

# Rapport de Projet OCR n°1

Sudoku Solver

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	

Membres du groupe 9 :

Charlotte Buat  
Camille Nguyen (*Cheffe de projet*)  
Florine Kieraga  
Sami Carret

# Sommaire

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Répartition des tâches accomplies pour la première soutenance</b>	<b>3</b>
<b>3</b>	<b>Etat d'avancement du projet</b>	<b>3</b>
<b>4</b>	<b>Aspects techniques</b>	<b>4</b>
4.1	Chargement d'une image . . . . .	4
4.2	Suppressions des couleurs . . . . .	4
4.2.1	Niveau de gris . . . . .	4
4.2.2	Noir et blanc . . . . .	5
4.3	Prétaitemet . . . . .	7
4.3.1	Rotation manuelle . . . . .	8
4.4	Détection de la grille et des cases . . . . .	8
4.4.1	Détection et isolation de la plus grande composante connexe .	8
4.4.2	Détection des lignes à l'aide de la transformé de Hough . . . . .	10
4.4.3	Isolation de la grille . . . . .	11
4.4.4	Isolation des cases . . . . .	12
4.5	Réseau de neurones . . . . .	12
4.5.1	Forward propagation . . . . .	13
4.5.2	Back propagation . . . . .	13
4.6	Résolution de la grille . . . . .	14
<b>5</b>	<b>Ressenti des membres du groupe</b>	<b>15</b>
5.1	Charlotte Buat . . . . .	15
5.2	Camille Nguyen . . . . .	15
5.3	Florine Kieraga . . . . .	15
5.4	Sami Carret . . . . .	16
<b>6</b>	<b>Conclusion</b>	<b>17</b>
6.1	Bibliographie . . . . .	18

## 1 Introduction

Dans ce rapport, nous détaillerons les différentes tâches que nous avons accomplies entre la formation de notre groupe et la première soutenance de projet OCR (*Optical Character Recognition*). Pour rappel, le projet du semestre 3 est un projet qui a pour objectif de réaliser un logiciel de type OCR capable de résoudre automatiquement une grille de sudoku. Celui-ci prend une image représentant une grille de sudoku en entrée et affiche en sortie la grille résolue. Pour ce faire, il est nécessaire de prendre en compte les différentes contraintes imposées par la photo ayant été sélectionnée (par exemple, une photo floue, penchée, en couleur... rendra le processus de prétraitement d'image plus long et fastidieux). Par ailleurs, ce logiciel est écrit en langage C.

Dans un premier temps, nous présenterons les membres qui composent notre groupe ainsi que la répartition des tâches à réaliser par l'ensemble des membres afin de concevoir le projet OCR. Dans un second temps, nous développerons l'aspect technique du projet avec les différents traitements appliqués à l'image, la détection de celle-ci ainsi que l'algorithme de résolution du sudoku. Et enfin, nous expliciterons les buts et objectifs que nous nous fixons pour la prochaine soutenance.

## 2 Répartition des tâches accomplies pour la première soutenance

Ce tableau présente les tâches accomplies par les membres du groupe entre la formation de notre groupe et la première soutenance. Les X représentent la ou les personne(s) ayant accompli cette tâche. Les s représentent la ou les personne(s) ayant aidé à l'accomplissement de cette tâche.

	Camille	Charlotte	Florine	Sami
Chargement d'image	X			
Suppression des couleurs			X	X
Prétraitement	s	X		s
Détection de la grille			X	X
Détection des cases de la grille			X	X
Récupération des chiffres			X	
Présence de chiffre dans les cases		X		
Réseau de neurones	X	X		
Reconstruction et résolution de la grille		X		
Affichage de la grille résolue (UI)	X			
Sauvegarde de la grille résolue			X	

## 3 Etat d'avancement du projet

Ce tableau présente les tâches accomplies par les membres du groupe pour la première soutenance.

	Soutenance 1
Chargement d'image	T
Suppression des couleurs	T
Prétraitement	A
Détection de la grille	T
Détection des cases de la grille	T
Récupération des chiffres	C
Présence de chiffre dans les cases	
Réseau de neurones	C
Reconstruction et résolution de la grille	C
Affichage de la grille résolue (UI)	
Sauvegarde de la grille résolue	C

## 4 Aspects techniques

### 4.1 Chargement d'une image

Afin de charger l'image qui sera utilisée pour résoudre un sudoku, nous avons créé une fonction capable de convertir n'importe quel format d'image (png, jpg, jpeg...) en fichier bmp.

