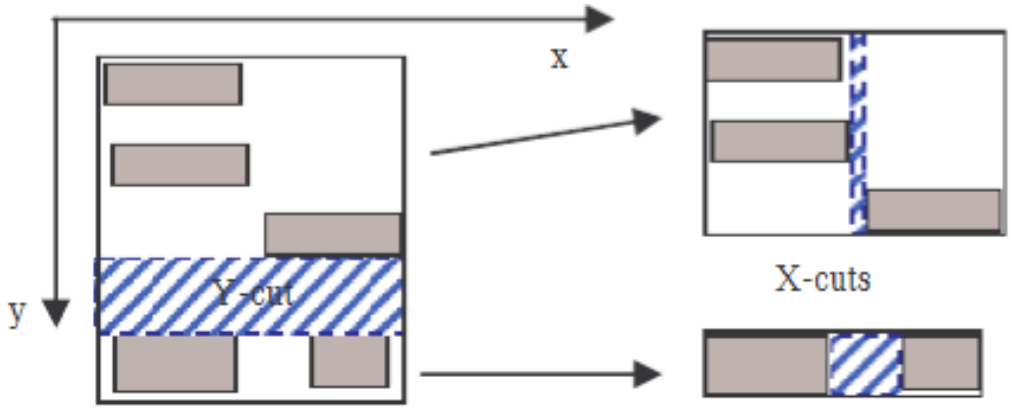


FIGURE 4.4 – Image segmentation

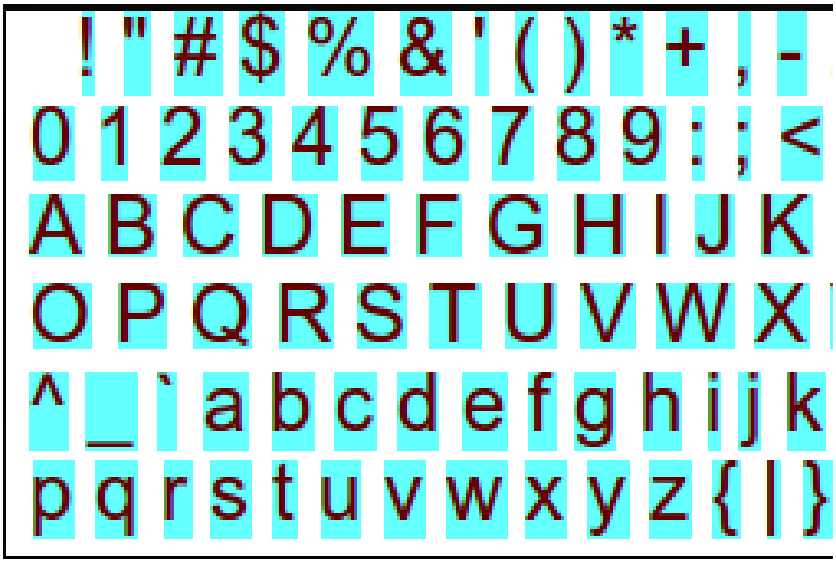


FIGURE 4.5 – Example 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut facilisis, urna vel scelerisque venenatis, quam purus vulputate orci, eget pulvinar diam quam finibus ante. Sed erat mi, lacinia nec accumsan eu, elementum vel sapien. Pellentesque auctor leo purus, at porttitor dolor viverra eget. Duis enim eros, tincidunt in sapien vel, aliquam porttitor justo. Nam laoreet augue sit amet faucibus pellentesque. Vestibulum suscipit magna eget nibh consequat congue. Donec pharetra sollicitudin arcu sit amet varius. Sed vehicula nisi quis orci mollis, id laoreet sem viverra. Phasellus sagittis orci diam, vitae sagittis erat maximus fermentum. Nam ultrices justo ut massa molestie molestie. Integer venenatis tristique erat vel consectetur. Aliquam placerat varius neque vitae luctus. Mauris euismod efficitur justo non congue. Nullam tincidunt scelerisque ex at tempus. Nunc cursus feugiat pellentesque.

FIGURE 4.6 – Example 2

L'objectif avec cette méthode était de pouvoir créer une liste d'image où chaque image correspond à un caractère. On aurait ensuite traité ces images pour qu'elles aient une taille de 28 pixels par 28 avant de les transformer en matrices pour les donner au réseau de neurones. Un des problèmes qui se montraient était la création d'une liste d'image que l'on n'arrivait pas à implémenter avec SDL. Un autre problème était qu'il n'était pas possible lors de la reconstruction du texte de replacer les espaces et les retours à la ligne.

