Rapport de suivi du projet OCR

Groupe NVAM*

Novembre 2024

*Mael Minard (chef de projet), Victor Flament, Adrien Fonck, Jaisy Brahimi

Table des matières

1 Introduction

Ce rapport a pour rôle de suivre l'avancement du projet OCR de l'équipe NVAM pour la première soutenance. Ce rapport sera divisé en deux parties, la première parlera des attendus pour la première soutenance et la deuxième aura pour objectifs de montrer l'avance que nous avons pris. Dans ce projet, nous nous intéressons au logiciel de type OCR (Optical Character Recognition) qui permettra la résolution d'une grille de mot caché. La reconnaissance optique de caractères (OCR) est un processus permettant de transmuter une image de texte en format de texte lisible par une machine, elle est importante pour les entreprises qui impliquent la réception d'informations provenant de médias imprimés. En effet l'OCR est imbriqué dans plusieurs corps de métier (juridique, commerciale, médicale, etc..) car la numérisation de toutes cette documentions prend du temps et de l'espace de stockage. De plus la numérisation donne des images texte qui ne peuvent pas être traité par les logiciels connu de traitement de texte pareillement à un document texte. L'OCR permet de résoudre ce défaut en convertissant ces images textes en données textuelles. Nous utiliserons ainsi un réseau de neurone, une méthode venant de l'intelligence artificielle permettant aux ordinateurs d'apprendre à traiter des données fortement inspirées du cerveau humain. Dans notre cas nous allons tenter de réaliser le logiciel OCR pour permettre de résoudre une grille de mots cachés. Pour cette première ébauche nous étudierons le chargement d'une image et sa suppression de couleurs, la rotation manuelle d'une image, la détection d'une position, le découpage d'une image, un algorithme de résolution et un réseau de neurone du XNOR. Enfin pour minimiser le temps d'apprentissage de notre machine nous avons débuter l'implémentation de la descente de gradient.

2 Présentations

2.1 Le groupe NVAM

NVAM est un groupe de projet de deuxième année formé par quatre étudiants en septembre 2024. Le groupe est composé de Jaisy Brahimi, Victor Flament, Adrien Fonck et Mael Minard.

La mission de notre équipe est de réaliser le projet OCR, c'est a dire créer un programme capable de résoudre une grille de mots mêlés a partir d'une image.

2.2 Les membres

2.2.1 Jaisy Brahimi

Jaisy Brahimi est un étudiant en deuxième année à l'école d'ingénieur informatique EPITA, à Lyon Part-Dieu. Ayant une appétence dans le domaine de la finance il vient d'une licence mathématique-informatique et se consacre maintenant à plein temps à une école spécialisé en informatique. Au sein du groupe NVAM, Jaïsy a donc pour rôle l'assistance à la réalisation du réseau de neurones qui sollicite l'étude de concepts

abstraits, notamment dans la descente de gradient qui a pour but la minimisation. En effet ce lien avec la finance n'est pas anodin car toute entreprise à pour but de minimiser ses coûts. Il devra donc assister à l'architecture du réseau de neurones, à son entraînement et à son exploitation. De plus il assistera aussi dans l'interface graphique.

2.2.2 Victor Flament

Victor Flament est un brillant élève au sein du campus de Lyon de EPITA. Tombant dans la passion des systèmes GNU/Linux durant sa seconde il se décida a faire des études de maths et d'informatique. Étant passionne de systèmes d'exploitations et de développement systèmes ce projet n'est pas des plus passionnants pour lui. Cependant l'algorithmique fait parti des domaines qu'il l'intéresse. C'est pour cela qu'il s'occupe des taches de solver et détection. Mais aussi du traitement d'images.

2.2.3 Adrien Fonck

Adrien a vécu à Gap, dans les Hautes-Alpes, mais il a passé son baccalauréat à l'école internationale de la région PACA, à Manosque.C'est aussi dans cet établissement qu'il a développé ses compétences informatiques, notamment au travers des spécialités Mathématique et Numérique Science Informatiques. Dans sa scolarité, Adrien a pu participer à différents concours d'informatique, particulièrement Prologin au niveau national (ou il finira sa course en demi-finale). Dans ce projet, il sera surtout présent pour la conception du réseau de neurones et aidera pour les détections des caractères.

