OCR - Soutenance 1

Jacque SAUDUBRAY, Kylian GILBERT, Bastien GAULIER, Paul PAZART

October 2022

Table des matières

1 Introduction

1.1 Qui sommes-nous ?

Nous sommes une équipe composée de Kylian Gilbert, Jacques Saudubray, Bastien Gaulier et Paul Pazart. Au sein du projet, Kylian et Jacques s'occupent du prétraitement de l'image et du découpage des cases ; Paul s'occupe du réseau de neurones. Bastien, quant à lui, s'occupe du résolveur de sudoku et épaule Paul dans la construction du réseau de neurones.

Notre projet consiste à résoudre un sudoku utilisant la reconnaissance de caractère, grâce à l'intelligence artificielle d'un réseau de neurone et utilisant un traitement précis sur l'image.

1.2 Environnement de travail

Pour mener à bien ce projet, nous avons établi un dépôt Github afin que nous puissions échanger et progresser sur notre projet à tout moment. Nous avons également créé un serveur discord pour communiquer entre nous. Enfin, un Trello a été mis en place afin de suivre les tâches en cours et restantes en temps réel.

1.3 Attente pour la première soutenance

Pour cette première soutenance, nous voulions que les éléments attendus soient terminés. Ces éléments sont le chargement et suppression des couleurs, la rotation manuelle de l'image, la détection de la grille et de la position des cases, le découpage de l'image et la sauvegarde de chaque case sous la forme d'une image, la création d'un résolveur de sudoku et la création d'un réseau de neurones qui simule une porte xor.

Nous voulons aussi que la création du réseau définitif avec les sauvegardes et les améliorations de performances (adaboost, softmax,...) soit commencée, et qu'un début de recherche soit initié pour le redressement automatique de l'image.

2 Récit

2.1 Traitement d'image

2.1.1 Introduction

Pour tout ce qui touche au traitement d'image, nous avons divisé le travail en plusieurs tâches que nous nous sommes ensuite attribuées. Comme dit ci-dessus Kylian et Jacques ce sont occupés de cette partie. Nous avons chacun réalisé les tâches de notre côté et nous les avons mises en commun sans problèmes. Pour ce qui est de la répartition de ces tâches, Jacques s'est occupé de la supression du bruit, de la binarisation de l'image et de la detection de la grille. Kylian a de son coté realisé la rotation manuelle de l'image ainsi que la détection des cases. Il faut savoir que tout le pretraitement a été réalisé a l'aide de la librairie SDL2 compatible avec les programmes C.

2.1.2 Supression des couleurs

La suppression des couleurs consiste à modifier l'image afin d'obtenir un panel de gris correspondant aux couleurs initiales de l'image. Cette partie a été réalisée lors d'un TP de programmation et cela était bien guidé. Nous n'avons eu aucun problème sur cette partie et cela ne nous a pas pris trop de temps. Il suffit de modifier les composantes r, g, b de chaque pixels de l'image afin d'obtenir du gris.

2.1.3 Supression du bruit

En ce qui concerne la suppression du bruit, il s'agit de compléter les endroits où il manque des pixels de bonne couleur. Cette suppression de bruit est très utile notamment afin de rendre les lignes et colonnes de la grille de sudoku plus épaisses et continues. Une méthode fonctionnant bien est le flou gaussien. Elle consiste à appliquer une couche de flou sur l'image en prenant chacun de ses pixels et en propageant sa couleur sur les pixels adjacents. Nous n'avons rencontré aucun problèmes pour l'implémentation du flou gaussien. Si nous avons le temps, nous utiliserons les matrices de convolution permettant de mieux manipuler le floutage et rajoutant d'autres fonctionnalités. L'utilisation de ces matrices n'est cependant pas nécessaire, car le flou gaussien que nous utilisons fonctionne déjà très bien.