Des recherches plus poussées nous ont fait savoir que l'utilisation d'une version récursive de la segmentation permettrait de résoudre ce problème notamment grâce au stockage des caractères dans un arbre binaire (qui est une structure récursive). L'algorithme utilisé est le Recursive XY Cut . Il consiste a diviser simultanément le texte horizontalement et verticalement en trouvant les plus grand espaces vides (ici sans caractères) puis de continuer sur les blocs ainsi découper jusqu'à obtenir des blocs sans espaces : les caractères. Cependant l'algorithme seul ne permet pas de réorganiser le texte correctement, nous le modifierons un peu de manière à couper horizontalement jusqu'à ce que cela ne soit plus possible, puis verticalement pour obtenir les caractères dans le bon ordre.

Malheureusement, nous nous sommes rendus compte trop tard de l'utilité de la récursion, et n'avons pas eu le temps de finir la segmentation récursive à temps. Seules des fonctions intermédiaires utiles pour le découpage ont été codées jusque là. La version compilable rendue pour cette première soutenance est donc la version récursive.

4.3 Réseau neuronal (Antoine)

Pour le réseau neuronal capable d'apprendre la porte logique XOR, je suis partis d'un design classique : 2 entrées, 1 couche cachée contenant 2 neurones, 1 sortie, et 1 biais entre chaque couche. Afin que les données soient organisées, j'ai opté pour des tableaux pour stocker le contenu des noeuds et les poids (les poids entre l'entrée et le *hidden layer* sont nécéssairement stockés dans une matrice).

Cet algorithme n'a en soit pas grand chose de compliqué. La forward-propagation (résolution de test) revient à additionner le produit des poids par leur entrée, et à stocker la valeur obtenue dans le noeud de sortie auquel les poids sont reliés avant d'appeler la fonction d'activation. La fonction que j'ai choisi est la fonction sigmoïde, suffisante pour un réseau de neurones de cette taille..

Pour ce qui est de la backward-propagation, il m'a été difficile de réellement saisir le fonctionnement des opérations et surtout comment décomposer le gradient d'erreur, mais je suis finalement parvenu à la faire fonctionner, et il n'y a rien de plus satisfaisant que de voir son réseau neuronal apprendre tout seul à résoudre un problème. La constante d'apprentissage ETA a été choisie arbitrairement : 0.1.

Tout cela me donne un réseau qui prend cette forme :

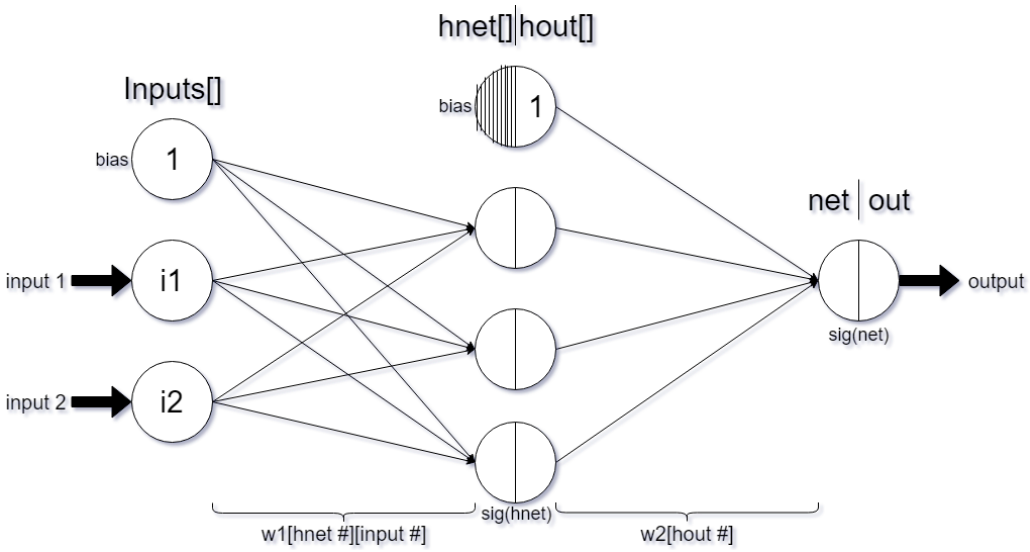


FIGURE 4.7 – XOR neural network

Mais il ne s'agit ici que du XOR, un *proof of concept*, le *Hello World* des réseaux de neurones. Pour le véritable réseau de notre OCR, il faudra traiter des caractères sous forme d'images de dimensions 28x28 pixels, soit 784 entrées. Il me faudra expérimenter avec le *hidden layer* afin de trouver combien de neurones seront nécéssaires, voire s'il est utile d'ajouter une couche supplémentaire. Enfin, pour la sortie, nous allons commencer par nous contenter de la reconnaissance des chiffres et des lettres (majuscules et minuscules), soit 62 caractères, donc 62 sorties.

