Rapport de soutenance intermédiaire

Lisa BOURLIER - Gwennan JARNO
Léo LOPES - Samuel PERRIER

Novembre 2022

Table des matières

1 Introduction **4**
1.1 Gwennan . . . 5
1.2 Lisa . . . 5
1.3 Léo . . . 5
1.4 Samuel . . . 6

2 Rotation manuelle **7**

3 Détection de grille **7**
3.1 Transformée de Hough . . . 7
3.2 Difficultés rencontrées . . . 9
3.3 Séparation des cases . . . 9

4 Chargement de l'image **10**

5 Suppression des couleurs **10**
5.1 Création d'un grayscale . . . 10
5.2 Création d'une représentation binaire de l'image . . . 10
5.2.1 La binarisation . . . 11
5.2.2 La méthode Otsu . . . 11
5.2.3 La méthode Sauvola . . . 12
5.3 Conclusion sur la binarisation . . . 13

6 Résolution de la grille **14**
6.1 Le solver . . . 14
6.2 Conclusion sur le solver . . . 15
6.3 Sauvegarde de la grille résolue . . . 16
6.4 Conclusion sauvegarde de la grille . . . 16

7 Réseau de neurones **18**
7.1 Implémentation . . . 18
7.2 Sauvegarde et chargement . . . 19

7.3 Dataset 20

8 Ressentis des membres **21**

8.1 Gwennan 21

8.2 Lisa 21

8.3 Léo 22

8.4 Samuel 22

9 Conclusion **23**

1 Introduction

Après un projet S2 tout aussi rythmé qu'enrichissant par le travail à fournir en équipe, il est temps d'appliquer les connaissances acquises durant l'année de SUP en débutant le projet S3 : OCR (Optical Character Recognition). Après réflexion, les quatre étudiants associés pour ce travail de projet sont :

Gwennan JARNO
Lisa BOURLIER
Léo LOPES
Samuel PERRIER – chef de projet

Chacun d'entre nous vise évidemment à donner le meilleur de soi-même afin de produire un travail de qualité, répondant aux critères du projet. Après une rapide présentation de chaque membre du groupe, ce rapport de 1ère soutenance présentera les différentes étapes du travail individuel et commun produit jusqu'au 06 Novembre, jour de rendu.

Répartition des tâches :

Tâches	Lisa	Gwennan	Léo	Samuel
Chargement d'une image		X		
Suppression des couleurs		X		
Rotation			X	
Détection de la grille et des cases			X	
Extraction des cases en bmp			X	
Réseau de neurones				X
Résolution de la grille	X			

1.1 Gwennan

Ancienne membre de la TAAG Team, je me suis souvent concentré sur l'aspect visuel de mes projets. Au lieu de dessiner, d'animer et de coder mes personnages, je me suis penchée sur un nouveau projet : le traitement d'image. Cette partie m'a toujours intéressée et c'est donc naturellement que je suis en charge du prétraitement des images ainsi que de leur chargement. Même si mes connaissances en codes peuvent parfois avoir des lacunes, je pense pouvoir les compenser avec mes connaissances en image et en filtres afin de remplir au mieux nos objectifs pour ce projet.

1.2 Lisa

Après l'expérience très enrichissante du projet AITA pendant le s2, j'étais prête à relever un nouveau défi avec l'OCR. J'apprécie particulièrement que ce soit aussi un projet de groupe, car cela permet une meilleure dynamique et de l'entraide pour avancer plus vite. Suite à la présentation du projet, j'ai choisi de réaliser le sudoku solver. Les autres parties reposent énormément sur des notions mathématiques, ce qui ne m'attirent moins. Le solver me permet de me concentrer sur la réflexion algorithmique, une compétence que j'essaye d'améliorer. J'aimerais aussi apprendre à manipuler des fichiers, je vais donc réaliser l'affichage de la grille résolue pour la soutenance finale.

1.3 Léo

Ayant travaillé à l'élaboration du projet Forvalt au S2, j'ai énormément appris sur la façon de travailler en équipe et l'importance de la bonne répartition des tâches au sein du groupe pour fournir un produit final de qualité. A l'approche du début du projet S3, je me suis intéressé aux grandes lignes qui le caractérisent et, étant attiré par les notions mathématiques, je me suis penché sur les différentes techniques pouvant être utilisées pour un logiciel de reconnaissance d'image/texte. Je suis donc rapidement tombé

sur la Transformée de Hough, une méthode mathématique qui permet de représenter des lignes (infinité de points) sous une autre forme en changeant de base de repère. C'est pour cette raison que j'ai la charge de réaliser la détection de grille/cases, en plus de la rotation manuelle et automatique de l'image uploadée.

