# Rapport 2: OCR Sudoku Solver

Jules HAQUIN Yann MESSE Vanina BARAHONA UNDURRAGA

Décembre 2022

# Table des matières


**1 Introduction** **3**
- 1.1 Présentation du groupe . . . 3
  - 1.1.1 Formation du groupe . . . 3
  - 1.1.2 Ses membres . . . 4
- 1.2 Présentation du projet OCR . . . 5
- 1.3 Répartition des taches . . . 5
  - 1.3.1 Soutenance 1 . . . 5
  - 1.3.2 Soutenance final . . . 5

**2 Prétraitement** **6**
- 2.1 Contraste (Rappel) . . . 6
- 2.2 Grayscale (Rappel) . . . 6
- 2.3 Le filtre Gaussien (Rappel) . . . 6
- 2.4 La binarisation (Rappel et modification) . . . 7
- 2.5 La rotation . . . 8
  - 2.5.1 La rotation manuelle (Rappel) . . . 8
  - 2.5.2 La rotation automatique (Recherche) . . . 9
  - 2.5.3 La rotation automatique (Application) . . . 12

**3 Détection de la grille** **15**
- 3.1 Détection des lignes . . . 15
  - 3.1.1 Filtre de Sobel . . . 15
  - 3.1.2 Transformée de Hough . . . 17
- 3.2 Découpage de l'image . . . 19
  - 3.2.1 Détection des lignes finies . . . 19
  - 3.2.2 Extraction de la grille . . . 20
- 3.3 Extraction des chiffres de la grille . . . 21
  - 3.3.1 Suppression du bruit . . . 21
  - 3.3.2 Centrage des chiffres . . . 22

**4 Solver** **23**

**5 Réseaux de neurones** **24**
  - 5.0.1 XOR . . . 25
  - 5.0.2 Reconnaissance de chiffres . . . 26

**6 Reconstruction de la grille** **27**

**7 Interface** **28**
- 7.1 Glade . . . 28
- 7.2 Le code Gtk : interface.c . . . 30
- 7.3 L'apparence . . . 34

**8 Conclusion** **36**
- 8.1 Conclusions individuelles . . . 36
- 8.2 Conclusion générale . . . 36

# 1 Introduction

## 1.1 Présentation du groupe

### 1.1.1 Formation du groupe

Notre groupe est formé par 3 étudiants d'EPITA Strasbourg :
— Yann Messé
— Jules Haquin
— Vanina Barahona Undurraga

La création de notre groupe n'a pas été sans accrocs. En effet, notre groupe ne s'est pas formé de manière spontanée. La création du groupe a commencé lorsque Yann a demandé sur Discord s'il y avait des groupes incomplets ou si des personnes étaient encore toutes seules. Vanina a répondu à Yann, cependant les autres personnes sans groupe n'ont pas répondu. Jules nous a rejoint par la suite et nous avons donc eu un groupe de 3.

Cependant, dans le cahier des charges, il était spécifié que les groupes devaient être exactement de 4. Nous avons donc cherché une autre personne et demandé qui était déjà dans un groupe pour savoir, par élimination, qui n'était pas encore dans un groupe. Toutefois, personne n'a répondu à ce message et nous avons appris que nous étions 37 et non 40.

Nous avons donc demandé à Jimmy (la personne responsable de l'OCR à Strasbourg) comment faire. Nous avons donc reçu l'autorisation de créer un groupe de 3.

Ainsi notre groupe a vu le jour.

### 1.1.2 Ses membres

Yann :

Je suis étudiant à EPITA, en $2^e$ année du cycle préparatoire. J'ai fait un bac spécialité Mathématiques et NSI.

Durant ma première année, j'ai fait en projet de groupe un jeu 2D RPG dans un monde futuriste. Ce premier projet m'a permis d'acquérir l'expérience basique de projet en groupe, ce qui me permet de mieux me débrouiller pour ce projet.

En effet, ce nouveau projet que nous avons durant notre premier semestre de deuxième année de prépa est l'OCR. C'est un projet de traitement d'image. Un domaine dont je ne connais pas grand-chose et que nous allons devoir découvrir pour le mener à bien. Je suis persuadé que ce projet se déroulera dans une bonne ambiance.

Vanina :

Je suis actuellement étudiante en deuxième année du cycle préparatoire à l'EPITA. J'ai fait un bac spécialité Mathématiques et Sciences de l'Ingénieur. L'année dernière, j'ai fait en projet de groupe un jeu de hack'n slash sur la mythologie grec intitulé khaos.
Ce nouveau projet me permet de progresser en C avec des notions importantes que j'ai encore du mal à maîtriser comme les pointeurs ; autour d'une thématique que je ne connais pas très bien : le traitement d'image.

Jules :

Actuellement étudiant en SPE à EPITA, j'ai eu un baccalauréat spécialité Mathématiques et Physique-chimie.
Je démarre ce projet de S3 avec de l'enthousiasme. En effet je trouve la thématique du traitement d'image et en particulier des réseaux de neurones intéressante. C'est un sujet qui me stimule intellectuellement, beaucoup plus que le projet de S2.
De plus, ce projet va me permettre de progresser en C, langage que je connaissais assez peu avant de rentrer en SPE.

## 1.2 Présentation du projet OCR

## 1.3 Répartition des taches

### 1.3.1 Soutenance 1

| Tâches | Yann | Vanina | Jules |
|---|---|---|---|
| Chargement d'une image et suppression des couleurs | * | | |
| Rotation manuelle de l'image | * | | |
| Détection de la grille et de la position des cases | | * | |
| Découpage de l'image (sauvegarde des cases sous forme d'image) | | * | |
| Implémentation de l'algorithme de résolution de sudoku (Solver) | | | * |
| Réseaux de neurones | | | * |
| A commencer | | | |
| Sauvegarde et chargement des poids du réseau de neurones | | | * |
| Jeu d'images pour l'apprentissage ; | | | * |
| Manipulation de fichiers pour la sauvegarde des résultats | | | * |

Légende : '*' indique la personne qui s'occupe de la tâche.

### 1.3.2 Soutenance final

| Tâches | Yann | Vanina | Jules |
|---|---|---|---|
| Le prétraitement complet | * | | |
| Rotation automatique de l'image | * | | |
| Le réseau de neurones complet et fonctionnel | | | * |
| Apprentissage | | | * |
| Reconnaissance des chiffres de la grille | | | * |
| La reconstruction de la grille | | | * |
| La résolution de la grille | | | * |
| L'affichage de la grille ; | | * | |
| La sauvegarde du résultat | | | * |
| Une interface graphique permettant d'utiliser tous ces éléments | | * | |

# 2 Prétraitement

Le prétraitement est l'intégralité des étapes qui permettent de transformer les données brutes en données plus facilement analysables par l'ordinateur.