Pour ce faire, nous avons utilisé la librairie SDL.h que nous utiliserons tout au long de ce projet afin de pouvoir travailler sur l'image. Pour pouvoir modifier une image il faut utiliser des SDL\_Surface. Grâce au SDL\_Surface, nous pouvons accéder à la taille de l'image mais aussi à chaque pixel de celle-ci pour les lire ou les modifier.

### 4.2 Suppressions des couleurs

#### 4.2.1 Niveau de gris

Afin de pouvoir travailler sur une image, il est nécessaire de supprimer les couleurs qui la composent. En effet celles-ci rendent le travail de détection de la grille plus compliqué.

Pour supprimer les couleurs de notre image, nous avons tout d'abord transformé celle-ci en niveaux de gris. Pour créer un pixel gris, il suffit d'utiliser la même valeur pour chacune de ses composant RGB (Red, Green, Blue) (en français : Rouge, Vert, Bleu). La conversion d'un pixel de couleur à un pixel gris est donc la moyenne de ses trois composantes RGB. Une fois cette moyenne obtenue, il suffit de changer la valeur des composantes RGB par la valeur de la moyenne.

Pour calculer la moyenne des composantes RGB, nous avons utilisé la formule :

$$\text{Moyenne} = 0.3*\text{Rouge} + 0.59*\text{Vert} + 0.11*\text{Bleu}$$

Pour transformer l'image en niveau de gris, nous avons donc parcouru chaque pixel de l'image et calculé la moyenne des composantes RGB puis changé la valeur des composantes RGB du pixel par la valeur de la moyenne.

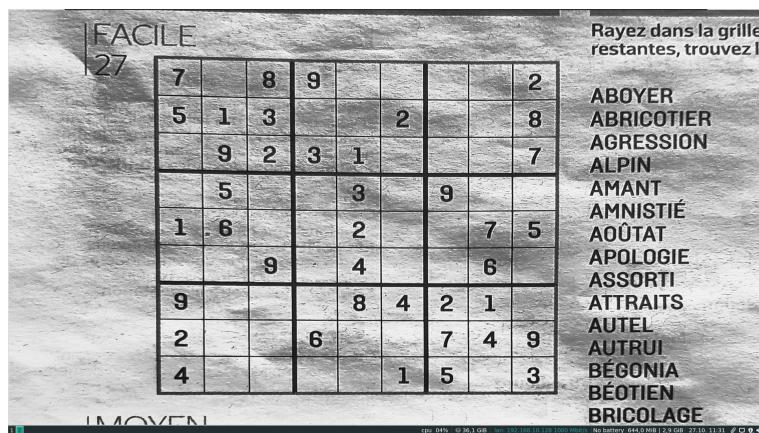


Image après conversion en niveau de gris

#### 4.2.2 Noir et blanc

L'intégration d'un algorithme permettant à une image en nuances de gris d'être passée en noir et blanc est essentielle pour permettre à notre OCR de détecter la grille, les cases et par la suite reconnaître les caractères. C'est un concept qui paraît simple mais qui peut devenir très périlleux à mettre en place.

Pour transformer une image de sudoku qui possède déjà un fond homogène ou des contrastes élevés, la tâche est assez simple, il suffit d'appliquer un seuil (nombre pour lequel chaque pixel ayant une valeur de nuance de gris supérieure au seuil deviennent blanc et les autres deviennent noir) global qui séparera écriture et fond pour permettre le traitement de l'image dans les parties suivantes. Malheureusement cette méthode a des résultats très limités pour les images ayant un fond moins homogène ou des contrastes bas ce qui est le cas pour certaines images que nous avons à traiter.

Pour pallier à ce problème, nous nous sommes rabattus sur un algorithme de seuillage local qui va trouver un seuil sur des petites fenêtres (après de nombreux tests nous avons trouvé optimal d'utiliser des fenêtres qui font 1/32 de la largeur de l'image) à travers l'image et leur applique le traitement expliqué plus tôt. Il existe de multiples méthodes de seuillage local plus ou moins efficaces.