1.4 Samuel

Ayant également participé à la création de Forvalt durant le S2, j'en suis ressorti enrichi en termes de méthode de travail mais aussi en manière de coder. Ce projet étant composé de diverses tâches toutes aussi intéressantes les unes que les autres, il a fallu que j'en choisisse une. Je me suis donc penché sur la partie du réseau de neurones. Il s'agit de quelque chose de tout nouveau pour moi, j'ai donc dû beaucoup me documenter afin d'implémenter au mieux le réseau de neurones. Lors du projet S2 j'étais chef de projet. J'ai choisi de renouveler l'expérience car cela m'aidait à m'améliorer dans le management d'une équipe.

2 Rotation manuelle

Après avoir chargé l'image et effectué le prétraitement en appliquant différents filtres, il est maintenant nécessaire de pivoter l'image dans le cas où celle-ci n'est pas droite. Comme demandé dans le cahier des charges, la rotation manuelle doit redresser l'image selon un angle saisi par l'utilisateur. Pour cela, l'algorithme mis en place récupère l'argument saisi lors de l'exécution du programme (argv[1]) et stock cette valeur dans une variable prévue à cet effet. Cette variable est ensuite utilisée dans la fonction SDL_RenderCopyEx.

De plus, j'ai décidé de développer une fonctionnalité supplémentaire qui vise à faciliter la rotation de l'image par l'utilisateur. En effet, l'algorithme mis en place prend donc en charge le redressement de l'image via les flèches directionnelles du clavier ($\leftarrow$ / $\rightarrow$). Pour chaque appui sur la touche flèche gauche l'image subit une rotation de -1 degré et 1 degré pour la flèche droite. Cela permet une meilleure précision de rotation et facilite l'utilisation chez l'utilisateur.

3 Détection de grille

3.1 Transformée de Hough

La *Transformée de Hough* est une méthode mathématique très connue dans le domaine de la reconnaissance algorithmique de formes. Elle est d'abord utilisée dans l'astrophysique ou encore l'étude de trajectoire mais, dès l'émergence de l'informatique, elle est rapidement développée dans l'analyse d'image afin de détecter des lignes ou, dans une forme plus avancée, des cercles ou ellipses. Le principe repose sur un changement de base de repère, en passant de coordonnées cartésiennes (X, Y) à des coordonnées sphériques (ρ, θ). Le but est de construire un tableau de valeurs (appelé « accumulateur » dans le cas de la Transformée de Hough) pour chaque pixel de l'image. Pour que la reconnaissance soit le plus efficace possible, l'image

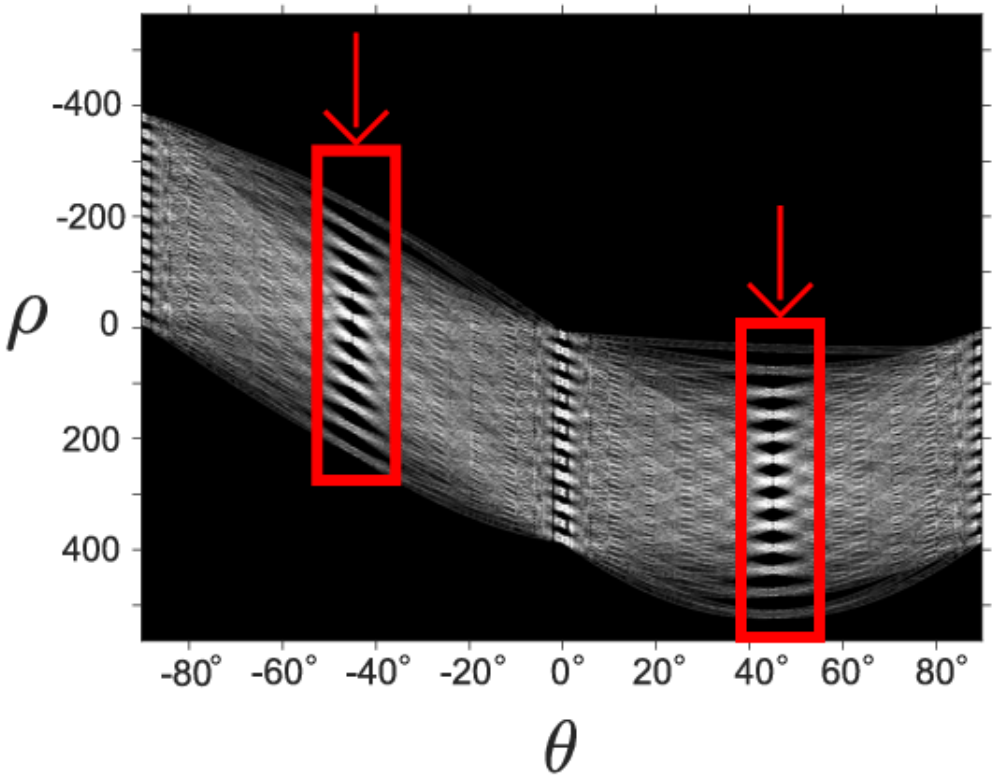
utilisée doit être prétraitée et se rapprocher au maximum d'une image à laquelle on aurait appliqué un *filtre de Canny* (traits blancs sur fond noir).