Afin d'entraîner cette future intelligence artificielle, je compte pour le moment utiliser les ressources fournies par NIST dans son *NIST Special Database 19*, disponible à l'adresse suivante : `https://catalog.data.gov/dataset/nist-handprinted-forms-and-characters-nist-special-database-19`.

Je pense qu'une des difficultés principales de la deuxième partie du projet sera d'apprendre à utiliser ces données (en extraire les images et les valeurs attendues), mais cette archive est très complète et contient suffisamment d'exemples pour entraîner le réseau de neurones.

Un élément restant à implémenter au plus vite est l'enregistrement des poids, afin de ne pas avoir à réentraîner le réseau neuronal à chaque fois. Je n'ai malheureusement pas pu m'en occuper à temps pour la première soutenance, il faudra donc régler ce détail au plus vite.

4.4 Interface utilisateur (Jérémy)

Tout d'abord, je me suis intéressé au choix des bibliothèques qui constitueront notre interface. Nous avons choisi GTK+, développé à l'origine en C, pour sa portabilité. Afin d'interagir avec notre application, il nous faut créer une fenêtre. Le widget permettant d'afficher une fenêtre se nomme "GtkWindow". Les objets graphiques utilisent la notion d'héritage afin de récupérer des propriétés et des méthodes de widgets parents pour modifier leur propre comportement sans avoir à tout redéfinir.

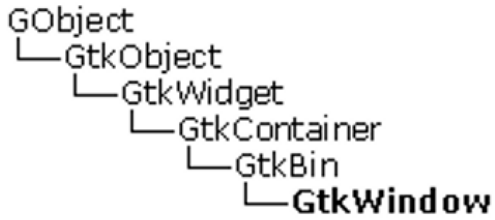


FIGURE 4.8 – Windows inheritance with GTK

Ensuite, nous devions utiliser plusieurs éléments essentiels d'une interface graphique : les box et les boutons. GTK+ a certains avantages comme certains inconvénients. Nous ne pouvons pas intégrer plusieurs widgets dans une fenêtre car un "GtkContainer" ne peut en contenir qu'un seul. Pour résoudre ce problème, je me suis penché sur l'utilisation de deux catégories de "GtkBox" :

- Les **GtkHBox** qui permettent de disposer les widgets horizontalement
- Les **GtkVBox** qui permettent de les disposer verticalement

De plus, les boutons permettent à l'utilisateur d'effectuer une action grâce à un simple clic de souris.

FIGURE 4.9 – Boxes and buttons inheritance with GTK

Pour cela, j'ai étudié le fonctionnement des signaux afin que ceux-ci ne se terminent qu'au moment où l'utilisateur le demande. Lorsque l'utilisateur interagit avec l'application, le widget concerné émet un signal. Il est associé à une ou plusieurs significations et permet donc de programmer toutes les actions possibles. J'ai donc créé plusieurs fonctions "callback" qui connectent le signal au bouton concerné. Nous pouvons effectuer plusieurs actions (les trois premières s'activent avec le signal "clicked") :

- Ouvrir une image
- Ouvrir une zone de texte
- Fermer cette zone
- Quitter l'application (avec le signal "destroy")

J'ai également inclus une barre de défilement pour que le texte converti ne soit pas bloqué par une taille fixe de la zone de texte.

Voici une image montrant l'avancement actuel de l'interface graphique.