Dans un premier temps nous avons utilisé la méthode des "local minima and maxima" qui va effectuer une opération utilisant le pixel le plus clair et le plus foncé de chaque fenêtre, la formule utilisée dans cet algorithme est très simple mais nous avions des résultats très limités en particulier sur l'image 4 et l'image 6 qui renvoyaient une image intégralement blanche.

Ensuite nous avons utilisé la méthode de Sauvola qui utilise une formule mathématique plus compliquée utilisant les écart-types locaux et la variance. Les résultats avec cette méthode étaient plus satisfaisants mais l'image 4 gardait une certaine quantité de bruit et l'image 6 apparaissait toujours intégralement blanche.

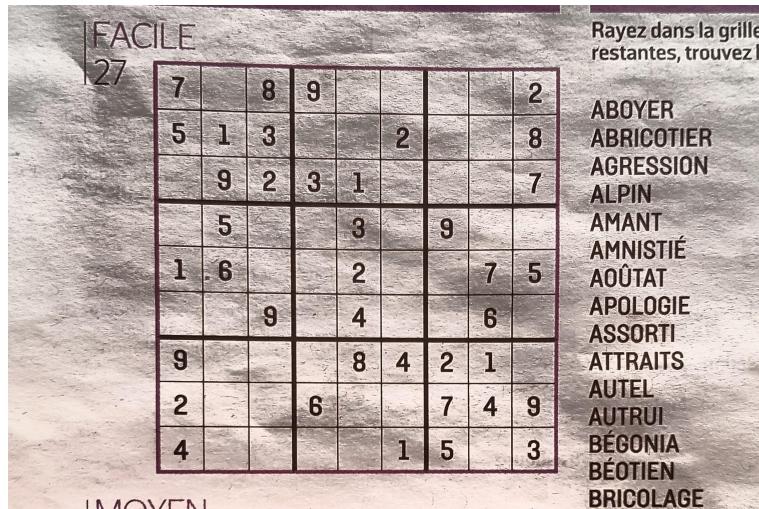


Image sans aucun traitement

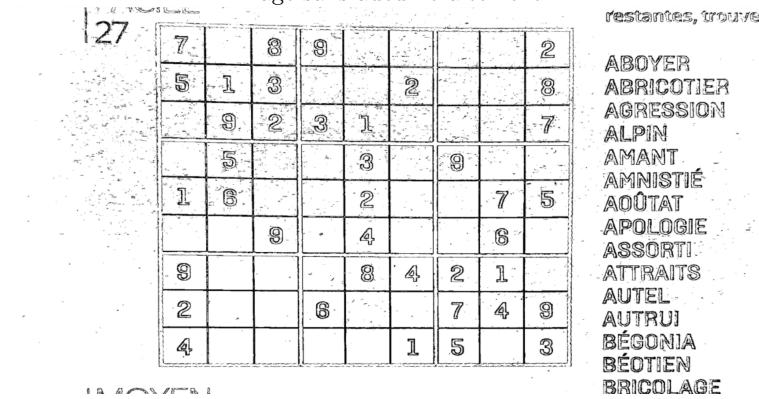


Image après suppression des couleurs et application de la méthode de Sauvola

Après de nombreuses recherches nous sommes ensuite tombés sur la méthode de Wellner qui utilise le principe d'image intégrale (l'image intégrale est une matrice dont chaque cellule a pour valeur la somme des pixels du rectangle supérieur gauche aux coordonées de cette cellules sur l'image que nous voulons traiter) et va prendre les pixels qui ont une valeur supérieur à la moyenne de chaque cellules de l'image intégrale. Cette méthode nous a permis d'avoir des résultats très concluants sur chacune des grilles peu importe le niveau de bruit ou le contraste de cette image.