La première étape est la construction de l'accumulateur. Pour ceci, le principe est de parcourir l'image et, pour chaque pixel, déterminer les coordonnées sphériques du point actuel. Ainsi, pour chaque pixel de l'image, l'algorithme calcule un ρ en suivant la formule suivante :

$$\rho = x cos(\theta) + y sin(\theta)$$

Seulement, il faut exécuter cette formule pour une valeur de θ allant de 0 à 180 degrés (-pi/2 à pi/2 en radian). L'algorithme injecte alors 180 valeurs de ρ dans l'accumulateur, pour chaque pixel de l'image.

Le résultat de l'accumulateur obtenu est le suivant :



On retrouve alors une caractéristique de ρ en fonction de θ, dans une base sphérique.

La deuxième étape est maintenant d'extraire des « peak » de cette caractéristique. En effet, en analysant l'image précédente on peut relever

des points vers lesquels les courbes convergent (mis en évidence par les cadres rouges sur l'image). En reportant les coordonnées de ces points (appelés « peak »), cela nous donne les coordonnées d'une ligne (affine donc d'équation y = ax + b) dans la base sphérique. Il nous reste alors à exploiter ces coordonnées pour retomber sur une ligne de notre image d'origine.

3.2 Difficultés rencontrées

Pour cette partie, j'ai rencontré quelques problèmes sur lesquels j'ai dû passer beaucoup de temps. En particulier, la majeure partie de mon temps a été consacré à l'extraction des "peaks" à partir de l'accumulateur et surtout à exploiter ceux-ci. Il a été très important de choisir un "Threshold", c'est-à-dire un seuil à partir duquel on considère un peak comme étant assez important pour l'extraire, l'exploiter et reconstruire la ligne qu'il représente. Enfin, la partie la plus délicate était la reconstruction des lignes. En effet, si on veut reconstruire des lignes facilement, il faut plutôt que celles-ci soient finies, tel un segment. Il faut définir alors un point de départ et de fin de ligne à partir du peak extrait. Compte tenu de ces difficultés, mon code est une dérivée de la transformée de Hough. En effet, les bases sont les mêmes (parcours pixel par pixel, construction d'une matrice..) mais une fois les lignes extraites, celles-ci sont injectées dans une nouvelle matrice (une pour l'axe horizontal et l'autre pour l'axe vertical).

3.3 Séparation des cases

Grâce a l'implémentation de ma dérivée de la transformée de Hough, la séparation de cases se fait directement à partir des matrices des lignes détectées. Une grille de Sudoku étant une grille 9x9, il suffit alors d'utiliser les lignes pour en extraire les cases de la grille, à chaque intersection de ligne formant une case. Une fois cette étape faite, le but est alors de générer une image de chaque case, contenant notamment un chiffre, pour l'injecter ensuite dans le réseau de neurones.

4 Chargement de l'image

Partie assez simple à première vue, il faut tout de même s'en méfier. En effet, pour load une image et l'afficher, il ne suffit pas d'écrire la simple commande "load". Nous devons aussi nous soucier de comment l'image va apparaître à l'écran. Sans convertir l'image au bon format, nous risquons d'obtenir un décalage de pixels, ce qui est problématique lorsque nous devons traiter les couleurs ensemble. Mais sinon, honnêtement, cette fonction ne pose pas de difficulté première.