Pour le prétraitement, j'ai choisi d'utiliser la librairie SDL Image, déjà disponible sur les ordinateurs de l'école, car elle permet de simplifier grandement le traitement et la modification d'une image. J'ai aussi utilisé la librairie maths pour pouvoir effectuer les calculs de coordonnées plus facilement.

## 2.1 Contraste (Rappel)

La première opération que j'effectue sur l'image est l'augmentation du contraste qui me permet d'avoir des différences plus marquées entre le clair et le sombre, pour un prétraitement plus efficace. Pour cela, j'applique la fonction d'augmentation de contraste(Gamma) à chaque valeur RGB.

## 2.2 Grayscale (Rappel)

La première partie du prétraitement que j'ai réalisé pour la première soutenance est le passage en nuances de gris. Cette partie consiste à transformer les valeurs R,G,B qui sont différentes en une valeur unique. Nous obtenons cette valeur avec l'aide de la formule de normalisation visuelle de couleur.

## 2.3 Le filtre Gaussien (Rappel)

L'étape suivante de la binarisation est le floutage, avec le filtre gaussien. Cette étape permet de réduire la valeur des pixels isolés. Ce qui facilitera la binarisation et diminuera le nombre de pixel isolés. J'applique le filtre gaussien sur une matrice 3*3 dont voici la matrice de convolution :

$$Gx = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * A$$

## 2.4 La binarisation (Rappel et modification)

Pour la binarisation, j'ai décidé d'utiliser la méthode de Sauvola, qui permet de calculer des seuils locaux. C'est une amélioration de la méthode d'Otsu qui utilise le même principe.

J'ai aussi écarté les autres méthodes qui employaient des méthodes de calculs globales, car je ne les trouvais pas adaptés à notre utilisation, le but étant d'avoir une grille de sudoku la plus épurée possible.

Ainsi, j'ai utilisé la méthode de Sauvola. Le désavantage de cette méthode est qu'elle utilise une variable multiplicative fixe, ce qui ne fonctionne pas pour certain cas, tel que les images très sombres ou très claires, ou alors avec de faibles contrastes même après l'augmentation du gamma. Je fais donc varier la constante pour prendre en compte ces paramètres.

Les modifications effectuées entre cette soutenance et la précédente sont :

— la modification de l'étendue du seuil local de 11*11 à 17*17
— la modification de certaines valeurs de k constantes
— l'ajout d'un nouveau cas, si l'image ne possède pas de fort contraste et n'est ni sombre, ni clair

Voici l'image obtenu après l'application de toutes les étapes du prétraitement sur l'image 2 :

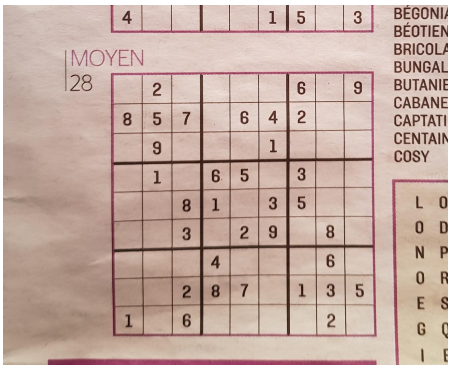


FIGURE 1 – Image de base

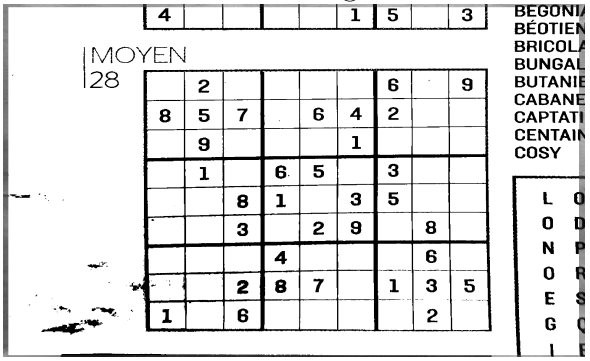


FIGURE 2 – Image après prétraitement

## 2.5 La rotation

### 2.5.1 La rotation manuelle (Rappel)

La rotation est un élément-clé du prétraitement et a été assez difficilement réalisée. En effet, pour effectuer la rotation, j'ai décidé de prendre le problème à l'envers. C'est-à-dire, de modifier les valeurs d'une autre surface (la nouvelle) en fonction des coordonnées trouvée par rotation dans la surface originelle (l'ancienne).

J'ai donc besoin d'une deuxième surface pour effectuer la rotation.J'effectue, s'il y a rotation, les 2 premières étapes sur la première surface (gamma et grayscale). Ce qui permet de simplifier le calcul des nouvelles coordonnées car il ne faut calculer qu'une seule couleur car l'image est en noir et blanc. Le reste du prétraitement est réalisé sur la 2ème surface qui sera celle que nous continuerons d'utiliser par la suite.

Pour trouver la valeur du pixel dans la surface originale, j'ai besoin de calculer les coordonnées dans l'ancienne surface. Pour cela j'utilise les formules de trigonométrie.

Pour cela je trouve d'abord le centre de l'image, j'utilise les propriétés de la trigonométrie des cosinus et des sinus. Ainsi l'angle de rotation est la conversion de l'angle en ° en Rad.

Cependant, la fonction ne s'arrête pas là, car les valeurs x et y ne sont pas entières. Donc, on récupère les valeurs entières qui entourent x et y et on applique une fonction mathématique qui est l'interpolation bilinéaire. Cette fonction permet d'obtenir la valeur du pixel en fonction des valeurs des pixels qui lui correspondent plus ou moins.

### 2.5.2 La rotation automatique (Recherche)

La rotation automatique est un problème largement rencontré lors du traitement d'images, soit pour recadrer une image, soit pour redresser un élément d'une image ou juste pour changer le sens de l'image ou son inclinaison.

Il existe de nombreux moyens différents pour redresser une image et de nombreux algorithmes existent. La première action que j'ai effectuée pour cette rotation automatique est de regarder les différentes façons de détecter un angle ou de faire une rotation d'image. J'ai trouvé de nombreux algorithmes très efficaces.