2.2.4 Mael Minard

Mael Minard est un étudiant en deuxième année à l'école d'ingénieur informatique EPITA, à Lyon.

Finissant son cursus baccalauréat général au Lycée Rosa Parks en 2023, il se dirige vers des études d'informatique, sujet qui le passionne depuis son plus jeune age.

Au sein de ce groupe, Mael fait office de chef de projet, il s'occupe donc de coordonner l'équipe, ainsi que de la représenter durant leurs présentations.

Étant donné sa passion pour l'informatique, et sa soif d'apprentissage, Mael se dirigera au sein de ce projet vers la réalisation du réseau de neurone, une tache demandant la lecture d'un grand volume de documentation abstraite. Il sera donc chargé de réaliser l'architecture du réseau de neurone, ainsi que les différentes fonctions permettant de l'entraîner et l'exploiter. Il sera également chargé d'entraîner le réseau de neurone sur sa machine personnelle, de sorte a pouvoir le sauvegarder pour le réutiliser dans le programme.

3 Éléments de première soutenance

3.1 Traitement d'images

3.1.1 Attendus

Le principe de cette partie est de rendre l'image la plus utilisable possible pour la détection des lettres et de leur reconnaissance. Les principales fonctionnalités attendues sont l'augmentation des contrastes, la réduction du bruit donc les grains et la rotation et aussi tout ce qui est niveau de gris puis noir et blanc.
La manière dont nous avons procéder es de créer divers filtres pour des images et de les appliquer pour augmenter la qualité d'image. Pour ce faire nous nous sommes principalement bases sur la bibliothèque SDL2 qui nous permet de travailler sur les pixels d'une image.

3.1.2 Niveau de gris / Noir et blanc

Pour faire le niveau de gris d'une image nous parcourons l'image pixel par pixel et pour chaque pixel nous calculons sa luminance, car le bleu n'est pas autant lumineux que le vert qui n'est pas autant lumineux que le rouge. Ensuite on attribue cette luminance pour la composante rouge, bleue et verte de chaque pixel. Un pixel ayant la même valeur pour ses trois composantes est un pixel gris, ce qui nous permet donc de faire un niveau de gris pour l'image.
Pour ce qui est du noir et blanc c'est un peu plus simple, on parcourt chaque pixel de l'image et pour chaque pixel on fait une somme de ses trois composantes, si cette somme est au dessus d'un certain seuil on le modifie comme un pixel blanc donc (255, 255, 255) et si c'est en dessous ou égal au seuil c'est un pixel noir (0, 0, 0) a la fin tout les pixels sont soit noirs soit blancs.

3.1.3 Négatif

Nous ne savons pas si ce filtre va nous être utile mais il nous semble qu'aussi utiliser le négatif d'une image peut être utile, pour ce faire nous parcourons l'image pixel par pixel et chaque composante var prendre la valeur 255 moins sa propre valeur, ainsi un pixel aura des composantes opposes a celles d'avant et on obtiendra le négatif de notre image.

3.1.4 Modifier le contraste

Augmenter le contraste est utile pour mieux utiliser l'image mais en plus augmenter le contraste permet de diminuer le grain d'une image de manière significative. Pour augmenter le contraste nous parcourons notre image pixel par pixel et pour chaque composante de notre pixel nous allons lui appliquer un calcul mathématique, soit c notre composante de notre pixel et n le facteur d'augmentation du contraste. Si c est inférieur ou égal a 255 nous lui attribuons la valeur de 255 divise par 2 multiplie par 2 fois c divise par 255 le tout a la puissance n, et si c est supérieur a 255 divise par 2 nous lui attribuons le négatif de la fonction avec le négatif de c. Ce qui donne ceci en théorie: si c ¡= 255 / 2 alors c = (255 / 2) * (2 * c / 255)$^n sinon c = 255 - (255/2) * (2 * (255 - c)/255)^n$.