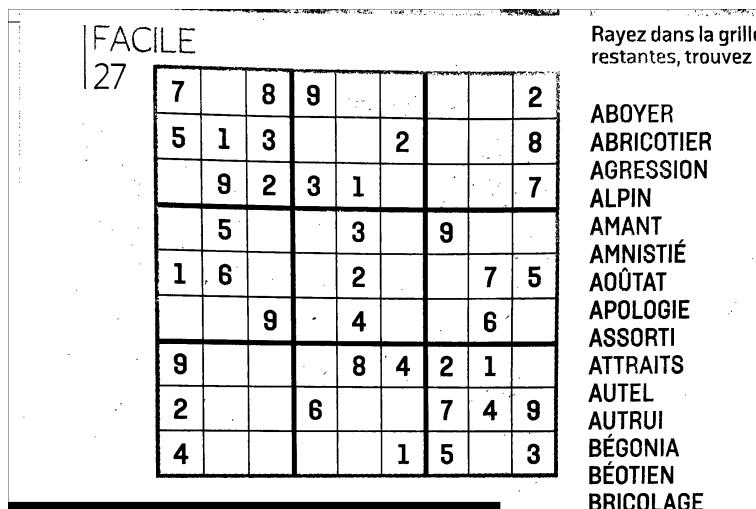


Image après suppression des couleurs et application de la méthode de Wellner

La création d'une matrice des sommes de tout les pixels de l'image est une méthode qui coûte beaucoup de mémoire, nous avons donc du avoir recours à un calloc pour permettre à l'ordinateur de supporter cette charge en mémoire.

Afin de rendre la grille de sudoku plus visible par rapport au reste de l'image, nous avons ensuite inversé tous les pixels de l'image en noir et blanc. Les pixels blancs devant donc noir et inversement pour les pixels noirs. La grille de sudoku devient alors blanche sur un fond noir.

### 4.3 Prétraitement

Pour le prétraitement des images, nous avons réalisé deux opérations sur celle ci:

- Un flou Gaussien
- Une dilatation

Le flou Gaussien consiste à faire la moyenne des pixels entourant chaque pixel, permettant d'atténuer les fortes variations de couleur et rendant la suppression du bruit plus facile.

Pour notre dilatation, nous avons crée un algorithme qui va rendre noirs tout les pixels autour de chaque pixel noir, rendant les lignes et caractères plus visibles et facile à détecter par les traitements suivants. Cela permet également de supprimer l'apparition de petits trous dans les lignes après l'application de la binarisation.

#### 4.3.1 Rotation manuelle

Pour pouvoir appliquer une rotation manuelle à nos images, nous avons utilisé une fonction très pratique de la librairie SDL 1.2 se nommant rotozoom, cette fonction permet de tourner et de zoomer une image. Pour le moment, la rotation est choisie manuellement par l'utilisateur en utilisant `./main "chemin vers l'image" "angle de rotation"`.

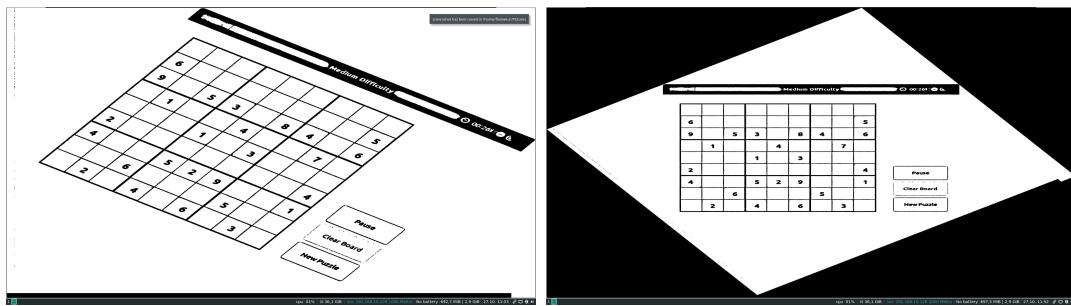


Image avant et après rotation de 35°

#### 4.4 Détection de la grille et des cases

Afin de pouvoir résoudre le sudoku sur l'image donnée par l'utilisateur, il faut que l'ordinateur détecte la grille présente sur l'image ainsi que les cases qui la compose. Pour cela, nous avons procédé à plusieurs changements sur l'image qui seront détaillé dans cette partie :

1. Détection et isolation de la plus grande composante connexe
2. Détection des lignes à l'aide de la transformée de Hough
3. Isolation de la grille
4. Isolation des cases

##### 4.4.1 Détection et isolation de la plus grande composante connexe

L'image donnée par l'utilisateur n'étant pas seulement composée de la grille de sudoku, il est nécessaire d'isoler celle-ci pour pouvoir ensuite détecter les différentes cases qui la composent.