Le premier est la transformation affine, qui est une transformation qui utilise les fonctions de base de géométrie et des matrices pour faire de nombreuses actions telle que la translation, la mise à l'échelle, la rotation ou encore l'inclinaison. Cette méthode est très ingénieuse et repose sur des notions de bases des matrices.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

FIGURE 3 – Matrice pour transformation affine

| Type | Propriétés | Signification |
|---|---|---|
| Translation | $a_{ii}=1$ ; i=1,2 $a_{ij}=0$ $i \neq j$ | |
| Mise à l'échelle | $a_{12}= a_{21}=0$ | |
| Rotation | $a_{11}= \cos \alpha$ $a_{12}= -\sin \alpha$<br>$a_{21}= \sin \alpha$ $a_{22}= \cos \alpha$ | $\alpha$ : angle de rotation |
| Inclinaison-Biais | $a_{11}= 1$ $a_{21}=tg\beta$<br>$a_{21}= 0$ $a_{21}=1$ | $\beta$ : angle d'inclinaison |

FIGURE 4 – Tableau des transformation affine

La seconde que j'ai trouvée est la correction d'application créée par W. Bieniecki, S. Grabowski et W.Rozenberg, publié dans "Image Preprocessing for Improving OCR Accuracy",MELSTECH'2007. Le principe de cette fonction est de choisir un axe de référence. De calculer la distance entre l'axe et le premier point non blanc croisé, et de mettre ces coordonnées dans une liste. Si la distance par rapport à l'axe est inférieur au point précédent tester, alors ce point n'est pas pris en compte. S'il n'y a pas assez de point dans la liste, l'algorithme recommence, mais dans le sens inverse. C'est-à-dire qu'il regarde si la distance est supérieure au lieu d'inférieur. On obtient donc une liste de point. On peut donc avec cette liste calculer un angle moyen qui permet de redresser les bords.

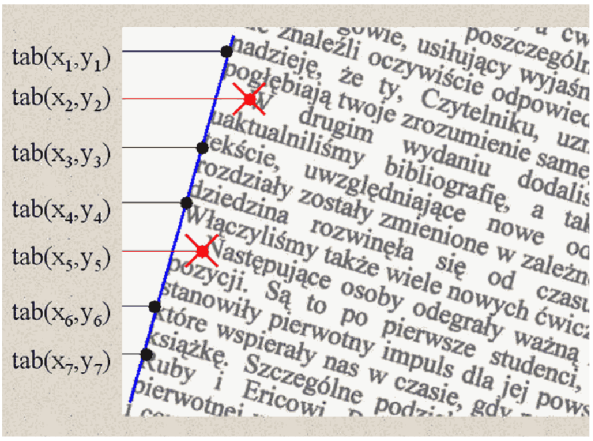


FIGURE 5 – Exemple de la méthode

La troisième est l'approche de Baird. Cette méthode repose sur une pseudo orientation de l'image. Le principe est le suivant, l'image est d'abord segmentée en composantes connexes. L'inclinaison est ensuite estimée à partir du maximum d'une fonction d'énergie. Les points centraux des composantes connexes sont ensuite projetés perpendiculairement sur une droite d'accumulation dont on fait varier la direction.

$$A(\theta) = \sum_{i=1}^{n} N_i^2(\theta)$$

FIGURE 6 – Formule de calcul pour trouver l'angle

Quoique toutes ces méthodes soient extrêmement intéressantes, j'ai décidé de ne pas les utiliser, car elles n'utilisent pas un aspect très important de notre projet. C'est que chaque image entrée possède une grille de Sudoku. Cela signifie qu'il y a toujours une certaine forme présente sur l'image à détecter et à reconnaître. Et que c'est cette forme en particulier que nous souhaitons orienter correctement.

J'ai donc décidé d'utiliser la détection de ligne pour calculer un angle moyen avec les formules trigonométriques. Cependant, après m'être concerté avec le membre de mon groupe responsable de la détection de ligne, je me suis rendu compte que la fonction de détection de ligne ne détectait que les lignes absolument horizontales et absolument verticales. Pour le bon déroulement de la suite de l'exécution de la détection de ligne, il a été décidé que nous allions utiliser les fonctions utiliser plus tard pour détecter les lignes, pour aussi détecter l'angle. Pour être sûr que la grille pourra être détectée.

### 2.5.3 La rotation automatique (Application)

La création de la rotation automatique a eu de nombreuses phases et changements, notamment dans les demandes des autres membres du groupe pour le bon déroulement des algorithmes et programmes suivants.

La première chose que j'ai faite après m'être rendu compte des spécificités de la détection de ligne, est de changer la fonction de l'algorithme. Les arguments de la fonction étaient, la première surface quelconque qui serra le résultat de la rotation, et la première surface, la surface où l'on a appliqué le filtre Sobel pour pouvoir détecter les lignes.

Comme expliqué précédemment, nous avons décidé de détecter l'angle en utilisant les fonctions de détection de lignes horizontales et verticales. Le plan était donc de détecter s'il y avait des lignes avec les fonctions faites pour, de compter le nombre total de lignes détectées, et de prendre l'angle pour lequel nous avons trouvé le plus de lignes.
Voila ce que faisait la rotation à ce moment là :

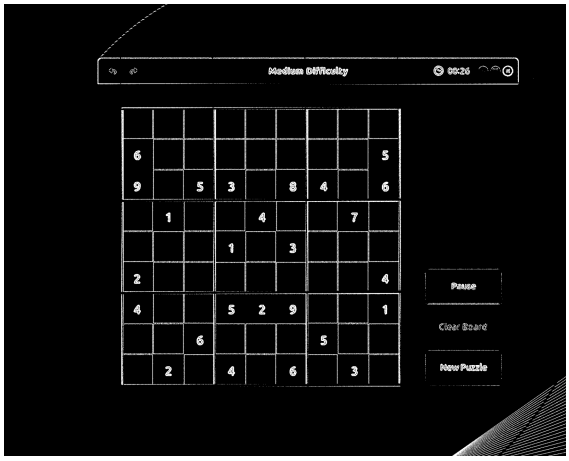


FIGURE 7 – Rotation nuances de gris

Le premier problème rencontré est que la rotation une fonction qui traite les images en noir et blanc, et que la fonction qui applique Sobel trace les lignes en bleu. J'ai donc dû recréer une fonction rotation qui avec le résultat de l'interpolation bilinéaire, appliquer un seuil pour savoir si le pixel est noir ou bleu. Ce qui a donc permis au programme de détection de l'angle d'être opérationnel.

Voici donc la bonne rotation de l'image ainsi que l'image retournée par le programme :


FIGURE 8 – Rotation noir et bleu

Cependant, il est nécessaire pour la suite du projet que la fonction retourne l'image en couleur. Pour cela, il faut premièrement ajouter un argument à la fonction, qui est la surface originelle dont je dois faire la rotation. Ensuite je récupère l'angle à l'aide de la détection de bord, puis je fait une rotation de l'image couleur pour obtenir l'image souhaitée.

Sauf que la rotation est encore une fois en nuances de gris ou alors avec la nouvelle rotation en noir et bleu. J'ai donc créé une nouvelle rotation pour la couleur. Pour cela j'ai créé de nouveaux outils.

Cependant je dois maintenant pour chaque couleur récupérer chaque composante couleur, calculer l'interpolation bilinéaire pour chaque couleur, et rechanger la valeur du pixel pour les 3 couleurs.