FIGURE 1 – Applicatiopn du flou gaussien et suppresion des couleurs

2.1.4 Binarisation de l'image

La binarisation de l'image permet de se retrouver avec une image aux pixels uniquement noirs ou blancs. C'est une étape nécessaire pour la suite du traitement d'image. Plusieurs méthodes sont utilisées pour la binarisation d'image. Nous avons choisi d'utiliser la méthode d'Otsu, celle-ci étant la plus courrament utiliser et l'une des plus simples a implementer. Elle consiste à déterminer un seuil tel que si la couleur d'un pixel est au-dessus de ce seuil, ce pixel devient noir et à l'inverse, celui-ci devient blanc. Ce seuil est calculé sur l'ensemble de l'image avec la couleur de chaque pixel. Nous avons réussi à implémenter correctement cette méthode, mais un problème est survenu pour certaines images. En effet, lorsqu'il y a des zones d'ombres sur l'image, cela créait des gros nuages noirs et pouvait nuire au traitement d'image. Pour cela, nous avons modifié la méthode afin de déterminer plusieurs seuils à des échelles locales. Ainsi, nous avons découpé l'image en plusieurs cases et nous avons déterminé un seuil pour chacune d'entre elles, étant ensuite appliqué à chaque pixels de cette zone. Cette méthode marche, mais certaines des images peuvent encore rencontrer des problèmes suivant le nombre de zones que l'on prend pour découper l'image. Nous devons encore faire des modifications manuellement pour chacune des images afin d'avoir un rendu optimal mais

nous aimerions automatiser cela. Une solution serait de prendre un seuil adaptatif à l'échelle de chaque pixels afin d'avoir une binarisation parfaite de l'image.

Une autre fonctionnalité a été rajoutée afin d'inverser les couleurs de l'images. En effet pour un meilleur traitement de l'image nous voudrions une grille blanche, incluant les chiffres en blanc sur un fond noir. Pour cela nous avons créer une fonction permettant de voir si la majorité des pixels sont blancs ou noirs et les modifier quand il le faut en inversant leur couleurs. Aucun problème de ce coté a signaler, la fonction étant simple a réaliser.

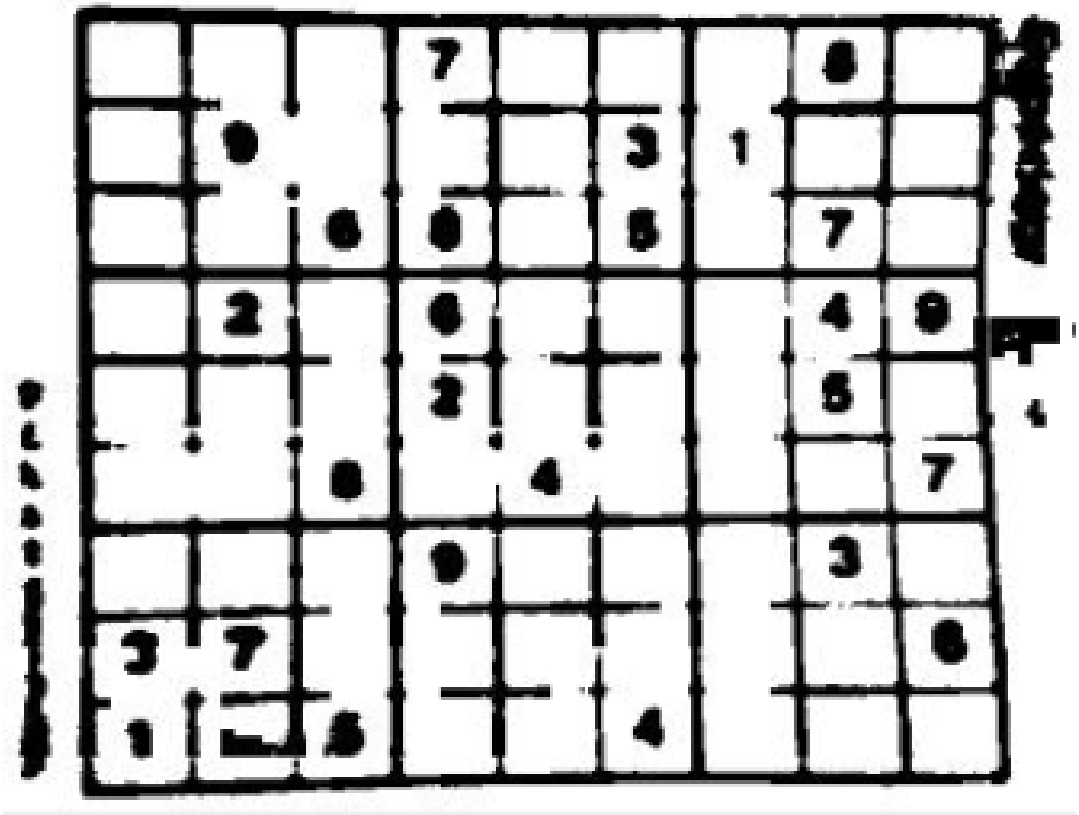

FIGURE 2 – Binarization d'image avec la méthode d'Otsu

2.1.5 Rotation de l'image

La rotation de l'image se fait en 2 étapes, la première est la rotation avec l'angle voulu, et ensuite, il faut faire une translation. Ces deux étapes peuvent se faire ensemble puisqu'on regarde sur une nouvelle image vierge quels pixels récupérer en appliquant les étapes dans le sens inverse. C'est-à-dire appliquer la translation puis faire la rotation dans le sens inverse via une multiplication de matrices. Le fait de le faire dans ce sens permet d'éviter les pixels manquants, car la multiplication de matrices utilise la trigonométrie ce qui implique l'utilisation de cosinus et sinus, fonctions qui ne