5 Suppression des couleurs

5.1 Création d'un grayscale

Etape fondamentale du traitement d'image, cette fonction avait déjà été explorée en TP et il m'a donc été assez facile de la réappliquer pour ce projet. Elle consiste à prendre chacune des trois valeurs présentes dans le pixel (le rouge, le vert et le bleu) et de manipuler ces valeurs afin de former un pixel gris. Tout cela est assez simple grâce à cette formule :
Average = 0,3*pixel_rouge+0,59*pixel_vert+0,11*pixel_bleu
Après, il nous suffit de remplacer chacune des valeurs initiales par cet average et le tour est joué.
Comme vous pouvez le deviner, je n'ai pas rencontré de difficulté particulière ni à comprendre ni à appliquer cette formule.

5.2 Création d'une représentation binaire de l'image

Ne sachant pas s'il fallait spécifiquement ou non rendre une image en noir et blanc ou une image en teintes de gris, j'ai préféré m'avancer sur la binarisation de mon image.

5.2.1 La binarisation

La binarisation, qui permet d'avoir une image avec seulement deux types de valeurs pour ses pixels, est assez simple à mettre en place. Cependant , il ne faut pas se contenter de choisir une valeur médiane de façon arbitraire et de dire que tous les pixels en dessous de cette valeur se transforment en blanc, et tous ceux au dessus, en noir. Cela est facile à faire mais n'est pas fiable du tout du fait que toutes les images ont des teintes de pixels différentes qui peuvent ainsi échapper à cette méthode. Il y a deux méthodes possibles pour la binarisation des images : celle à seuil et celle par segmentation. Celle par segmentation consiste à diviser l'image en zone ou en région afin de pouvoir séparer les éléments. La binarisation par seuillage consiste quant à elle à définir une valeur seuil qui permettrait ainsi le choix des pixels noirs ou blancs. Cette dernière méthode ressemble à mon idée initiale de mettre une valeur pivot afin de définir quelles couleurs appliquer. C'est donc pour cette raison que j'ai choisi de faire une binarisation par seuillage plutôt que par segmentation.

Il existe aussi différentes méthodes de seuillage comme par exemple celle d'Otsu ou de Sauvola. Ces deux méthodes sont celles qui reviennent le plus souvent dans les recherches pour binariser des images.

5.2.2 La méthode Otsu

Dans la méthode d'Otsu, qui est une méthode de seuillage global, le seuil qui minimise la variance intra-classe est recherché à partir de tous les seuillages possibles :

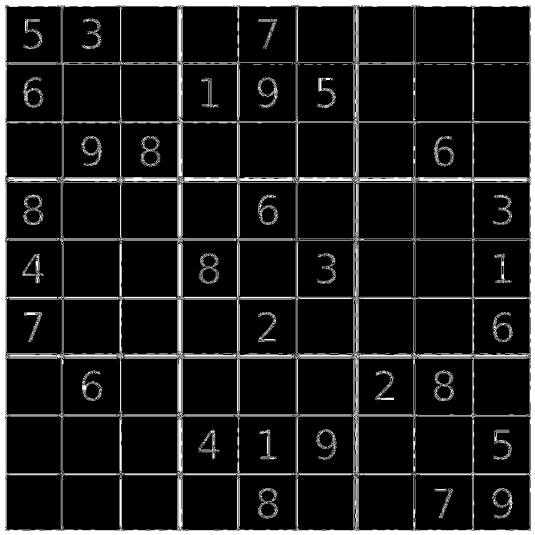
$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

Les poids ω_i représentent la probabilité d'être dans la ième classe (sachant que chacune des ces dernières est séparée par un seuil t). De plus, les σ_i^2 sont les variances de ces classes. Ainsi, avec la méthode Otsu, minimiser la variance intra-classe revient à maximiser la variance inter-classe :

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2$$

Cette dernière est exprimée en termes des probabilités de classe ω_i et des moyennes de classes μ_i qui à leur tour peuvent être mises à jour itérativement. Cette idée conduit à un algorithme concluant mais elle rencontre des problèmes lorsque le document est mal éclairé. Elle n'est donc pas efficace à 100%.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



Cette méthode m'a aussi été déconseillée et, à la place, j'ai entendu beaucoup de bien de la méthode Sauvola.