Après la création de toutes ces nouvelles fonctions j'ai pu obtenir les résultat suivant :

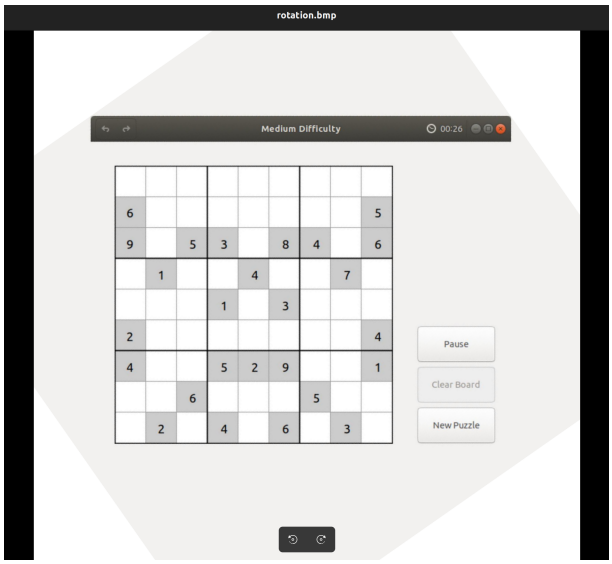


FIGURE 9 – Rotation couleur

C'était le résultat attendu et souhaité pour la poursuite correcte du programme.

# 3 Détection de la grille

## 3.1 Détection des lignes

### 3.1.1 Filtre de Sobel

Pour détecter les lignes, il faut d'abord détecter les bords et produire une image de bord. J'ai pour cela utilisé l'algorithme de Sobel. Le principe est de déterminer où il y a des changements radicaux d'intensité dans l'image.

Tout d'abord on cherche a calculer le gradient G de chaque pixel.

Pour cela on calcule Gx le gradient horizontal et Gy le gradient vertical qui sont la convolution d'une matrice définie (kernel) et de A, une matrice 3x3 contenant le pixel dont l'on veut calculer le gradient et ses 8 voisins qui l'entourent.

Ici on manipule des matrices, mais le tableau de pixels étant en une dimension, on peut le voir en tableau de deux dimensions grâce au row major order.

$$Gx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$$

$$Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Faire la convolution de deux matrices c'est multiplier point par point les éléments de même index des deux matrices, et d'en faire la somme. Voici ci-dessous un exemple quelconque :

$$\text{Posons } A = \begin{bmatrix} -3 & 4 & -2 \\ 1 & 6 & 0 \\ 0 & -2 & -3 \end{bmatrix}$$

Alors $Gx = 1 \times (-3) + 0 \times 4 + (-1) \times (-2) + 2 \times 1 + 0 \times 6 + (-2) \times 0 + 1 \times 0 + 0 \times (-2) + (-1) \times -3 = 4$

Le problème est qu'un pixel ne contient pas une seule valeur, mais trois : r, g et b. Je fais donc la moyenne des trois pour en avoir une.

Une fois qu'on a Gx et Gy, on peut à présent calculer G.

$$G = \sqrt{(Gx)^2 + (Gy)^2}$$

Enfin on compare ce gradient à un seuil. Si G est en dessous du seuil alors c'est un bord et on met le pixel à 0 (noir) sinon ce n'est pas un bord et on met le pixel à 255 (blanc).

Exemple de résultat de notre détection de bord :

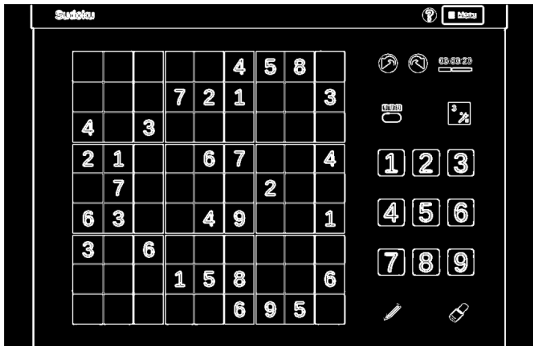


FIGURE 10 – Image de bord produite à partir du filtre de Sobel

### 3.1.2 Transformée de Hough

La transformée de Hough est une technique de reconnaissance de formes qui nous permettra de détecter les lignes.

La fonction houghTransform prend en paramètre l'image de bord et un tableau d'entier en deux dimensions appelé accumulator. Cet accumulateur représente **l'espace de Hough**.

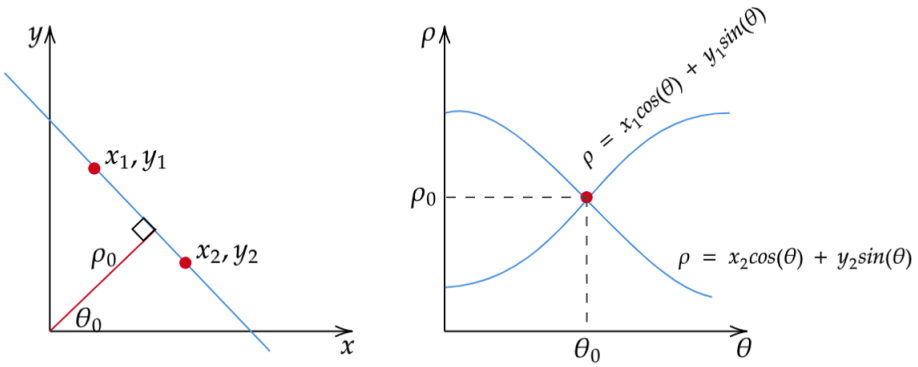


FIGURE 11 – Représentation d'une droite dans l'espace de Hough

Une droite va être représentée par une ligne normale perpendiculaire à cette droite et qui passe par l'origine. rho ($\rho$) est la longueur de cette ligne normal et se calcule :
$\rho = xcos(\theta) + ysin(\theta)$
Thêta est son angle.
Là où deux points se coupent dans l'espace de Hough, il y a une droite.

Après avoir calculé l'accumulateur de l'image, à l'aide de la fonction $houghTransform()$, nous parcourons l'accumulateur. Chaque valeur qui est au-dessus d'un certain seuil représente donc une droite dans l'espace de Hough. Plus ce seuil est bas, plus le nombre de lignes détectées va être grand.