retournent que rarement des nombres entiers. Or, la position d'un pixel sur l'image est constituée de deux entiers. L'arrondi aurait ainsi pu créer des trous dans l'image retournée. Le plus gros défaut de cette méthode est la perte ou le surplus en cas de rotation particulière telle que 45°. Mais on a aucune perte pour des rotations de 90° ou 180°. Donc, par principe de précaution, on est obligé d'agrandir l'image. De toute façon, elle sera redimensionnée plus tard.

2.1.6 Détection de la grille

Pour ce qui est de la détection de la grille, nous avons décidé de la détecter à l'aide d'une méthode particuliere consistant a trouver le plus large élément connexe de l'image. Pour cela, il faut parcourir les pixels et voir les zones de pixels isolées étant formées par des pixels en contact les uns les autres. Puis on garde la plus grande zone de pixels en les comparant toutes ensemble en foncton de leur nombres de pixels, étant donc celle correspondant a la grille. Il nous reste qu'a retrouver les coordonnees de la grille et a délimiter l'image a ces coordonnées. On a cependant rencontré des problèmes avec les différentes grilles. Si il y a un autre élément sur l'image ayant plus de pixels, le programme ne va pas reconnaitre la grille et prendra ce nouvel élément a la place.

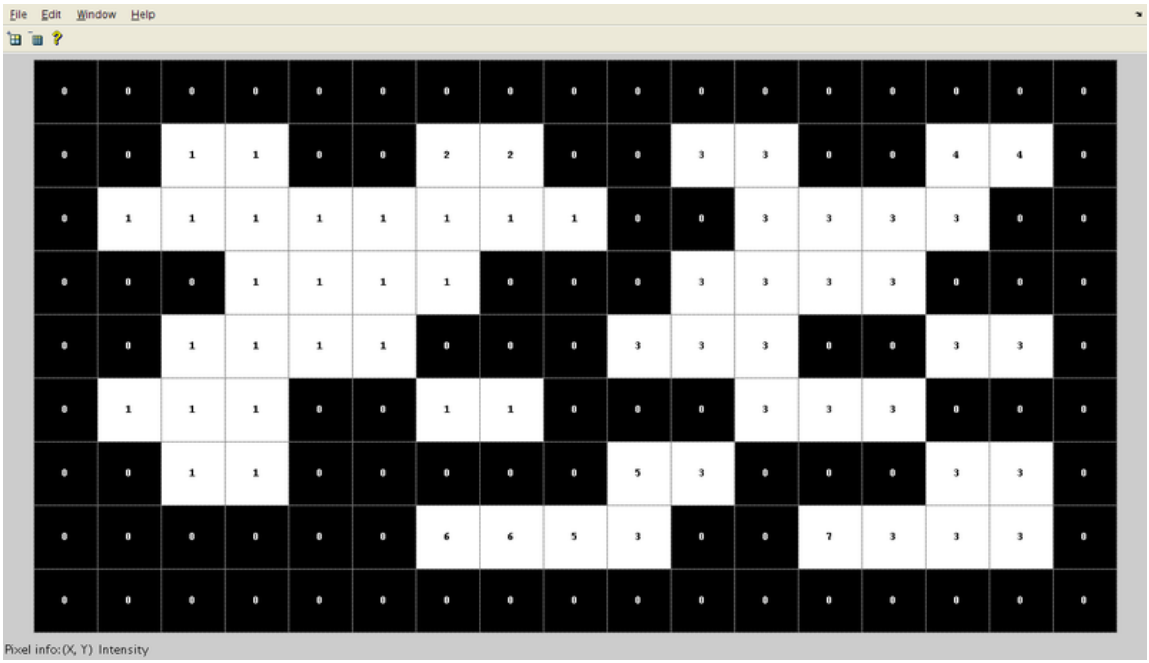

FIGURE 3 – Element connexe le plus grand d'une matrice

2.1.7 Decoupage des cases

Nous n'avons pas encore terminé cette partie car nous avions besoin que la détection de grille soit totalement fonctionnelle avant. Une fois l'image redimentionnée autour de la grille nous avons prévu de détecter les lignes a l'aide de la transformée de Hough. Cette méthode est très efficace et nous avons deja commencer a travailler dessus. La transformée de Hough se fait en deux étapes

distinctes. Puisque Hough à découvert qu'on pouvait représenter facilement toutes lignes par une courbes sinuosidale. On peut voir sur l'image qu'il y a différentes intensités. de couleur blanches sur la courbes. Ainsi pour récupérer les lignes que l'on veut garder on regarde en fonction d'un "threshold", c'est à dire une moyenne et si le point est supérieur à cette moyenne alors on dessines la ligne.