3.1.5 Modifier la luminosité

Augmenter la luminosité peut être utile pour augmenter l'impact des lettres sur l'image, qu'elles soient plus 'entières' et donc plus facilement reconnaissables. Pour ce faire on va parcourir notre image pixel par

pixel et pour chaque pixel on va appliquer un calcul pour chacune de ses composantes, on va faire 255 fois la composante divise par 255 le tout puissance le facteur d'incrémentation. Ce qui donnerait avec c la composante et n le facteur d'augmentation: 255 * (c / 255)^n.

3.1.6 Blur de Gauss

De base l'algorithme du blur de gauss sert a flouter une image, mais dans notre cas on l'utilise a très petite dose, on floute juste un petit peu et sur une image âpres un grayscale ou un noir et blanc cela permet d'épaissir les lettres ou même de combler quelques trous si elle ne sont pas entières ou déformées.
Pour effectuer ce flou on va prendre un nombre de pixels autour du pixel qu'on veut flouter, pour chaque pixel on va lui effectuer un calcul pour prendre en compte ses composantes et sa distance par rapport au pixel que l'on veut flouter. En effet on cherche a donner au pixel une valeur qui serait une moyenne pondérée par la distance au pixel. On va donc générer une matrice qui sera la pondération pour chaque pixel environnant, grâce a elle on multipliera les valeurs des pixels par leur position dans la matrice, le pixel que l'on veut flouter étant celui au centre de la matrice. Voici la matrice K si on prend les 9 pixels autour.

$$K = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} = \begin{bmatrix}1 & 2 & 1\end{bmatrix}\frac{1}{4} * \begin{bmatrix}1\\ 2\\ 1\end{bmatrix}\frac{1}{4}$$

Après l'idée est juste de faire la somme des composantes en facteur de leur position et a la fin de faire la moyenne, par rapport a la somme des valeurs de la matrice.

3.1.7 Dying filter

Ce filtre est moins efficace que ce qu'on pourrait penser mais aide bien a aider a réduire le bruit d'une image. Son principe est simple, pour chaque pixel on regarde les pixels environnants et lui met la valeur du plus grand nombre de pixels environnants. Donc si un pixel noir est entoure de 5 pixels blanc et 3 pixels rouges il va devenir blanc.
Pour le faire on parcourt notre image pixel par pixel et on fait un histogramme des pixels, si tout les pixels sont différents il prend la valeur des moyennes des pixels sinon il prend la valeur du plus grand nombre de pixels.

3.1.8 Rotation

Avec SDL nous avons deux moyen d'exécuter la rotation, par texture ou par surface, nous avons implémente pour les deux. Pour la texture ce sera juste pour l'aspect visuel et pour la surface nous modifieront directement la va structure de l'image. Pour la texture nous avons directement utilise une fonction fournie par SDL et ainsi lui donner n'importe quel angle. Pour la surface nous avons trois fonctions: la rotation a 90 degrés adroite, a 90 degrés a gauche et a 180 degrés. Juste ce que l'on va faire, on fait une copie des pixels, et on parcourt notre image pixel par pixel et on remplace par le nouveau pixel dans la copie.

3.1.9 Prochaines étapes

Pour l'instant nous estimons que le traitement d'images est aboutit a 60%, en effet nous pouvons avoir de très bons résultats avec nos filtres et des images très bonnes pour l'utilisation mais nous utilisons nos filtres a la main. Nos prochaines étapes sont de trouver comment automatiser l'utilisation des filtres. Nous avons déjà des pistes comme calculer la luminosité et le contraste de notre image mais c'est pas encore fait. Ensuite il faudrait sans doute implémenter un filtre de Canny et utiliser les courbes de Hough pour réussir a redresser automatiquement une image courbée ou penchée.