Enfin, voici un exemple de notre détection de lignes sur une image de sudoku :

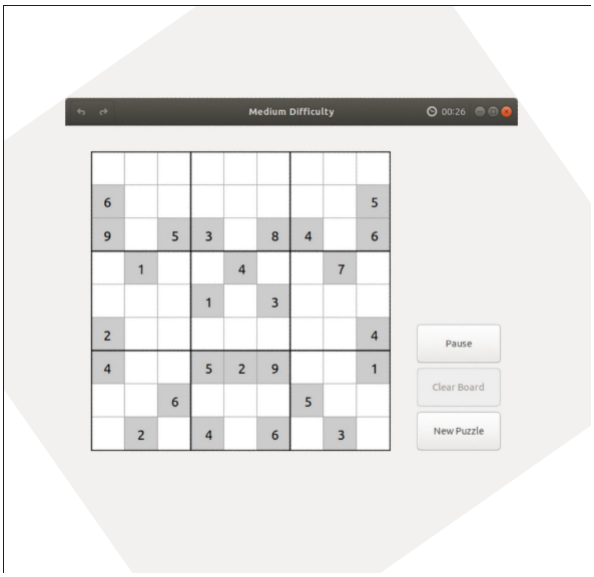


FIGURE 12 – Image de base

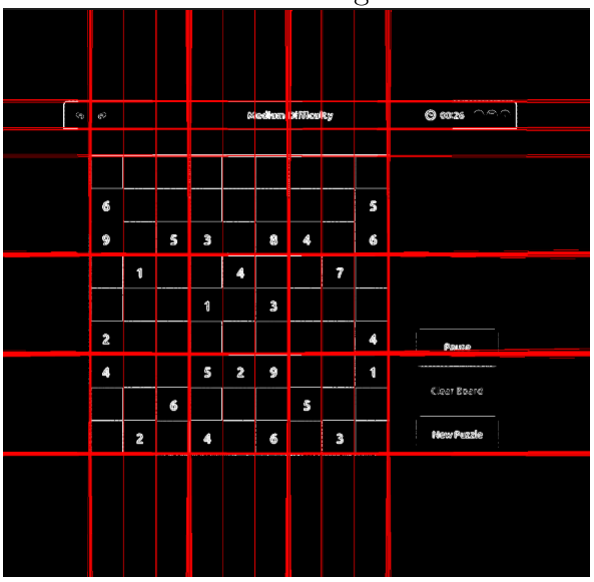

FIGURE 13 – Image après la détection de bord + la détection de lignes

## 3.2 Découpage de l'image

### 3.2.1 Détection des lignes finies

Comme vous avez pu le voir avec la partie précédente, les lignes que nous détectons sur l'image sont infinies, ce sont des droites.

Or, nous avons besoin de réduire ces lignes, afin de pouvoir détecter la position de la grille sur l'image. Pour passer d'une image avec des lignes infinies à une image avec des lignes finies, nous avons procédé de la manière suivante :

- nous parcourons les pixels de deux images, l'image avec les lignes infinies que nous créons avec la Transformée de Hough et l'image de bord que nous créons avec le filtre de Sobel
- chaque pixel qui est un pixel de bord (blanc sur l'image de bord) et qui fait parti d'une droite (rouge sur l'image précédente) est un pixel qui appartient à une ligne finie.
- on répète cette procédure pour tous les pixels de l'image.

Enfin, voici un exemple d'une détection de lignes finies, appliquée à la même image qu'à la section précédente :

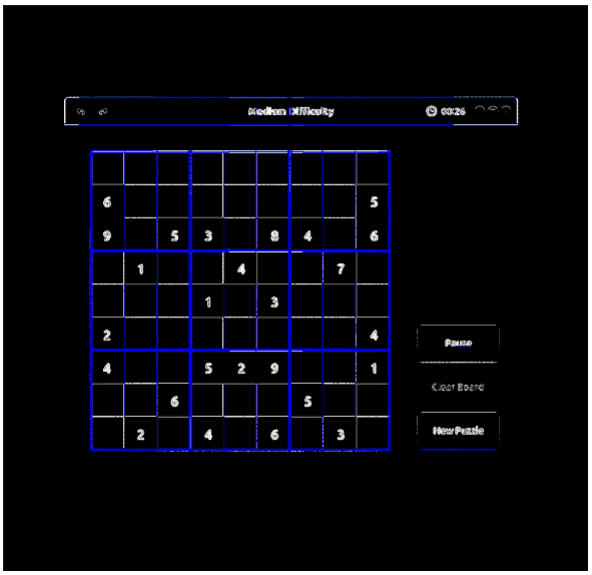

FIGURE 14 – Image après détection des lignes finies

### 3.2.2 Extraction de la grille

Une fois que nous avons détecté les lignes finies de l'image, la prochaine étape est de trouver la position exacte de la grille sur l'image.

Pour faire cela, nous allons déjà répertorier toutes les lignes horizontales et verticales de l'image qui sont plus grandes qu'un certain seuil. Nous procédons de la manière suivante :

- on parcourt toutes les lignes de l'image, si le nombre de pixels bleus (cf image précédente) de la ligne est supérieur à un certain seuil (ici 300 pixels) alors la ligne est considérée comme une ligne horizontale
- on réitère ce procédé sur les colonnes de l'image, afin de trouver les lignes verticales
- nous stockons la position de chaque ligne dans une *struct* appelée *Line*

.

Une fois que nous avons déterminé les positions des lignes horizontales et verticales qui sont plus grandes qu'un seuil, il ne reste plus qu'à déterminer la position de la grille sur l'image.

Pour détecter la position de la grille, nous parcourons toutes nos lignes horizontales en cherchant la ligne la plus haute sur l'image qui possède une intersection avec deux lignes verticales de longueur similaire.

A partir de cela, nous pouvons facilement détecter la position du reste de la grille.

Voici un exemple d'extraction de la grille, toujours appliqué à l'image de la section précédente :

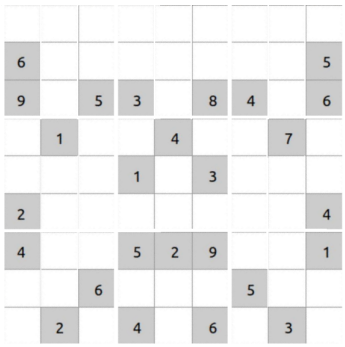


FIGURE 15 – Grille extraite de l'image

## 3.3 Extraction des chiffres de la grille

Après avoir extrait la grille de l'image, il nous suffit de diviser la grille en 81 carrés pour avoir une image de chaque chiffre.

Cependant, il est nécessaire d'appliquer quelques traitements à chaque chiffre extrait de la grille.

La raison est la suivante : la détection de grille n'est pas parfaite, donc les chiffres extraits de la grille ne sont pas parfaitement centrés et il y a quelques parasites (bouts de lignes, bruit de l'image) qui gênent la reconnaissance par le réseau de neurones.

### 3.3.1 Suppression du bruit

Après avoir appliqué le filtre de Gauss et après avoir transformé l'image en noir et blanc (le chiffre en blanc sur un fond noir), nous éliminons les éléments parasites de l'image.

Pour enlever les parasites de l'image d'un chiffre, nous procédons de la manière suivante :

- nous parcours tous les pixels de l'image du chiffre
- pour chaque pixel, nous déterminons récursivement le nombre de pixels blancs qui sont "accessibles" depuis ce pixel, c'est à dire le nombre de pixels blancs que l'on peut atteindre depuis le pixel, en ne passant que par des pixels blancs
- si le nombre de pixels blancs "accessibles" est inférieur à un certain seuil, alors le pixel devient noir
- le seuil est déterminé en fonction de la moyenne du nombre de pixels blancs de chaque chiffre

.
.
De cette manière, dans la très grande majorité des cas, l'image de chaque chiffre que nous obtenons ne contient aucun parasite. C'est seulement un chiffre blanc sur un fond noir, ou alors seulement un fond noir pour une case vide.