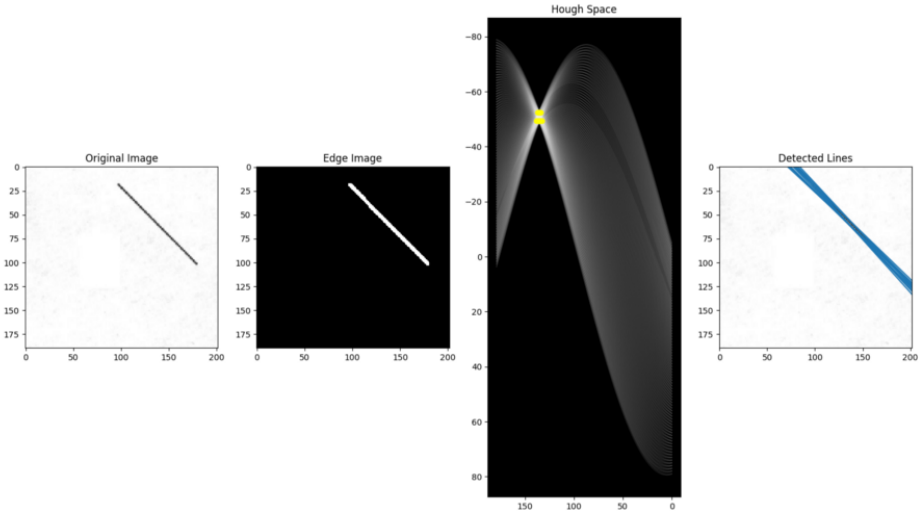

FIGURE 4 – Schéma expliquant la transformée de hough

Donc pour avoir ces lignes à détecter il faut mettre en place un "edge detection", c'est à dire une détection des lignes. Un des algorithmes les plus connues et utilisé est celui de Canny. Il permet de mettre en avant toutes les lignes d'une images ce qui permet donc par la suite d'appliquer la transformée de hough vu au dessus.On n'a malheureusement pas eu le temps de le mettre en place pour cette soutenance.

FIGURE 5 – Schéma expliquant l'algorithme de Canny

2.2 Réseau de neurones

Pour cette soutenance, nous avons tenu le délai pour la création d'une maquette de notre réseau de neurones. L'algorithme a été pensé pour être le plus simple possible et fonctionnel. L'implémentation du réseau a été réalisée par Paul PAZART et le système de sauvegarde par Bastien GAULIER.

2.2.1 Implémentation

Le réseau de neurones implémenté est un ou exclusif (XOR) comme attendu dans le cahier des charges. Il est composé de deux nœuds de traitement internes et d'un de traitement de sortie. Il utilise le principe de la descente de gradient pour la "backpropagation" en plus du "forward pass". Le principe de la descente de gradient est le calcul les erreurs pour chaque exemple de l'ensemble de données de formation. Cependant, elle ne le fait qu'après que chaque exemple de formation ait fait l'objet d'une évaluation rigoureuse. Il est juste de comparer ce processus à un cycle. Certains parlent également d'une époque de formation. Nous avons choisi la descente par lots qui présente plusieurs avantages. Tout d'abord son efficacité de calcul qui est extrêmement pratique, car elle développe une convergence stable et un gradient d'erreur stable. Cela dit, la descente de gradient par lots présente également quelques inconvénients. Parfois, son gradient d'erreur stable peut entraîner un état de convergence défavorable. En outre, il a également besoin de la présence de l'ensemble de données d'entraînement dans son algorithme et sa mémoire. La "back propagation", quant à elle, est l'essence même de la formation des réseaux neuronaux. Il s'agit de la méthode de réglage fin des poids d'un réseau neuronal en fonction du taux d'erreur obtenu dans l'époque (c'est-à-dire l'itération) précédente. Un réglage correct des poids permet de réduire les taux d'erreur et de rendre le modèle fiable en augmentant sa généralisation. Enfin, celle du "forward pass" est de transmettre les données d'entrée dans le sens direct à travers le réseau. Chaque couche cachée accepte les données d'entrée, les traite selon la fonction d'activation et les transmet à la couche suivante.