5.2.3 La méthode Sauvola

Contrairement à la méthode Otsu, cette dernière est une méthode de seuillage local. Avec cette méthode, le seuil T pour chaque pixel de l'image est donnée par :

$$T(x,y) = mean(x,y) \times \left[1 + k \times \left(\frac{s(x,y)}{R} - 1\right)\right]$$

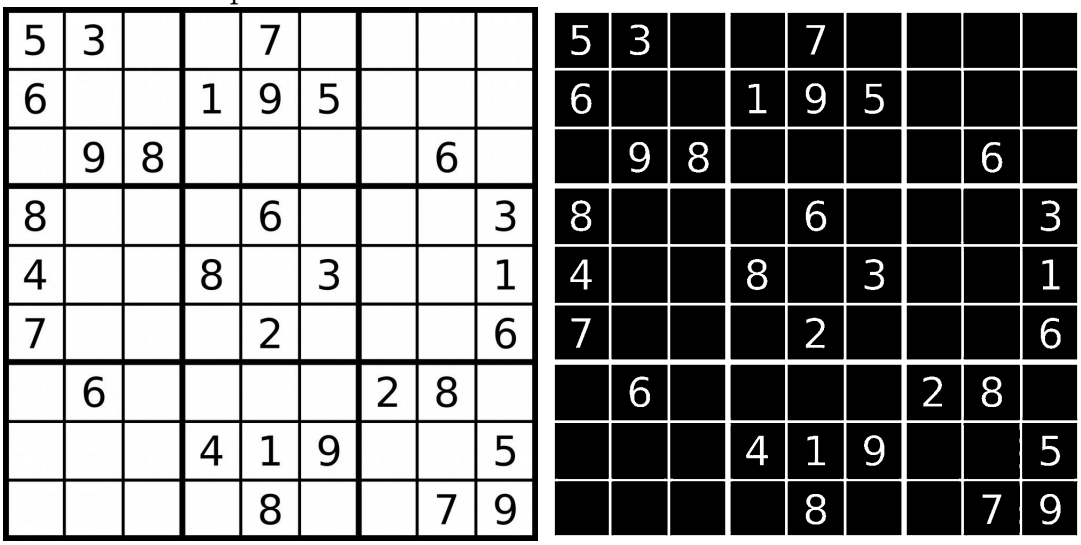
avec :

- R qui représente la valeur maximale de l'écart-type dans l'image en niveau de gris. Il est conseillé de mettre R à 128.
- k qui est un paramètre qui prend une valeur positive dans l'intervalle [0.2, 0.5].

- $mean(x, y)$ qui représente la matrice des moyennes locales pour chaque pixel de l'image.
- $s(x, y)$ qui représente la matrice des écarts-types locaux pour chaque pixel de l'image.

Pour déterminer le seuil T correspondant à chaque pixel de l'image, il est donc nécessaire de calculer la matrice des moyennes locales de l'image et la matrice des écarts-types locaux de l'image. Cette méthode est donc plus favorable au traitement d'image que la méthode Otsu car, au lieu de traiter l'image dans la globalité, elle la traite en plusieurs morceaux. Ainsi elle est plus efficace avec les images présentant des variations de couleurs.

Voici un exemple :



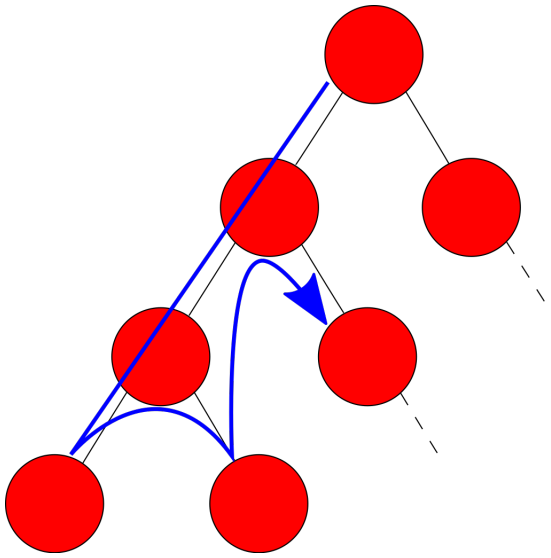
5.3 Conclusion sur la binarisation

J'ai trouvé cette partie plus complexe et difficile. Les formules ne sont pas vraiment claires à mon goût et surtout je me suis inspirée de plusieurs exemples en tout genre ce qui m'a parfois influencée dans du code erroné. Il est donc avant tout important de comprendre clairement une formule et de la maîtriser avant de vouloir directement coder.