### 3.3.2 Centrage des chiffres

Une fois que nous avons bien supprimé tout le bruit parasite d'une image de chiffre, il ne nous reste plus qu'à centrer le chiffre sur l'image, afin qu'il soit reconnu plus efficacement par le réseau de neurones.

Cette étape est relativement simple, pour chaque chiffre de la grille, nous procédons de la manière suivante :

- on parcourt chaque pixel de l'image
- on détermine le pixel le plus haut, le plus bas, le plus à gauche et le plus à droite
- cela nous donne un encadrement de la position du chiffre
- on copie la partie de l'image qui correspond à l'encadrement, et on la colle au milieu d'un fond noir
- enfin, on redimensionne l'image obtenue pour qu'elle ait une taille de 28x28 pixels

# 4 Solver

Pour la résolution de la grille de sudoku, nous avons choisi d'implémenter l'algorithme de backtracking. C'est un algorithme récursif que nous connaissons déjà bien, car nous avons fait un TP dessus l'année dernière en C#.
L'algorithme fonctionne de la manière suivante :

- on trouve une case vide
- on essaye de la remplir par chaque chiffre (1-9)
- on lance la récursion sur tous les chiffres qui font que la grille reste valide
- s'il ne reste plus de cases vides, la grille est résolue
- s'il n'y a plus d'appels récursifs alors qu'il reste des cases vides, alors la grille n'est pas solvable

Notre programme solver fonctionne de la manière suivante :

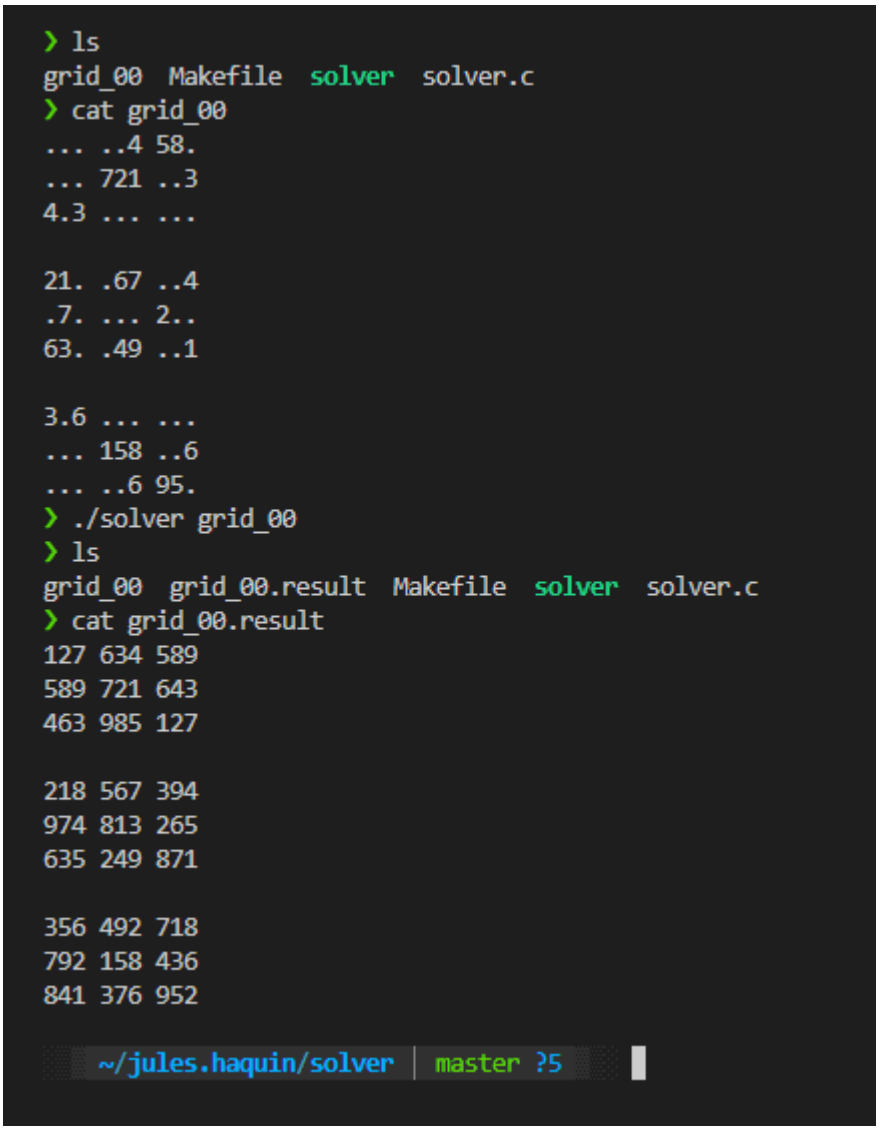


FIGURE 16 – Exemple d'utilisation du solver

# 5 Réseaux de neurones

Nos deux réseaux de neurones ont une structure générale assez similaire, malgré le fait qu'ils aient deux buts totalement différents. Ils sont tous les deux composés d'une couche pour l'entrée, d'une couche cachée et d'une couche pour la sortie. La seule différence majeure réside dans le nombre de neurones pour chaque couche.
Ils peuvent se décomposer en deux grandes parties :

**Forward-propagation :**
Cette fonction consiste à calculer la sortie de chaque neurone de la couche cachée et de la dernière couche.
Pour cela, nous utilisons la fonction sigmoïde :

$$f(x) = \frac{1}{1 + e^{-x}}$$

Sauf pour calculer la sortie de la dernière couche du réseau OCR, pour laquelle nous utilisons la fonction softmax :

$$g(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Cette fonction permet à notre réseau de prédire le nombre qu'on lui donne avec une probabilité entre 0 et 1.

**Back-propagation** C'est avec cette fonction que nos réseaux sont capables d'apprendre à résoudre un problème donné. Cette fonction est appelée après la forward-propagation. Elle permet d'ajuster les poids et les biais du réseau, en tenant compte de l'erreur effective qui est constaté après la forward-propagation.
L'entraînement des réseaux consiste donc à appeler la forward-propagation puis la back-propagation un certain nombre de fois (*epoch*), afin que les poids et les biais permettent à notre réseau de donner la bonne réponse.