Pour isoler la grille, nous avons tout d'abord utilisé la méthode des composantes connexes. Cette méthode permet de regrouper les pixels de même couleur qui sont en contact dans l'image. Pour chaque groupe de pixels en contact et de même couleur, une couleur leur est attribué. L'image de l'utilisateur ayant suivi un prétraitement, elle est maintenant composée seulement de pixels noirs (de valeur R = 0, G = 0 et

$B = 0$ ) et de pixels blancs (de valeur  $R = 255$ ,  $G = 255$  et  $B = 255$ ). La grille de sudoku étant représentée par les pixels de couleur blanche, les zones de composantes connexes cherché seront donc les zones blanches. Nous avons utilisé l'algorithme *Hoshen-Kopelman* qui est basé sur l'algorithme *Union-Find*.

L'algorithme *Hoshen-Kopelman* se déroule de la façon suivante :

- Pour chaque pixel de la grille, si le pixel rencontré est blanc alors nous vérifions :
  - Si le pixel n'a pas de pixels voisins blancs alors un nouveau "label" (une nouvelle couleur de cluster) lui est attribuée
  - Si le pixel a des voisins blancs alors sa nouvelle couleur est décidée par rapport à la couleur de ses voisins à l'aide de l'algorithme *Union-Find*

L'algorithme *Union-Find* consiste en :

- Find (trouver) : déterminer si deux pixels sont de même classe équivalente, si les deux pixels sont de classe équivalente alors le pixel dont nous cherchions à déterminer la couleur va prendre la couleur du pixel de classe équivalente.
- Union (unir) : réunir les pixels équivalents



Image après détection des composantes connexes

Nous avons considéré que la grille de sudoku était la plus grande composante connexe de l'image. Nous avons donc retiré toutes les autres composantes pour ne conserver que la plus grande composante, la grille de sudoku. Celle-ci devient donc la seule composante blanche sur le fond noir de l'image.

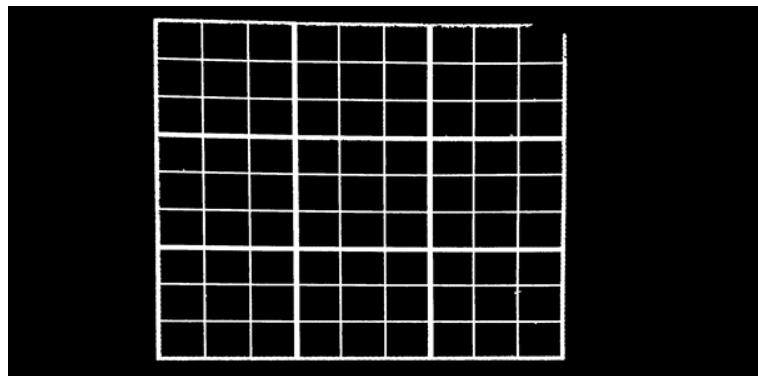


Image après isolation de la grille

#### 4.4.2 Détection des lignes à l'aide de la transformé de Hough

Une fois la grille isolé, il est devenu plus facile de détecter les lignes horizontales et verticales qui la composent. Pour détecter ces lignes nous avons utilisé la transformée de Hough. Celle-ci consiste en l'utilisation de  $\rho$  et  $\theta$ . Chaque ligne étant un vecteur de coordonnées  $\theta$  et  $\rho$ :

- $\rho$  : la distance du segment perpendiculaire à la droite d'angle  $\theta$  et passant par l'origine du repère
- $\theta$  : l'angle

Chaque ligne vérifie :  $\rho = x \cos(\theta) + y \sin(\theta)$

Nous avons donc crée un accumulateur qui va enregistrer et incrémenter les valeurs de  $\theta$  ( $\theta$  étant entre 0 et 180 dégrées) et  $\rho$  en les calculant pour chaque pixel de notre image. Nous avons ensuite tracé les lignes donc l'angle  $\theta$  était de valeur 0 ou 90. 0 étant l'angle pour les lignes horizontales et 90 étant l'angle pour les lignes verticales.