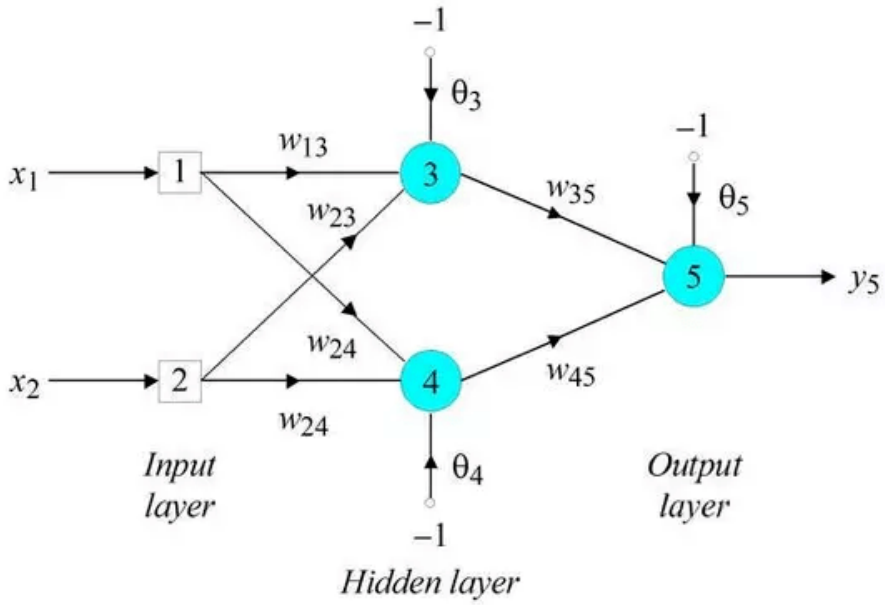

FIGURE 6 – Schéma de notre réseau neuronal. w représente le poid, thêta le biais et y5 la sortie

2.2.2 Système de sauvegarde

Un réseau de neurones fonctionnel, c'est très bien, s'il est entraîné, c'est encore mieux, mais si il n'a pas de moyen de se rappeler cet entraînement, ça ne sert à rien. C'est pourquoi il était nécéssaire d'implémenter un système de sauvegarde à notre réseau. Ainsi, pour cette soutenance, cela a été réalisé pour notre petit réseau neuronal mettant en place le XOR, et cela est en cours d'adaptation vers un réseau général.

Mais que sauvegarder exactement ? Les informations qui varient en fonction de l'entraînement du réseau sont les poids des liaisons et les biais des neurones. Dans le code, ceux-ci sont représentés sous formes d'arrays à une ou deux dimensions dans le cas du XOR ; ainsi, le système de sauvegarde ne va vraiment enregistrer que des listes de nombres décimaux (des *(*double)).

Ainsi, un fichier de sauvegarde ne sera qu'une suite de listes de nombres et de listes de ces listes. Dans le cas du réseau XOR, cela ressemblera à cela :

```
$ cat XOR3.txt
{ 0.000409 0.040547 }
{ 0.008496 }
{ -7.801209 -3.163903 }
{ -5.205072 }
[
{ 5.089568 6.979848 }
{ 5.089150 6.978140 }
]
[
{ -11.804881 }
{ 11.105633 }
]
```

FIGURE 7 – Fichier de sauvegarde du XOR.

Cependant, ce format risque de changer d'ici au rendu final ; nous envisageons tant de tout enregistrer dans un seul fichier comme de répartir les informations en fichiers distincts.

2.3 Traitement du sudoku

Pour cette soutenance, nous avons respecté notre objectif de faire le traitement du sudoku. Tout d'abord, la résolution du sudoku lui-même a été réalisée : L'algorithme de résolution a été étudié, mis en place puis optimisé. Ensuite, le système de sauvegarde de la grille a été fait, de façon à ce qu'on puisse non seulement sauvegarder une grille mais aussi en charger une. Cette section a été réalisée par Bastien GAULIER.