6 Résolution de la grille

6.1 Le solver

Une fois la grille numérisée et traitée par le réseau de neurones, il faut la résoudre. C'est la tâche du solveur, qui prend en charge un fichier texte avec la grille à résoudre, et crée un nouveau fichier texte .result avec la grille résolue. Il faut d'abord se concentrer sur le sudoku solver.
Pour cela, un algorithme très utilisé et déjà étudié lors des TP C# est le backtracking. Il permet de tester systématiquement l'ensemble des affectations potentielles à un problème donné de manière beaucoup plus optimisé qu'un simple test itératif. Le principe consiste à sélectionner une variable du problème, et pour chaque affectation possible de cette variable, à tester récursivement si une solution valide peut-être construite à partir de cette affectation partielle. Si aucune solution n'est trouvée, la méthode abandonne et revient sur les affectations qui auraient été faites précédemment.



Dans le cas d'un sudoku solver, la solution valide recherchée est une grille complétée par des chiffres allant de 1 à 9 sans qu'il n'y ait d'itération sur la même ligne, colonne ou section 3x3. L'affectation partielle est le positionnement d'une valeur sur une case. Le problème peut être modélisé par un arbre général complet de profondeur 9, la racine étant la grille vide, et chaque case un nœud. Il y a donc 9 fils par nœud pour les valeurs possibles allant de 1 à 9, et à chaque étape, on teste si la valeur du fils rend la grille

valide. Si c'est le cas, on avance d'un nœud et on vérifie les nœuds suivants, sinon l'algorithme revient en arrière et teste un autre fils.
Il faut donc commencer par écrire une fonction is_valid qui permet de vérifier si l'insertion d'une valeur à une position respecte les conditions. Celles-ci sont l'unique présence d'un chiffre sur la même ligne, la même colonne et la même section de 3x3 cases. A la différence de la fonction implémentée lors du TP C#, cette fois-ci on ne parcourt pas l'intégralité de la grille, seulement la ligne, colonne et cases 3x3 concernées. Il faut aussi noter que cette fonction teste la position potentielle d'une valeur sans modifications de la grille, qui arrivent plus tard. Elle retourne un 0 ou 1 selon si la valeur respecte les conditions.
Maintenant, il faut écrire la fonction récursive qui va mettre en place le backtracking. La condition d'arrêt est lorsque la valeur de l'indice de la ligne atteint 9, on sait alors que toute la grille a été remplie et chaque case respecte les conditions. Tout d'abord, la grille est remplie de 0 sur les cases à compléter, donc il faut traiter ces cases, et non celles déjà remplies de base. Pour chaque case, on va appeler la fonction is_valid avec les valeurs 1 à 9. Si une des valeurs renvoie un résultat positif, on place la valeur sur la case, et on appelle la fonction récursive sur la case d'après. Si on ne trouve aucune valeur valide pour la case suivante, on revient en arrière et on remet un zéro sur la case complétée puis on passe à la valeur suivante. Cet algorithme permet donc de trouver une solution au sudoku le plus rapidement possible. Il suffit de faire une fonction chapeau appelant la fonction récursive de backtracking, et le solver est terminé.

6.2 Conclusion sur le solver

Cette partie ne m'a pas posé beaucoup de difficultés car j'étais déjà familière avec le principe du backtracking grâce au tp de l'année dernière. La documentation sur ce sujet est également très fournie et variée, je n'ai pas eu de mal à trouver des vidéos explicatives. Le plus dur a été l'implémentation en C, avec la manipulation d'un char array à deux dimensions.

6.3 Sauvegarde de la grille résolue

Après avoir résolu la grille, il faut la sauvegarder sur un nouveau fichier. Cette partie ainsi que la récupération de la grille du fichier texte ont été effectué après l'implémentation du solver. Il a donc fallu convertir les informations du fichier en un double char array pouvant être passé en paramètre du solveur. Tout d'abord, pour ouvrir le fichier texte, on utilise la fonction fopen de la librairie stdio.h et un pointeur de fichier. Puis, après avoir initialisé un double char array avec des zéros, on parcourt chaque caractère du fichier grâce à fgets. Il ne faut pas prendre en compte les zéros représentés par des points et les espaces inutiles de la présentation du fichier texte. On obtient ainsi la grille à résoudre sous le format voulu, et on peut la résoudre avec l'algorithme solveur.
Maintenant se pose le problème inverse, il faut convertir un double char array en présentation texte. Cela a été facilement résolu avec des boucles for et des retours à la ligne. Le vrai défi de la sauvegarde a été de créer un fichier du même nom que celui fourni avec .result à la fin. En effet, la fonction strcat de la librairie string.h ne peut pas être utilisée car elle concatène la deuxième chaîne directement dans la première, or le nom du fichier fourni ne dispose pas de l'espace supplémentaire. Il faut donc déterminer la taille du nom du fichier, puis créer un char array de cette taille additionné avec 7 et enfin utiliser strcat sur l'array de bonne taille et .result. Cette étape réalisée, il suffit de créer un nouveau fichier ayant les droits d'écriture avec fopen et le nouveau nom du fichier. On fprintf la grille résolue sous le bon format, puis on closef le fichier et la sauvegarde est terminée.