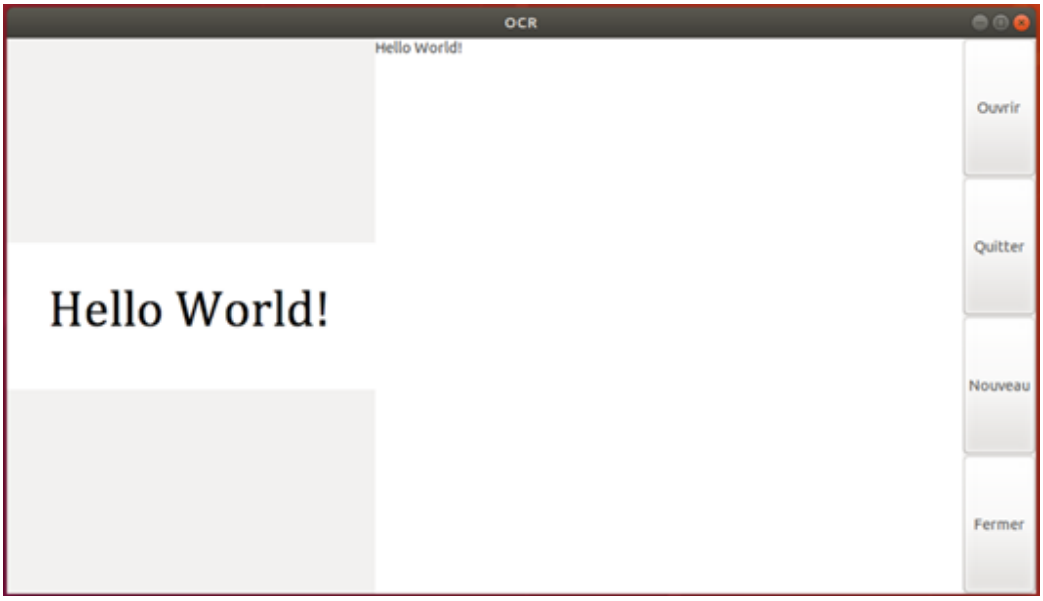


FIGURE 4.10 – Current user interface

La prochaine étape sera de créer le bouton "Convert". En effet, nous devrons relier l'interface graphique avec toutes les étapes nécessaires à la construction d'un texte (traitement, découpage et réseau neuronal). Sauvegarder ce dernier sera également possible pour la soutenance finale.

5. Avances, retards, difficultés rencontrées

Traitement de l'image

Le début du projet fut très difficile pour différentes raisons. Tout d'abord il a fallu apprendre un nouveau langage de programmation : le C. Il a donc fallu se renseigner sur ses spécificités, ses difficultés et ses bibliothèques. De plus ce langage est venu avec énormément de nouveautés pour moi. En effet je n'avais que très peu utilisé les lignes de commande auparavant, et encore moins utilisé Vim. J'ai donc appris à utiliser ces nouveaux outils tout en me renseignant sur le C et en commençant à travailler sur ce projet. L'éditeur n'a pas été simple à prendre en main car il demande l'ajout de plusieurs fonctionnalités devenir rendre plus agréable à l'utilisation. De plus, ses raccourcis ne sont pas forcément intuitifs et demandent quelques semaines de pratique pour les maitriser. J'ai ensuite lu plusieurs documentations concernant SDL, qui allait devenir très importante pour ma partie.

Après les premiers travaux pratiques j'ai vite compris qu'il fallait organiser son code en plusieurs fichiers afin d'obtenir un projet lisible et de ne pas perdre de temps lorsque l'on cherche des éléments. J'ai donc pris le temps d'organiser mes fonctions en différents fichiers pour améliorer sa lisibilité.

Une des parties qui m'a posé le plus de problème a été la compréhension des pointeurs. Les pointeurs sont des outils nouveaux pour moi, je ne savais absolument pas ce dont il s'agissait avant de commencer le projet.

Découpage de l'image

Une première difficulté pour moi a été de me familiariser avec le langage C et la librairie SDL. Chercher à créer une liste d'images pour compléter ma segmentation itérative m'a pris du temps et s'est soldé par un échec. M'être rendu compte trop tard qu'une version récursive était probablement une meilleure solution a fait que je me retrouve en retard pour cette première soutenance.

Le point positif est qu'il vaut mieux s'en être rendu compte maintenant et avoir commencé à coder la version récursive plutôt que de s'en rendre compte plus tard et avoir perdu encore plus de temps.

Réseau neuronal

J'ai eu beaucoup de difficultés à comprendre pourquoi le réseau de neurones, qui la majorité du temps fonctionne sans problème, n'apprenait parfois pas la moitié de la table de vérité du XOR. Il s'agissait en fait d'un problème de minimum local. N'ayant pas réussi à implémenter de momentum, je me suis contenté de mitiger la probabilité de tomber dans un minimum local en ajoutant un neurone supplémentaire dans la couche cachée.

Pour le réseau de neurone de l'OCR, j'utiliserai simplement un seed prédéfinit pour le random : une valeur d'initialisation de l'aléatoire pour laquelle l'apprentissage fonctionne, car il n'est pas nécéssaire d'avoir un aléatoire différent à chaque apprentissage du réseau, du moment que celui-ci fonctionne.