2.3.1 Le sudoku

Commençons par rappeler l'objectif de notre OCR : détecter et résoudre un sudoku. Tout d'abord, qu'est-ce que c'est ? Un sudoku est un jeu d'origine japonaise mettant en scène une grille de 9x9 cases, divisés en carrés de 3x3 cases appelés régions. Le but du jeu est de remplir cette grille avec des chiffres de 1 à 9 de façon à ce que chaque ligne, chaque colonne et chaque région ne comporte pas plus d'une fois le même chiffre. De plus, au début du jeu, certaines cases sont déjà occupées par des chiffres. Il faut aussi ajouter que, bien que tous les sudokus que l'on retrouve dans les magazines de jeux sont solubles, il existe certaines dispositions de chiffres qui, sans rendre la grille incorrecte la rend infinissable.

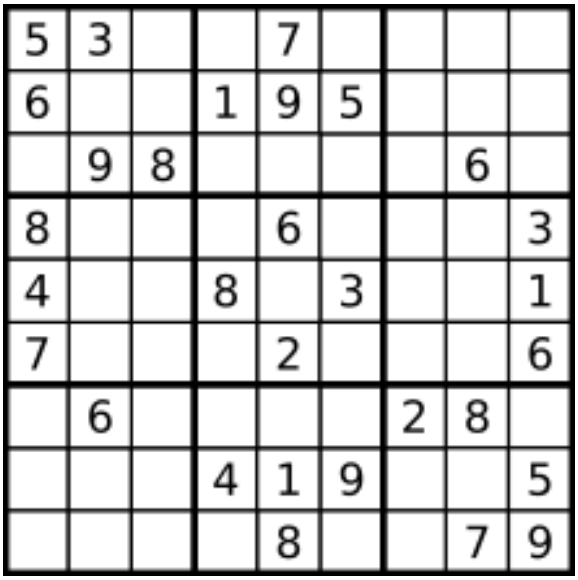

FIGURE 8 – Un sudoku solvable.

2.3.2 Algorithme de résolution

Afin de résoudre la grille de sudoku qui sera détectée par notre OCR, nous nous sommes tournés vers l'algorithme du **Backtracking**. A la fois simple et assez rapide, il était pour nous le choix évident pour notre projet, d'autant que nous avions déjà eu l'occasion de le manipuler.

Son principe, récursif, est simple : on va parcourir toutes les cases du sudoku ligne par ligne en partant du coin supérieur gauche. Pour chaque case vide, on va essayer d'insérer des nombres pour ensuite vérifier si les règles du sudoku sont respectées. Si l'algorithme trouve une solution au sudoku, il renvoie VRAI et la grille qu'on lui a donnée en entrée est modifiée pour représenter la solution. Sinon, il renvoie FAUX et la grille de départ est intacte.

Ainsi, voici les grandes lignes de cet algorithme pour une case C :

— Si C est en-dehors de la grille, on a potentiellement trouvé la solution : il faut simplement vérifier si la grille est encore valide. On renvoie ce résultat.
— Si on est revenu au début sans trouver de solution, c'est qu'il n'y en a pas : on renvoie FAUX.
— On vérifie si la grille respecte encore les règles du sudoku. Si non, on renvoie FAUX.
— Si C n'est pas vide, alors on appelle l'algorithme sur la case suivante.
— Sinon, on y place tour à tour les chiffres de 1 à 9, et pour chacun, on appelle l'algorithme sur la case d'après.
— Si l'un de ces appels récursifs renvoie VRAI, il faut aussi renvoyer VRAI car on a trouvé la solution.
— Si aucun appel récursif n'a trouvé de solution, on n'en a pas trouvé sur cette case : on renvoie FAUX.

Une fois que nous avons implémenté cet algorithme, il nous est vite apparu qu'il n'était pas très

optimisé, et bien que le temps de son exécution soit bien en-dessous de la seconde, nous voulions sauver le maximum de temps possible. En effet, les parties du traitement d'image prendront assez de temps comme cela, nous ne voulons pas ralentir encore plus notre programme.

Ainsi, nous avons mis en place un niveau d'optimisation à notre algorithme de backtracking, qui se remarque lors de l'étape d'insertion des chiffres dans la case courante. Désormais, l'algorithme n'essaye pas aveuglément tous les chiffres de 1 à 9 pour une case : il ne va placer que les chiffres qui ne rendront pas la grille invalide. Cela enlève l'étape de vérification très gourmande pour une rapide vérification pour chaque chiffre. Dès lors que nous avons instauré cela, l'algorithme s'exécute très rapidement, même pour les grilles insolubles.