6.4 Conclusion sauvegarde de la grille

Cette partie m'a posé plus de difficultés, car je n'avais jamais manipulé de fichiers en C. Surtout, je n'avais pas anticipé que la grille fournie ne serait pas un double char array, ce que j'ai utilisé pour tester mon solver. Il m'a donc fallu convertir les formats en plus d'ouvrir les fichiers, ce qui m'a pris beaucoup de temps. Pour la manipulation de fichiers, j'ai trouvé

de la documentation précise assez facilement ce qui m'a permis de comprendre les bases et l'utilisation de pointeurs dans les fichiers. Bien que plus chronophage, cette partie était en général très intéressante.

7 Réseau de neurones

Le but de cette partie est de reconnaître un chiffre sur une image préalablement découpée, redimensionnée et traitée avec un réseau de neurones qui doit être entraîné. Je me suis donc aidé du site neuralnetworksanddeeplearning.com mais également des excellentes vidéos réalisées par 3Blue1Brown qui réalise également la reconnaissance de chiffre.

7.1 Implémentation

Pour implémenter le réseau de neurones, j'ai donc créé plusieurs structures avec chacune ses différentes données afin de les manipuler plus facilement. Il y a donc le neurone qui contient son niveau d'activation, son biais, son niveau d'erreur, et ses différents poids ; le layer qui contient les neurones ; et le réseau qui contient le nombre d'entrée, le nombre de neurones pour les couches intermédiaires, le nombre de neurones pour la couche finale ainsi que les différents layers.
Dans notre réseau de neurones il y a donc plusieurs layer qui contiennent eux mêmes plusieurs neurones. Le nombre de neurones pour le layer d'entrée est fixe : il s'agit du nombre d'entrée. Le nombre de layers pour la couche intermédiaire ainsi que le nombre de neurones de ces layers peuvent varier. Enfin, le layer de sortie contient lui aussi un nombre de neurones fixe, les valeurs d'activation de ces neurones nous ressortent la probabilité que ça soit telle ou telle valeur.

Pour cette soutenance intermédiaire, j'ai donc implémenté un XOR. Pour cela, le réseau de neurones contient donc 1 layer d'entrée, 1 layer pour la couche intermédiaire et 1 layer de sortie. Le layer d'entrée contient deux neurones qui correspondent aux deux chiffres de l'opération XOR. Le layer de la couche intermédiaire contient 5 neurones car c'est avec cela que j'ai obtenu les meilleurs résultats. Enfin, le layer de sortie contient deux neurones qui correspondent soit à 0 soit à 1.
Pour relier tout ces différents éléments entre eux, j'ai réalisé plusieurs fonctions, la première étant celle de propagation en avant. Celle-ci permet de

donner au réseau une entrée et d'effectuer les différents calculs pour calculer le nouveau niveau d'activation des différents neurones. Pour effectuer ces calculs il suffit de multiplier le niveau d'activation du neurone par le poids reliant ce neurone avec le neurone précédent est d'appliquer soit la fonction ReLU (pour toutes excepté celle de sortie) soit la Softmax (pour la couche de sortie) avec les formules suivantes :
ReLU (pour *Rectified Linear Unit*) :

$$f(x) = max(x, 0)$$

Softmax :

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Pour entraîner ce réseau, j'ai codé deux fonctions : rétro-propagation et la descente de gradient. La fonction de rétro-propagation permet de calculer le niveau d'erreur du neurone, pour cela il suffit de partir du niveau de sortie et de calculer la différence la valeur d'activation du neurone avec ce qu'elle aurait du être et d'appliquer la fonction ReLU'. Ensuite, on applique ce procédé sur l'avant dernier layer, puis celui d'avant, etc.
ReLU' :

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

Une fois ces nouveaux niveau d'erreurs calculés, on applique la méthode de descente de gradient. Il s'agit d'un algorithme d'optimisation qui calcule le nouveau poids et bais du neurones en multipliant le taux d'erreur du neurone par son activation et par une valeur définie. Pour entraîner le réseau, il suffit donc d'appliquer ces deux fonctions avec des entrées dont on lui spécifie les sorties voulu un nombre de fois assez grand pour obtenir des résultats convenables.