Pour pouvoir exploiter ses lignes plus tard, nous les avons tracée de différentes couleurs :

- Vert représentant les lignes verticales
- Bleu représentant les lignes horizontales
- Rouge représentant les intersections entre les lignes verticales et les lignes horizontales

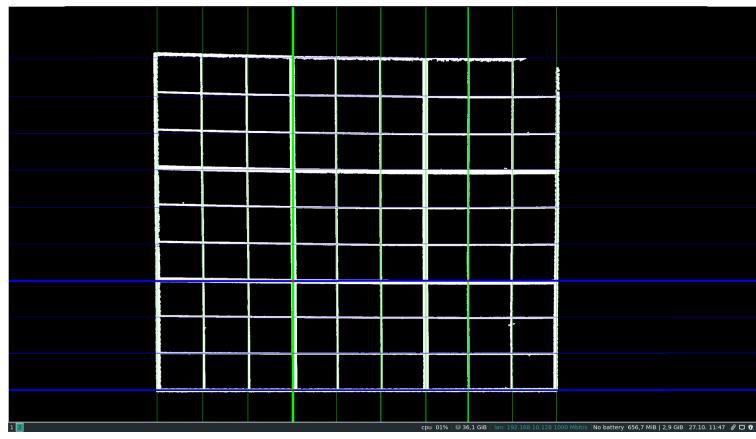


Image après détection des lignes

#### 4.4.3 Isolation de la grille

Une fois les lignes détectées la prochaine étape fut d'isoler la grille de sudoku. Pour isoler la grille de sudoku, nous avons enregistré la position ( $x,y$ ) de chaque coin de la grille grâce aux lignes déterminées précédemment. En effet, la grille étant maintenant la seule composante de notre image, toutes les lignes tracées à l'aide de la transformée de Hough représentent la grille de sudoku.

Il nous a donc suffit de partir de chaque extrémité de l'image et d'avancer jusqu'à détecter la couleur d'une ligne (vert ou bleu) ou d'intersection (rouge). Si la couleur détectée est le rouge on peut en déduire que nous avons trouvé un coin de la grille de sudoku. Si nous avons la couleur d'une ligne alors nous continuons le long de celle-ci jusqu'à rencontrer un pixel rouge dont nous allons enregistrer la position. Étant parti de chaque coin de notre image nous avons donc récupéré les quatre coins de la grille.

A l'aide des quatre coins détectés, nous avons crée une nouvelle image de la taille de la distance entre chaque coin et l'avons rempli des pixels étant à l'intérieur des quatre coins.

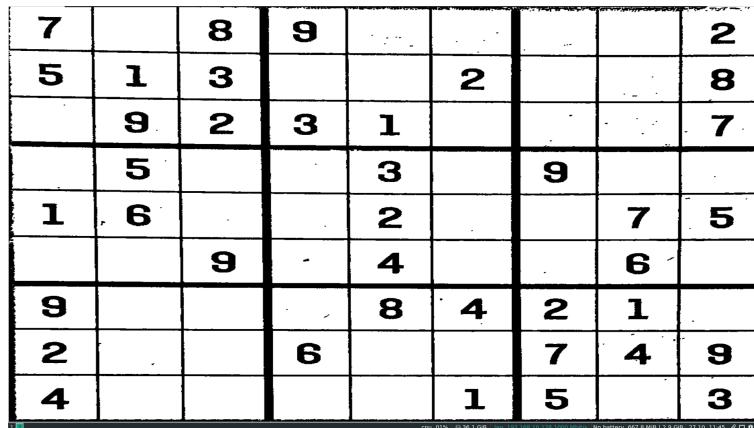


Image après détection des composantes connexes

#### 4.4.4 Isolation des cases

La dernière étape fut d'isoler chaque case de la grille dans des images différentes. Pour cela nous avons seulement découpé l'image de la grille de sudoku créée précédemment en 81 images de longueurs et largeurs identiques. En effet, une grille de sudoku étant généralement un carré composé de petits carrés, ce découpage permet d'isoler correctement chaque case.