Voici alors le nouvel algorithme :
- Si C est en-dehors de la grille, on a trouvé la solution : on renvoie VRAI.
- Si on est revenu au début sans trouver de solution, c'est qu'il n'y en a pas : on renvoie FAUX.
- Si C n'est pas vide, alors on appelle l'algorithme sur la case suivante.
- Sinon, on y place tour à tour les chiffres qui gardent la grille valide, et pour chacun, on appelle l'algorithme sur la case d'après.
- Si l'un de ces appels récursifs renvoie VRAI, il faut aussi renvoyer VRAI car on a trouvé la solution.
- Si aucun appel récursif n'a trouvé de solution, ou il n'existe aucun chiffre ne compromettant pas la grille, on n'a pas trouvé de solution sur cette case : on renvoie FAUX.

2.3.3 Système de sauvegarde

Après avoir détecté et résolu un sudoku, notre OCR doit aussi être capable de le sauvegarder, pas simplement de l'afficher puis d'en effacer toute trace. Ainsi, pour cette soutenance, ce système a été réalisé, de façon à ce qu'une grille de sudoku, après avoir été résolue, est enregistrée dans un fichier sur la machine.

Le format utilisé dans le fichier texte de sauvegarde est très simple : la grille y est représentée telle quelle, les chiffres de 1 à 9 les uns à la suite des autres dans un carré de 9*9 caractères. Les cases vides sont représentés par un point('.').

```
$ cat solved/s2_solved.oku
52..7....
6..195...
.98....6.
8...6...3
4..8.3..1
7...2...6
.6....28.
...419..5
....8..79
```

FIGURE 9 – Fichier de sauvegarde d'un sudoku insoluble

Il faut aussi noter que notre système de sauvegarde enregistre quand même un sudoku même si il n'est pas soluble. En effet, il peut être utile de se rappeler une grille détectée par l'OCR, ou simplement pour montrer un exemple de sudoku insoluble. De plus, il est aussi possible pour ce système de charger un sudoku à partir d'un fichier de sauvegarde.

Enfin, our l'instant, les systèmes de résolution de sudoku et de sauvegarde de grille sont indépendants du reste de l'OCR, mais leur fusion est préparée et sera simple à réaliser.

2.4 Emois

2.4.1 Jacques

Voici mon ressenti sur le projet et le travail réalisé jusqu'a maintenant : Lorsque j'ai regardé l'ensemble du projet au départ, je me suis dit que le travail allait etre conséquent. Puis en se répartissant les taches, on se rend compte que ce n'est pas si difficile que ca. Il y a plein de ressources sur Internet pouvant nous aider meme si certaines méthodes restent compliquées a implémenter. J'ai eu quelques problèmes sur mes taches et certaines ce sont pas encore résolues mais plus le projet avance, plus j'apprend a manipuler correctement les librairies SDL2 et la programmation C permettant de trouver des solutions de plus en plus rapidement.

Concernant le groupe, je trouve que l'ambiance est tres bonne. Chacun a ces propres taches et s'y met a cents pour cents. Mais il arrive aussi de nous entraider quand nous avons du mal sur certaines taches, notamment par groupes de deux. Kylian et moi meme car nos taches se ressemblent et nous travaillons sur le traitement d'image tout les deux et Bastien et Paul entre eux.

Meme si certaines taches n'ont pas été complétées a temps, il reste du temps avant la soutenance

finale et j'ai trouvé le rythme de travail adapté au projet. J'ai pris un peu de temps a me mettre dans ce rythme notamment jusqu'au tp de programmtion sur la bibliothèque SDL. Plus je travaille sur le projet, plus je le trouve intéressant et cela donne envie de perfectionner tout ce que l'on a implémenté. Je suis confiant sur ce qui va arriver par la suite et je pense que nous arriverons a rendre un travail complet et qui marche pour la dernière soutenance.

2.4.2 Kylian

Ce projet me faisait peur au début, car je ne connaissais rien de ce qu'il fallait faire et je n'avais pas trop d'idée de comment commencer. Il y avait plein de moyens d'appréhender ce projet, vu le nombre de manière différentes possible. Avec un langage et un environnement que je viens de découvrir. De ce fait, j'ai pris du retard au niveau de la partie d'implémentation, mais j'ai compris comment faire pour la suite du projet. Néanmoins je prends quand même du plaisir à voir ce projet avancer et se concrétiser. Au final, je regrette un peu de ne pas avoir avancé plus.

2.4.3 Bastien

A propos de ce que j'ai pu ressentir dans cette période de ce projet, je dirais que ça a été assez progressif. Premièrement, lorsque nous avons commencé à répartir les tâches entre nous, je ne connaissais rien du langage utilisé, le C, ni vraiment de ce que nous allions faire. Ainsi, j'ai pris en charge la partie de résolution de sudoku, qui est plus facile à appréhender que les autres. De plus, comme cette partie comportait une dimension de sauvegarde dans des fichiers, on m'a aussi confié les interactions du réseau de neurones avec l'extérieur du programme : sauvegarde des poids, récupération d'images d'entraînement...