### 5.0.1 XOR

Pour le XOR nous avons choisi d'implémenter un réseau avec deux neurones pour l'entrée, deux pour la couche cachée et un neurone pour la sortie. Nous pouvons représenter notre réseau par ce schéma :

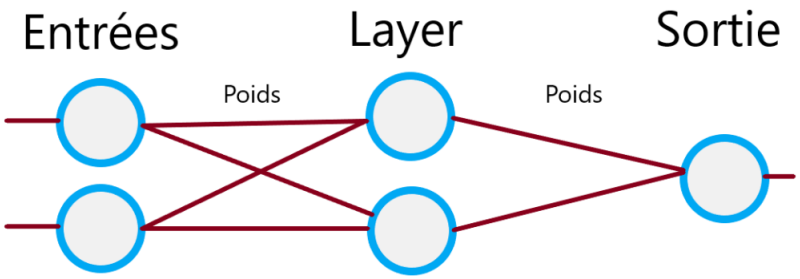


FIGURE 17 – Schéma du réseau de neurones XOR

Les résultats que nous obtenons avec 10 000 *epochs* et un taux d'apprentissage de 3 sont les suivants :

```
❯ ./xorNN
Network training with 10000 epochs and learning rate of 3.000000
XOR(0.000000, 0.000000) --> output = 0.005975, expected 0.000000
XOR(0.000000, 1.000000) --> output = 0.993875, expected 1.000000
XOR(1.000000, 0.000000) --> output = 0.993721, expected 1.000000
XOR(1.000000, 1.000000) --> output = 0.007615, expected 0.000000
Save network? (y/n): n

  ~/jules.haquin/NeuralNets | master ?5
```

FIGURE 18 – Utilisation du XOR

### 5.0.2 Reconnaissance de chiffres

Pour la reconnaissance de chiffres, notre réseau dispose de 784 neurones d'entrée car il prend des images de 28 pixels sur 28 pixels.

Après de multiples essais, nous avons choisi d'avoir 150 neurones pour la couche cachée. Chaque neurone étant relié à un neurone de la première couche et à un neurone de la couche de sortie.

La couche de sortie est quant à elle composée de 10 neurones. Chaque neurone correspond à un chiffre (sauf le zéro qui correspond à une image de case vide). La sortie finale du réseau de neurone est donc égale au neurone dont la sortie est la plus élevée.

Notre réseau peut être schématisé de la manière suivante :

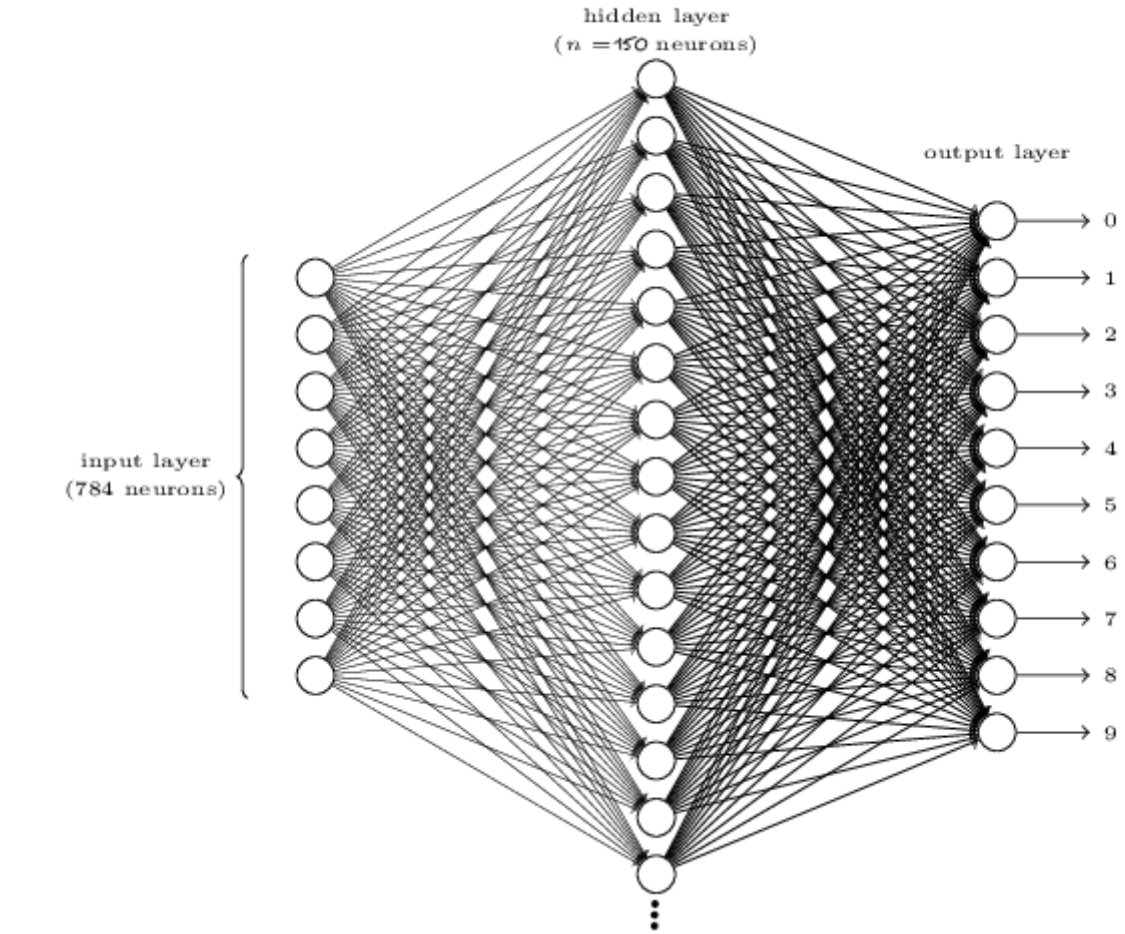


FIGURE 19 – Schéma du réseau de neurone

Presque la totalité du code qui permet de faire fonctionner le réseau était déjà écrit pour la première soutenance, notre travail a donc constitué de quelques améliorations mineures sur le code mais surtout de l'amélioration de notre jeu d'images pour l'apprentissage.

En effet, nous avons rajouté plusieurs milliers d'images à notre dataset de base, toujours en utilisant l'API de Google Fonts et la librairie Python Pillow. Nous avons également appliqué tous les prétraitements que nous appliquons sur les chiffres que nous voulons reconnaitre.

Grâce à ce travail, nous avons pu obtenir un réseau de neurones qui reconnait, dans la grande majorité des cas, le bon chiffre à partir d'une image.

# 6 Reconstruction de la grille

Maintenant que nous pouvons découper la grille, reconnaître les chiffres et la résoudre, nous pouvons la reconstruire.
Cette reconstruction se fait en 3 grandes étapes.

- Créer une surface blanche vide.
- Dessiner sur cette surface tous les chiffres à la bonne place. Pour cela nous avons une collection des neufs chiffres de 1 à 9 dans des carrés de 56x56 pixels. Nous utilisons la fonction SDL_BlitSurface() qui permet de copier notre surface contenant le chiffre sur notre surface vierge créer à l'étape 1 dans un carré à la position choisis.
- Enfin, nous traçons les lignes de la grille de sodoku.