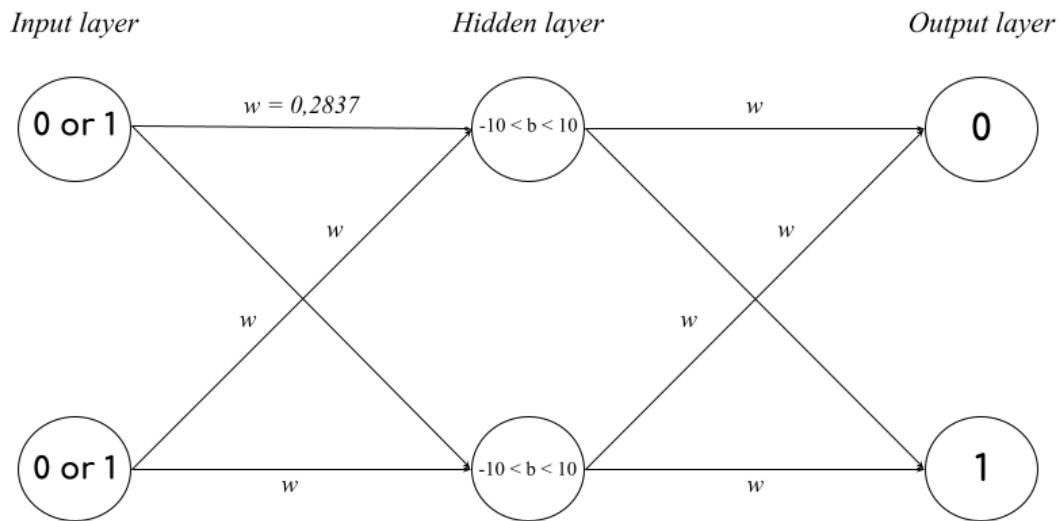
Case on (3,0) après le découpage de la grille

#### 4.5 Réseau de neurones

Pour réaliser un réseau de neurones capable d'apprendre la fonction OU EXCLUSIF, nous nous sommes beaucoup aidés de documentations et de vidéos explicatives trouvées sur internet afin de comprendre comment ce mécanisme marchait. Ainsi, nous avons créé 3 couches de neurones sous forme de listes avec des tailles variant selon le nombre de neurones qu'elles contiennent :

- La couche d'entrée contenant 2 neurones (Input layer)
- La couche cachée contenant 2 neurones (Hidden layer)
- La couche de sortie contenant 2 neurones (Output layer)

La couche d'entrée possède deux éléments - deux neurones - chacun pouvant prendre les valeurs 0 ou 1. Ces derniers sont reliés aux neurones contenus dans la couche cachée avec des liaisons possédant chacun des "poids" (weights) différents, tandis que des "biais" (bias) sont affectés aux neurones de la couche cachée : ces deux valeurs permettent de faire varier "l'importance" du neurone étudié, c'est à dire son impact sur la déduction finale du programme, ce qui déterminera alors quelle valeur sera lue à la couche de sortie.



#### 4.5.1 Forward propagation

Pour calculer les valeurs de chaque neurones de la couche cachée ainsi que ceux dans la couche output, nous utilisons une fonction d'activation sur chacun d'eux. Nous avons opté pour la fonction sigmoïde :  $f(x) = \frac{1}{1+e^{-x}}$  avec  $x = \sum_{i=0}^n weight[i] * layerprecedente[i] + bias$ , n étant le nombre de neurone du layer précédent.

#### 4.5.2 Back propagation

Après que nous soyons arrivés à un résultat après un test, nous devons procéder à une back propagation. Lors de cette propagation, nous mettons à jour les valeurs

de chaque biais et chaque poids des neurones contenus dans la couche hidden et la couche output jusqu'à ce que la valeur d'erreur finale se rapproche le plus possible de 0. Pour savoir par quelle valeur chaque élément devra être remplacé, nous utilisons les quatre formules élémentaires de la back propagation, trouvées sur le site [Neural networks and deep learning](#).

#### 4.6 Résolution de la grille

Pour la résolution de la grille, nous partons d'un fichier suivant un certain format contenant toutes les informations sur la grille de sudoku récupéré puis nous résolvons la grille et enregistrons cette grille dans un nouveau fichier en suivant le même format. Nous avons utilisé la méthode de backtracking pour construire notre fonction.

Pour ce faire, nous avons :

1. Récupéré les informations du fichier sous la forme d'un tableau en initialisant les cases vides avec des 0.
2. Utilisé une fonction récursive sur le tableau. Cette fonction va tester, pour chaque élément égal à 0 du tableau, tous les nombres de 1 à 9 en testant le premier nombre sur la première case et en testant si la grille est toujours valable puis en mettant le premier nombre sur la deuxième case et en testant si la grille est toujours valable etc. Dès que l'on arrive sur une grille non valable (plusieurs fois le même chiffre dans une grande case, une colonne ou une ligne), on termine la récursion et on revient donc dans la boucle de récursion précédente qui change la valeur de la case actuelle pour le chiffre d'après. Le programme s'arrête lorsque la grille est complètement remplie.
3. Réenregistré le tableau obtenue dans un nouveau fichier final.