A cause de ce problème sur lequel j'ai passé trop de temps, je n'ai pas pu commencer à m'occuper de l'enregistrement des poids du réseau neuronal. Il faudra donc mettre cela en place pour la soutenance finale.

Autrement, l'autre difficulté de cette partie a été d'apprendre à utiliser les pointeurs et *m*alloc pour utiliser les tableaux en C.

Interface utilisateur

Concernant l'interface graphique, j'ai eu de nombreuses difficultés que j'ai réussi à surmonter.

En effet, je devais intégrer correctement deux box (l'une étant la zone d'image, l'autre, l'emplacement des boutons) dans un conteneur (appelé "main_box") qui lui-même se trouve dans une fenêtre. Le tout rendait finalement notre application intuitive.

De plus, je pensais qu'il fallait contenir un widget image dans une zone de texte. Je me suis rendu compte que ce n'était pas la bonne solution après quelques jours, me tournant vers une option plus pragmatique : créer une nouvelle box.

Ma principale épreuve était de rapidement apprendre à coder en C, car j'utilise dans mon interface des listes chainées et la notion d'héritage. Au final, je ne suis ni en retard, ni en avance par rapport à la répartition des taches, sauf si on considère un début d'interface pour la première soutenance comme une avance.

6. À venir

— **Traitement :** corriger ou adapter le traitement en fonction des besoins après des tests sur le réseau de neurones réel.

— **Découpage :** reconstruction du texte. Normalement, l'implémentation récursive permettra une reconstruction simple.

— **Réseau neuronal :** véritable réseau de neurones de l'OCR, sauvegarde des poids.

— **Interface :** lier l'interface aux différentes fonctions.

7. Expériences personnelles

— **Louis :** Ce projet me permet personnellement de me forcer à toujours avoir quelque chose à améliorer. Il m'a aussi permis de me familiariser très rapidement avec le C et le terminal. De plus, l'entente au sein du groupe est très agréable et me permet de travailler efficacement.

— **Cédric :** Je trouve ce projet intéressant et j'ai envie de mener à bien ma partie même si cela s'annonce beaucoup plus compliqué que ce je l'imaginais. Ce projet m'a néanmoins forcé à apprendre à utiliser Linux, dont j'apprécie désormais beaucoup l'utilisation si bien que j'en ai fait mon OS principal, et le langage C.

— **Antoine :** Cette première partie du projet a été très intéressante pour moi. Cela m'a permis d'apprendre ce qu'est un réseau neuronal, comment cela fonctionne, et comment en créer un. L'intelligence artificielle n'est plus un concept totalement opaque pour moi, et je comprend désormais ce dont il s'agit même s'il me manque probablement énormément de notions. Après avoir traité de simples 0 et 1 pour une porte logique, j'ai hâte de créer le réseau qui sera capable de traiter des images de caractères, bien que cela sera difficile à mettre en place.

— **Jeremy :** Ce projet m'a permis pour l'instant, d'apprendre à coder en C plus rapidement qu'en travaux pratiques. J'ai eu l'occasion de découvrir ce qu'est GTK et la manière dont nous pouvons créer une interface graphique. De plus, il s'agit ici de mon deuxième projet informatique en groupe, ce qui est pour moi très intéressant et très bénéfique afin de développer mon expérience d'ingénieur. J'ai eu quelques difficultés, certes, mais celles-ci me permettent d'être fier lorsque je parviens à les résoudre. J'ai hâte de pouvoir construire une véritable interface graphique et à contribuer davantage à ce projet.

8. Conclusion

La première soutenance de projet arrive, et le groupe est satisfait de son travail. Notre réseau de neurones XOR fonctionne, nous disposons désormais d'outils de traitement d'image qui faciliteront notre travail à venir, le découpage de l'image en lignes puis caractères fonctionne parfaitement, et l'interface utilisateur avance bien. Toutefois, il reste beaucoup de travail, et le plus difficile reste à venir. Le réseau de neurones attendu est beaucoup plus complexe que celui chargé d'apprendre la porte logique XOR, et il faut également l'entrainer. L'interface doit être terminée et reliée aux fonctions de l'OCR au plus vite. Et il faudra probablement corriger l'implémentation du traitement d'images et du découpage en fonction des résultats de futurs tests "en conditions réelles".

Malgré les difficultés rencontrées, le groupe est motivé, et très confiant. Bien que certaines parties vont devenir plus chargées, d'autres vont s'alléger, permettant ainsi aux membres d'aider les autres plus facilement. L'OCR sera donc réalisé dans son entièreté en temps et en heure.