Ensuite, le développement du code a commencé : la résolution du sudoku s'est faite sans beaucoup de difficulté, étant donné que nous avions déjà étudié l'algorithme l'année dernière. Cependant, les contraintes sont arrivées sur le système de sauvegarde des poids du réseau de neurones.
En effet, il me fallait enregistrer des arrays à deux dimensions, mais je ne savais pas comment ils fonctionnaient en C ni comment les implémenter autrement. Je n'étais pas non plus à l'aise avec les pointeurs et l'allocation mémoire dynamique alors que la solution à mes problèmes y résidait.. Il m'a fallu du temps et beaucoup d'erreurs pour arriver à un résultat satisfaisant, mais j'ai finalement réussi ! Je suis très content d'être parvenu à faire ce que j'étais censé faire. De plus, cela m'a permis d'apprendre des choses vis-à-vis du langage, en particulier des pointeurs, et je suis certain que je n'en ai pas encore fini d'apprendre !

Enfin, le projet n'est pas fini et de nouvelles difficultés vont surgir sur notre route, mais je suis confiant en notre groupe. Je suis certain que nous arriverons à faire un OCR dont nous serons fiers !

2.4.4 Paul

Je me suis occupé du réseau de neurones. Au départ, je me suis beaucoup renseigné sur ce qu'est un réseau de neurones en regardant différentes vidéos et différents sites. Le problème est que, malgré avoir intégré le principe de forward propagation et backward propagation, je n'arrivais à trouver une bonne manière à l'implémenter jusqu'à je trouve une autre vidéo de Nicolai Høirup Nielsen expliquant comment construire complétement un réseau neuronal pour une porte ou-exclusif. J'ai

donc pu enfin créer le réseau qui est constitué uniquement de tableaux de doubles. Ainsi, cela m'a orienté pour la prochaine version qui sera constituée de structures neurone composé d'une valeur, d'un tableau de poids et d'un biais.

Ensuite, il a fallu entraîner notre IA lorsque le système de sauvegarde fut prêt. Nous l'avons entraîné environ 500 fois et il a obtenu des résultats plutôt solides (précision > 95 %). Cependant, notre sauvegarde s'est retrouvée corrompue. Une enquête interne est toujours en cours pour déterminer la cause.

Enfin, il a fallu créer le makefile. Ce dernier m'a pris 7 heures pour le créer à cause du problème de dépendance des fichiers. Il me manquait seulement un paramètre lors de la compilation...

3 Planning

3.1 Avance

Au niveau de l'IA final, nous avons commencé sa conception : les fichiers sont en places, les idées sont présentes et la recherche a été faites. Nous avons également commencé l'écriture du code source. Cependant, on est encore au début.

Pour le traitement d'image, le pretraitement a bien été avancé mais nous n'avons pas une grande avance pour la dernière soutenance.

3.2 Retard

Au niveau de l'IA et de la sauvegarde, tout les objectifs ont été atteints.

Concernant le traitement d'image, la détection de la grille et la séparation de chaque cases de celle-ci ne sont pas encore fonctionnelles et doivent etre retravaillées.

4 Nos attentes pour la soutenance finale

4.0.1 Réseau de neurones

Pour notre réseau de neurones, nous voulons implémenter un système de couches pour les noeuds intermédiaires, la fonction softmax pour la couche de sortie, remplacer la fonction d'activation sigmoid par relu et mettre en place le système adaboost. Notre objectif est d'obtenir un réseau rapide, puissant et flexible.

En ce qui concerne la sauvegarde des poids, il sera possible qu'elle change de format d'ici à la soutenance finale pour passer à un système de fuchiers multiples. Cependant, avec l'architecture de réseau de neurone que nous avons à cette date, il ne paraît pas difficile d'étendre la sauvegarde à un réseau plus étendu.

4.0.2 Traitement d'image

Pour le traitment d'image nous voudrions régler les problèmes actuels dans un premier temps puis se concentrer sur les prochaines taches. Nous devons revoir certaines choses sur le prétraitement et finir la détection des cases et de la grille. Nous nous concentrerons ensuite sur la rotation automatique et un système de sauvegarde de l'image avec la solution sera ensuite a faire pour terminer le projet. Nous pensons pouvoir y arriver a temps en gardant le rythme de travail que nous avons actuellement.