## 5 Ressenti des membres du groupe

### 5.1 Charlotte Buat

Ce projet, bien que très intéressant, fut assez éprouvant mentalement, notamment du au temps entre la formation des groupes et la soutenance qui était très court et l'utilisation du langage C était une première pour moi.

La partie sur laquelle j'ai travaillé que j'ai trouvé la plus compliquée était le réseau de neurone. En effet, celle-ci nous a apporter à Camille et à moi un bon lot de difficulté.

Dans l'ensemble, je trouve que ce projet est tout de même une bonne expérience et nous apprend à nous organiser afin de finir des projets dans les temps.

### 5.2 Camille Nguyen

Pour ma part, j'ai trouvé le projet OCR très intéressant malgré le stress causé par le peu de temps que nous avons eu entre le lancement du projet et la première soutenance.

Le réseau de neurones, sur lequel j'ai travaillé avec Charlotte, fut assez compliqué à mettre en place. Pour bien comprendre le concept d'un réseau de neurones, nous avons regardé beaucoup de vidéos explicatives et lu beaucoup de documents, notamment sur les fonctions qu'il fallait appliquer aux poids et aux biais des neurones, et la syntaxe des formules mathématiques utilisées dans le backpropagation fut assez longue à déchiffrer.

### 5.3 Florine Kieraga

La première partie de ce projet OCR fut très intéressante pour moi, notamment dans la découverte des différentes méthodes de modifications de l'image ainsi que les méthodes de détection d'objets sur une image. Le temps entre la formation de groupe et la première soutenance étant très court, il nous fallut donc s'approprier les différentes notions assez rapidement.

Le plus compliqué pour moi fut la détection de la grille et des cases. En effet, plusieurs méthodes de détection peuvent être utilisé, il fut donc difficile de choisir quelle méthodes utilisé. Mais aussi, traduire les différents algorithmes en langages C, n'aillant jamais fait de C auparavant, il fallut découvrir comment celui-ci fonctionnait et traduire les différents algorithmes dans ce langage.

#### 5.4 Sami Carret

De mon côté, j'ai beaucoup appris en m'occupant de la binarisation de l'image, principalement en ce qui concerne le C : pointeurs, traitement d'image, etc. En plus de cela j'ai également pu expérimenter avec des méthodes de traitement d'image assez poussées ce qui fut très intéressant et enrichissant d'un point de vue culturel et intellectuel. J'ai également appris à travailler plus méthodiquement pour régler les erreurs auxquelles je faisais face.

J'ai beaucoup apprécié travailler sur ce projet car c'est un sujet qui m'intéresse beaucoup et dont j'aimerais potentiellement faire mon travail, même si la quantité de recherche que j'ai effectué pour trouver la bonne méthode était assez conséquente, ça reste un souvenir enrichissant pour moi et je serai heureux de retravailler sur un projet comme ça.

## 6 Conclusion

Pour conclure, nous sommes parvenus à achever les tâches qui nous étaient demandées pour cette première soutenance. Cette première partie de projet nous a permis de grandement nous instruire sur le sujet de l'OCR mais aussi de nous améliorer dans notre maîtrise du langage C, dans la manipulation d'images en C ainsi que dans l'utilisation de logiciels et de librairies tels que vim ou SDL.

Nous avons cependant trouvé assez difficile de gérer notre temps dû au court délai qui nous était imparti pour cette première soutenance et les autres projets scolaires que nous devions réaliser parallèlement au projet OCR. Malgré cela, nous pensons que notre cohésion de groupe ainsi que les réunions hebdomadaires organisées afin de suivre l'avancée de chaque membre du groupe et de planifier les tâches à suivre, fut optimale pour la progression de notre projet.

Pour la prochaine soutenance, nous prévoyons de terminer la phase de prétraitement ainsi que tous les autres éléments nécessaires à la réalisation de notre solveur de sudoku afin que celui-ci fonctionne parfaitement !

## 6.1 Bibliographie

- \* Algorithme Hoshen-Kopelman
- \* Transformée de Hough
- \* Composantes connexes
- \* Neural networks and deep learning (Utilisé pour le backpropagation)