3.2 Détection de la position

3.2.1 De la grille et de la liste de mots

Cette partie est très ardue et est la plus technique avec le réseau de neurone. Le nombre important d'exemples et de cas différents la rend vraiment complique a appréhender.
Pour réaliser cette détection nous avons utilise une méthode qui ne marche que partiellement, nous expliquerons par la suite le pourquoi. Notre idée est de s'intéresser a la concentration des lettres. En effet nous cherchons la concentration des lettres dans l'image. Pour ce faire nous allons découper notre image en sous blocs et pour chaque sous bloc on va calculer sa concentration en pixels noirs. Si cette concentration est au dessus d'un certain seuil alors on le considère comme étant utile.
Mais ensuite il faudrait pouvoir regrouper les blocs adjacents et ainsi créer des formes distincts a travers notre image représentant ses différents formes visuels. A priori il y en a deux, la grille et la liste de mots. Pour cela nous créons un tableau de ces formes. Des que nous trouvons un

bloc intéressant nous regardons si il est adjacent a une de ces formes, si c'est le cas nous l'y intégrons sinon on créer une nouvelle forme pour ce bloc.
Ensuite nous vérifions si des formes sont adjacentes, en effet des blocs peuvent être adjacents mais être dans deux formes distinctes mais donc ces deux formes en sont qu'une seule. Pour ce faire on parcourt notre liste de forme et pour chaque forme on regarde les formes qui viennent après dans la liste et on regarde si elles sont adjacentes.
Nous avons donc une liste des différentes formes de notre image, pour l'instant nous partons du principe que la plus grande forme est la grille et la plus petite la liste de mots.
Malheureusement notre méthode rencontre de sérieux problèmes, notamment si la grille est penchée impossible de reconnaître correctement sa forme. Ou encore si les lettres de la grille sont très espacées mais que la liste de mots est proche alors tout est considère comme un seul et même bloc. Nous n'avons pas encore réussi a régler ces problèmes.

3.2.2 Des lettres dans la grille

Pour détecter l'emplacement de chacune des lettres dans la grille de mots nous avons un algorithme assez simple mais plutôt efficace.Nous parcourons notre et grille ligne par ligne. Des qu'on a une ligne avec un pixel noir alors on continue de parcourir jusqu'à trouver une ligne sans pixel noir. Quand c'est le cas nous par courons l'espace entre cette ligne et la précédente ligne sans pixel noir colonne par colonne. Et nous refaisons un peu la même chose avec les colonne, on parcourt les colonnes, des qu'on a une colonne avec un pixel noir on chercher la première colonne sans pixel noir, des que c'est le cas c'est qu'on a entoure une lettre. Et donc petit a petit chaque lettre est détourée. On regarde aussi cette surface, si elle fait moins de 2% de l'espace total

de l'image on part du principe que c'est un bruit parasite.

3.2.3 Des lettres dans les mots de la liste

Pour ce faire nous faisons exactement la même chose que pour la détection des lettres dans la grille
Il nous arrive d'avoir quelque ratés, en effet sur des images il arrive que les lettres soient collées avec prétraitement de l'image et donc nous n'arrivons pas a détecter la lettre seule.

3.3 Réseau de neurone du XNOR

Pour créer un réseau de neurones nous permettant de reconnaître le XNOR ($\overline{AB}$ + AB), nous avons commencé par implémenter des réseaux à un neurone afin de commencer à se familiariser avec toutes les notions relatives aux réseaux de neurones. Après cela, nous avons enchaîné avec une version très explicite du réseau de neurone associé au XNOR en prenant trois neurones. Les trois neurones nous serviront à décomposer notre XNOR en plusieurs porte logique. En effet, posons X1 et X2 deux entrés du réseau de neurones, alors nous savons que X1 XOR X2 sortira le même résultat que (X1 NAND X2) AND (X1 OR X2). Sachant cela nous pouvons représenter graphiquement notre neural network XOR comme dans la figure 1 ci-dessous