7.2 Sauvegarde et chargement

Lorsque le réseau est entraîné, il est intéressant d'enregistrer les poids et biais des différentes neurones pour éviter d'avoir à effectuer l'apprentissage

une nouvelle fois. J'ai donc codé une fonction crée un fichier contenant ces différents éléments. Le format de ce fichier est comme suit :

Nombre de neurones de la couche d'entrée
Nombre de neurones des layers de la couche intermédiaires
Nombre de neurones de la couche de sortie
Nombre total de layers de toutes les couches
Nombre de neurones du layer n°0
Valeur du biais du neurone n°0 du layer n°0
Nombre de poids du neurone n°0 du layer n°0
Valeur du poids du poids n°0 neurone n°0 du layer n°0
...

Une autre fonction permet de faire l'inverse : elle lit le fichier et initialise le réseau avec les différentes valeurs.

7.3 Dataset

Pour entraîner le réseau de neurones, il est nécessaire de posséder des images contenant des chiffres. J'ai donc choisi d'utiliser la base de donnée MNIST qui regroupe 60 000 images déjà labellisé pour pouvoir entraîner notre réseau.

8 Ressentis des membres

8.1 Gwennan

J'ai eu l'impression de moins bien gérer ce début de projet. Au départ, j'étais trop en avance, puis j'ai entendu des informations contraires, ce qui m'a fait douter et vouloir tout reprendre avant de me rendre compte que ma version initiale était bonne. J'ai donc eu l'impression de naviguer en tous sens et j'espère que ce que j'ai fait remplit l'objectif de ce rendu.
Au niveau du code, on peut considérer le grayscale comme une fonction très simple du faite que nous l'avons déjà vu en tp. C'est pour cela que j'ai voulu contrebalancer en implémentant un algorithme otsu et un autre sauvola. Les deux sont présents dans mon code dans la cas où j'aurais besoin de changement ou d'ajustement par la suite.
De plus, je me suis énormément renseignée sur les filtres en avance et j'espère que ceux que j'ai déjà implémentés (celui de contraste et le median filter) me seront utiles pour la suite. Il m'a été assez difficile de trouver le principe de ces algorithmes parce qu'à chaque fois je tombais simplement sur leur appel de fonction en python (qui est en une ligne), ce qui ne m'éclairait pas sur leur construction générale.

8.2 Lisa

Bien que j'ai encore eu du mal à gérer mon temps et planifier en avance, j'ai réussi à remplir tous mes objectifs pour cette première soutenance. J'étais assez appréhensive sur mes compétences en programmation pour mener à bien mes tâches, bien que le solveur soit une des parties les moins compliquées. Particulièrement sur le fait que je doive gérer mon architecture de dossier en C, avec mon propre makefile et fichiers headers.
J'ai beaucoup aimé apprendre plus en profondeur la gestion de fichiers en C, car ces compétences seront utiles pour de futurs projets. Un autre avantage acquis est la maîtrise de Github, après une expérience plutôt négative lors du projet s2.

8.3 Léo

Même si je me suis énormément renseigné sur mon sujet et ai lu beaucoup de documentation, je pense avoir sous estimé la difficulté de ma partie. En effet, la détection de grille me paraissait pas aussi complexe et j'ai du y consacrer beaucoup de temps tout en devant revoir plusieurs fois mon implémentation pour mieux gérer l'extraction des cases de la grille.
J'ai aimé découvrir les différents principes mathématiques autour de ma partie et j'estime que ceux-ci me serviront par la suite.

8.4 Samuel

Je suis très satisfait de mon travail et de mon implémentation même si cela n'a pas était facile. J'ai trouvé assez difficile de trouver les idées importantes de toutes la documentation et de comprendre les différentes formules mathématique. Je pense que l'utilisation du réseau de neurones avec des images ne sera pas si difficile car la totalité de la structure a été réalisé.

9 Conclusion

Nous sommes tous assez ravis du travail accompli jusque là même si nous avons tous pour la plupart un peu mal géré le projet. Nous espérons réellement rendre pour la soutenance finale un OCR accompli et ferons tout notre possible pour apprendre de ce premier rendu.