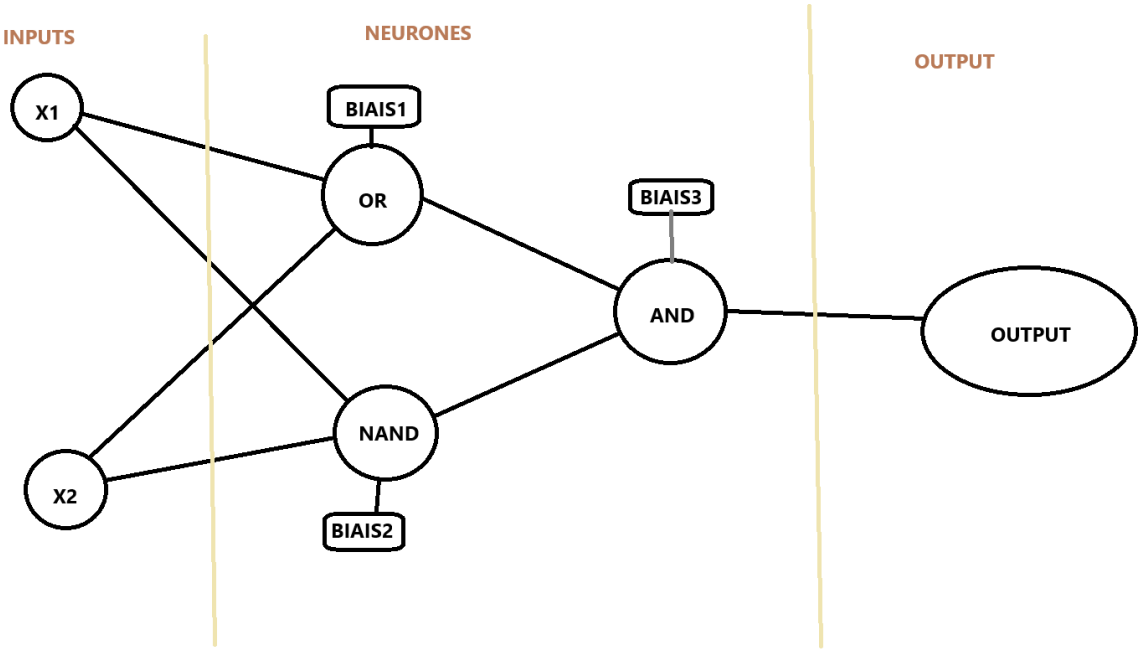

Figure 1: réseau de neurones du XOR/XNOR

Afin d'entrainer notre réseau relatif au XNOR, nous avons décidé d'adopter la manière la plus simple pour le moment, faire la décente gradienne manuellement, calculer séparément toutes les dérivées partielles ainsi que les pas etc. Ainsi notre code est plus explicite ce qui nous a aidé à comprendre les différentes étapes d'entrainement du réseau.

3.4 Algorithme de résolution

3.4.1 Attendus

Le solver attendus doit pouvoir être utilise en format ligne de commandes, il prendrait deux paramètres: le premier un nom de fichier vers

une grille de mots mêles uniquement en majuscules, et le deuxième serait un mot a trouver dans cette grille. Le programme renvoie sur la sortie standard deux couples d'entiers correspondants respectivement aux coordonnes de la première et de la dernière lettre du mot a trouver. Si le mot n'est pas trouve dans la grille le programme écrit "Not Found" toujours sur la sortie standard

3.4.2 Première implémentation - naïve

La première version de notre programme était voulu naïve et brut-force. Le principe était simple: on commençait par charger le fichier texte de la grille et nous le parcourions deux fois, une pour compter les retours a la ligne et le nombre de lettres par lignes, ainsi on connaissait les dimensions de la grille et on pouvait vérifier qu'il n'y ait pas d'erreurs dedans, si un caractère n'est pas une lettre ou si toutes les lignes n'ont pas la même taille. Nous pouvions donc allouer la taille de la grille sur le heap et parcourir une deuxième fois pour remplir ce tableau.
Ensuite on parcourait la matrice représentant la grille et pour chaque lettre on vérifiait si elle correspondait a la première du mot cherche et si c'était le cas on testait dans les huit directions si cela correspondait a notre mot.
A la fin si on avait des coordonnées on les affichait sinon on renvoyait "Not Found".
Cette méthode est simple a écrire, cependant elle est très coûteuse, encore plus si on se met dans le cas ou on cherche plusieurs mots dans la même grille, pour chaque mot on recharge la grille et on re-parcourt en vérifiant pour chaque lettre. C'est pour cela que nous avons changer d'idée pour l'implémentation retenue.

3.4.3 Implémentation retenue - Présentation

La version que nous utilisons actuellement du solver est pensée différemment: nous partons du principe que nous cherchons une liste de mots dans une grille. A partir de cette liste de mots nous allons construire un arbre ou les mots seront regroupes par rapport a leurs suffixes, ce qui facilitera la reconnaissance de lettres. Ensuite les différentes positions seront dans une liste chaînée.

3.4.4 Implémentaton retenue - Structures utilisées

Dans cette implémentation nous avons trois structures distinctes

- Grid
Cette structure nous permet juste de stocker le contenue du fichier de la grille ainsi que ces dimensions, elle est pas d'une grande utilité mais permet de simplifier le code et d'éviter d'avoir des fonctions a rallonges. Elle a différente fonctions associées, une recupérant les dimensions, une autre qui l'initialise et une qui la remplie. Ainsi qu'une dernière qui free ses différents attributs.

- Arbre de mots
Cette structure est centrale a notre algorithme, c'est elle qui va contenir tout nos mots, elle est sous forme d'un arbre général avec une implémentation premier fils/frère droit. La racine sera vide pour des raison d'implémentation mais les autres nœuds auront un char comme clé. Le premier niveau comportera la première lettre de chacun des mots, le deuxième niveau deuxieme lettre et ainsi de suite. Ainsi les mots 'papa' et 'pedantic' auront un nœud en commun, le premier p.
L'idée va donc être de construire cet arbre avec la liste des différents mots, il y a une fonction pour initialiser l'arbre avec la racine et une

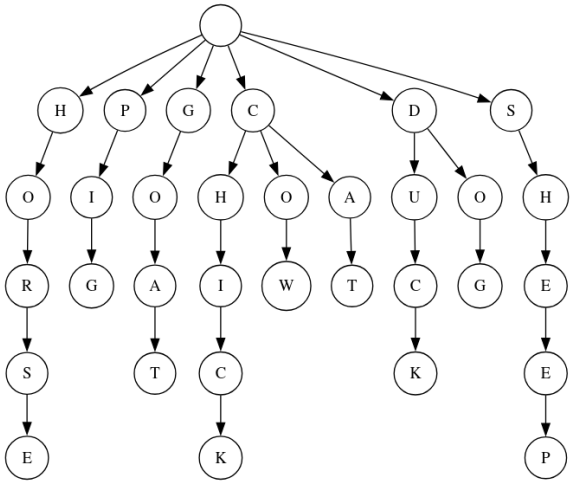

Figure 2: Exemple d'arbre de mots

pour y ajouter un mot. Il y a aussi une fonction pour créer un arbre a partir d'une liste de mots et qui s'occupe de l'initialiser. Il y a aussi une fonction pour enlever un mot de l'arbre, ce sera pour quand on aura trouver un mot. Il y a la possibilité de détruire l'arbre et ainsi libérer la mémoire. Il y a d'autres fonctions pour connaître le nombre d'enfants d'un nœud, savoir si c'est une feuille ou connaître la taille de l'arbre.
Il y a aussi la fonction get-child qui prends en paramètres un arbre et un char et renvoie le fils de l'arbre donne ayant comme clé le char donne, ou null si il n'y en a pas. Cette fonction nous permettra de chercher les motifs dans la grille et ainsi voir si le construit dans la grille fait parti de la liste. Un exemple d'un arbre de mots:

- Liste de positions
Cette structure est une structure de liste chaînée un peu banal, chaque maillon de la chaîne contient un pointeur sur char qui est le mot auquel

elle réfère, deux points de positions et un pointeur vers le prochain maillon. Un point de position est une structure de deux entiers, un x et un y, elle n'est pas vraiment utile mais sert juste a clarifier le code. Donc chaque mot de la liste est sensé être un mot trouve avec sa position de début et sa position de fin.
Elle n'a que des fonctions banales de liste chaînée, la possibilité de l'initialise, y ajouter un élément, l'afficher ou encore la détruire pour libérer toute la mémoire.
Cette liste nous permet d'avoir le résultat pour chacun des mots trouves dans la grille.

3.4.5 Implémentaton retenue - Algorithme

Comme explique précédemment on va tout d'abord charger la grille de mots, construire notre arbre générale contenant touts les mots a chercher et initialiser une liste ou mettre les résultats.
Ensuite on va parcourir notre grille, pour chaque lettre de notre grille on va aller utiliser get-child de notre arbre et la lettre courante. Si le résultat est null alors aucun mot ne commence par cette lettre et on continue. Par contre si cela correspond on va essayer de voir si c'est la position de départ d'un des mots a trouver.
Pour ce faire on va chercher dans les 8 directions (gauche, droite, haut, bas, haut-gauche, haut-droite, bas-droite et bas-gauche). Et on va get-child de la lettre de ou on se trouve et tant qu'on est dans la grille et que ce résultat n'est null on continue. Si le nœud de l'arbre qu'on a récupérer est une feuille alors c'est qu'on a trouve un mot. Donc on ajoute a notre list un maillon avec ce mot, la position de ou on est partis, la position ou on est actuellement et un pointeur vers null car c'est la queue de la liste. On enlève le mot de l'arbre car on a plus besoin de le chercher.
Une fois tout la grille parcourue on détruit l'arbre et la grille car on

en a plus besoin et retourne la liste de positions des mots.

3.4.6 Implémentaton retenue - Programme solver

A ce stade on pourrait se demander qu'en est il de l'implémentation demandée. En effet dans l'implémentation choisie on travaille avec une liste de mots or la on nous demande de faire un programme qui prends juste une grille et un mot en paramètres. Pour ce faire on a un autre programme qui réutilise l'implémentation présentée. On appelle notre programme avec la grille donnée et une référence vers notre mot, ce qui fait une liste de longueur 1. Si la liste renvoyée est null on écrit "Not Found" sur la sortie standard sinon on écrit les coordonnées de la lettre de début et de fin du mot.

3.4.7 Prochaines étapes

A ce stade nous considérons le solver comme étant fini a 100%. Cependant on pourrait avoir a le modifier, que ce soit pour l'améliorer ou corriger des bugs qui n'ont pas encore été trouves. Et sans doute que la structure de la liste chaînée doive être modifiée quand nous aurons besoin de nous en servir dans les prochaines étapes.

4 Avance prise à l'initiative du groupe

4.1 Réseau général avec descente gradienne stochastique

Un réseau de neurone permettant de reconnaître des lettres de l'alphabet à partir d'une image est plus complexe qu'un réseau de neurone re-

connaissant uniquement le XNOR, il a besoin de plus de neurones. De ce fait, son architecture doit être plus généraliste, et le code doit laisser la possibilité d'expérimenter lors de l'entraînement du réseau, en changeant par exemple la taille du réseau ou encore la configuration de ses couches cachées. De plus, les fonctions d'entraînement du réseau doivent être plus optimisées.

C'est pour cela que nous avons décidé de prendre de l'avance et de coder directement un réseau de neurone bien structuré, implémentant également la descente gradienne dite "stochastique", qui est une technique consistant à entraîner notre réseau seulement sur une petite sélection de données choisies aléatoirement parmi un ensemble plus large, avant de continuer sur une autre petite sélection, jusqu'à voir tout l'ensemble. Cette technique a l'avantage de réduire le temps de calcul de chaque "pas d'apprentissage" que le réseau va effectuer, tout en gardant une bonne approximation du pas optimal à effectuer, car la sélection est choisie aléatoirement de manière uniforme parmi l'ensemble complet.

4.2 Données et fonctions annexes à l'entraînement du réseau

Il existe plusieurs fonctionnalités qui ne sont pas nécessaires pour entraîner le réseau, mais qui permettent une meilleure utilisation de celui-ci. En effet, si les données d'entraînement du XNOR peuvent être directement écrites manuellement dans le code, car il n'y a en fait que 4 entrées possibles pour un réseau XNOR, le nombre d'entrées possibles pour un algorithme de reconnaissance de lettres dans une image est gigantesque. Il faudra alors pouvoir charger plusieurs centaines de milliers de données d'entraînement pour entraîner le réseau convenablement.

Pour ceci, on utilise une fonction qui parcourt un large fichier texte,

téléchargé préalablement dans un dépôt de données pour entraînement de réseaux de neurones, puis charge ce fichier dans la mémoire vive sous forme d'une large structure qui sépare chaque donnée, classifiant l'entrée qui sera fournie au réseau, puis le résultat attendu, pour pouvoir corriger le réseau si celui-ci donne un résultat erroné.

L'entraînement d'un réseau de neurone assez large peut être un processus chronophage, de ce fait, après avoir entraîné le réseau de neurone, il faut pouvoir le sauvegarder, afin de le réutiliser par la suite sans devoir l'entraîner avant chaque utilisation. C'est pour cela que nous avons développé deux fonctionnalités permettant de sauvegarder un réseau de neurone dans un fichier, puis de le charger à partir de ce même fichier.

5 Planification et répartition des taches

5.1 Répartition des taches

Traitement d'image	Victor
Détection des éléments	Victor Adrien
Intelligence artificielle	Addrien Mael Jaisy
Résolution de la grille	Victor
Interface graphique	Victor Jaisy

5.2 Avancement du projet

Traitement d'image	90%
Détection des éléments	40%
Intelligence artificielle	60%
Résolution de la grille	100%
Interface graphique	10%

5.3 Planification des taches

Tache	Septembre	Octobre	Novembre	Décembre
Traitement d'image	50%	90%	100%	100%
Détection des éléments	0%	40%	80%	100%
Intelligence artificielle	30%	60%	90%	100%
Résolution de la grille	100%	100%	100%	100%
Interface graphique	0%	10%	40%	100%

Voici le tableau de planification des taches, avec pour chaque mois l'avancement prévu de chaque tache majeure. Nous détailleront maintenant ce tableau.

- Le traitement d'image est primordiale pour pouvoir tester le réseau de neurone, et travailler sur la détection des élément de l'image, il sera alors fini en priorité par Victor. Cette tache est actuellement presque totalement fonctionnel, Victor y ayant prêté une grande attention. Elle pourra donc être finie des le mois de novembre.

- La détection des différents élément de l'image (puis leur découpage en sous-images) est une étape très importante du programme, cependant, elle n'a pas reçu de priorité spéciale n'étant pas une nécessité pour le développement d'autres taches. Son développement est actuellement assez primitif mais pose les bases pour son implémentation. Cette tache sera une priorité pour les membres devant s'y pencher lors du mois de novembre, puis elle sera complétée en décembre.

- L'intelligence artificielle sous la forme d'un réseau de neurone est un axe majeur de ce projet, c'est pourquoi le développement de cette facette a débuté dès septembre. En septembre, le réseau XNOR a été implémente par Adrien, ce qui a fourni une base pour le développement du réseau de neurone de reconnaissance d'image. Actuellement, toutes les fonctionnalités importantes

lies au réseau sont implémentées, cependant, elles ne sont pas toutes fonctionnelles. Il faudra donc régler tous les problèmes durant le mois de novembre, puis enfin optimiser durant le mois de décembre.

- La résolution de la grille est la dernière étape du programme, cependant c'est cette tache qui se rapproche le plus d'un algorithme classique, avec une donnée d'entrée facile a modéliser. C'est pour cela qui Victor a pu prendre de l'avance sur cette tache en la réalisant dès septembre. Grace a cette initiative, cette tache est totalement complétée et ne représente pas une charge de travail pour l'équipe.

- Enfin, l'interface graphique est une facette plus triviale a développer, mais a ne surtout pas négliger pour autant, car c'est avec elle que va interagir l'utilisateur. Cependant, dans le cadre de notre logiciel, l'interface graphique peut se permettre d'être minimale, c'est pourquoi son développement n'est que très peu avance actuellement. En novembre, l'interface graphique sera légèrement complétée, puis sera probablement une des dernières taches sur laquelle l'équipe se penchera avant de compléter le projet en décembre.

5.4 Conclusion

On a donc vu et rappelé dans ce document le but du projet OCR, l'équipe NVAM, ses membres, l'avancée actuelle des différentes taches, l'avance prise ainsi que la répartition des taches et la planification de celles ci. Les prochains mois devront êtres marqués d'efforts continués pour pouvoir répondre aux attentes de ce projet.