Pour exemple, voici ci dessous une grille à résoudre et sa reconstruction résolue.

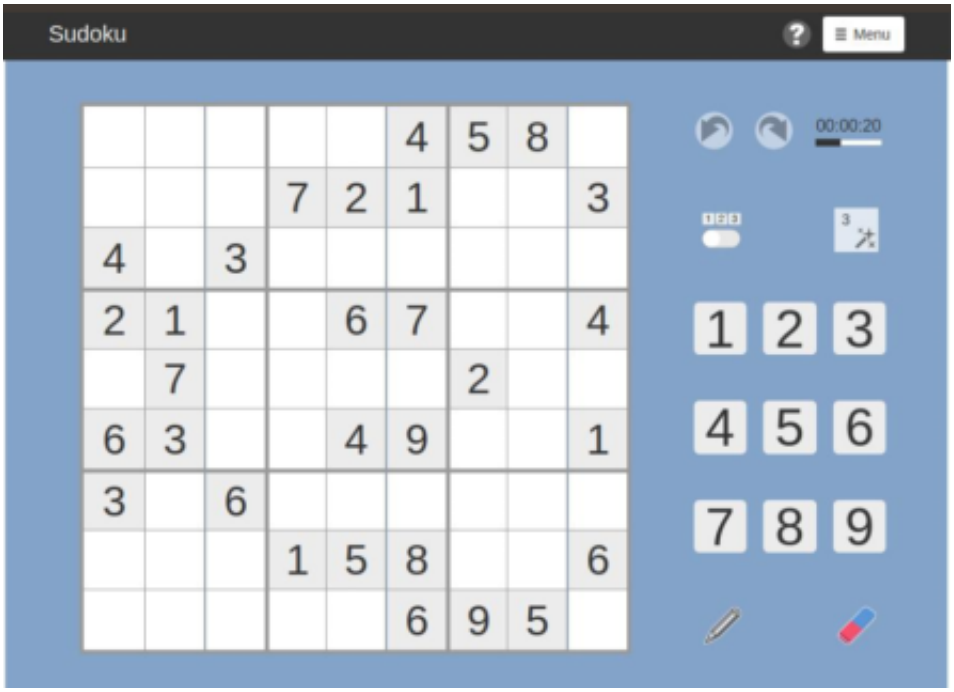


FIGURE 20 – Exemple d'une grille à résoudre

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

FIGURE 21 – Exemple d'une grille reconstruite résolue.

# 7 Interface

Pour rendre notre projet accessible, nous avons réalisé une interface. Cette dernière est composée de deux pages : une principale et une pour le réseau de neurones.

## 7.1 Glade

Pour concevoir l'interface, nous avons utilisé Glade. Glade est un outil de conception d'interface graphique GTK interactif. Il prend en charge toutes les parties de gestion/génération de l'interface. Glade enregistre l'interface graphique en générant un fichier XML.

**Containers** Pour placer les éléments dans l'interface nous utilisons des conteneurs.

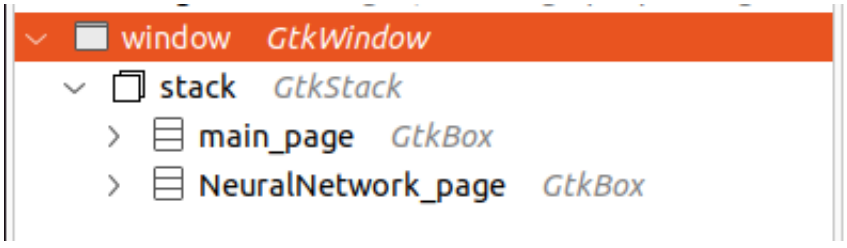


FIGURE 22 – Architecture principale de l'interface

GtkWindow
GtkWindow est la fenêtre qui contient tous les éléments nécessaires à notre interface.

GtkStack

Pour pouvoir passer de la page principale à la page réservée aux réseaux de neurones, j'utilise une stack.

GtkBox
Le conteneur que j'ai le plus utilisé est le GtkBox.

**Control**
Pour interagir avec notre interface, nous utilisons différents boutons adaptés à chaque besoin, de façon à ce que cela soit instinctif.

GtkButton
Le bouton le plus utilisé est le plus classique. Il envoie un signal quand l'utilisateur clique dessus.

GtkFileChooserButton
Nous l'utilisons pour que l'utilisateur puisse sélectionner le fichier correspondant au sudoku qu'il souhaite résoudre.

GtkEntry
Nous l'utilisons quand nous avons besoin que l'utilisateur écrive, ici le nom du fichier qu'il veut sauvegarder.

GtkSpinButton
Nous l'utilisons pour paramétrer le réseau de neurone. En effet, ce bouton permet à l'utilisateur de choisir une valeur dans un ensemble donné.

GtkCheckButton
Nous l'utilisons pour que l'utilisateur puisse sélectionner le réseau de neurone par défaut. Nous trouvons qu'une case à cocher est plus appropriée qu'un bouton "classique".

**Display**

GtkLabel
Pour une question de meilleure compréhension nous utilisons des Gtk Labels afin de donner des indications.

GtkImage
Les images peuvent être utilisées pour une meilleure compréhension comme par exemple pour les boutons de rotations. En effet, l'image d'une flèche est tout aussi parlante que du texte.
Pour pouvoir afficher la grille de sudoku à résoudre et sa solution, nous avons utilisé une GtkImage.

## 7.2 Le code Gtk : interface.c

Les fonctions Gtk permettent de réaliser des interfaces graphiques en C.

**Builder**
Le builder est un objet qui va lire le fichier XML ui2.glade et instancier les objets que nous pourrons exploiter par la suite. Pour cela on crée un nouveau GtkBuilder avec la fonction gtk_builder_new(). Ensuite on ajoute le contenu de notre .glade grâce à la fonction gtk_builder_add_from_file.
A présent, avec le builder nous pouvons accéder à tous les éléments de notre interface en utilisant la fonction gtk_builder_get_object().

```
// Gets the widgets.
//Window
GtkWindow* window = GTK_WINDOW(gtk_builder_get_object(builder, "window"));
//Stack
GtkStack* stack = GTK_STACK(gtk_builder_get_object(builder, "stack"));
```

FIGURE 23 – Exemple de création des variables window et stack contenant respectivement la GtkWindow et la GtkStack de l'interface.

**Connecter les signaux**

Quand l'utilisateur effectue une action particulière en interagissant avec le programme, cela émet un signal. Par exemple, quand l'utilisateur clique sur un GtkButton, cela envoie un signal "clicked" ; ou encore quand il sélectionne un fichier depuis GtkFileChooserButton cela envoie un signal "file-set".

```
g_signal_connect(start_button, "clicked", G_CALLBACK(on_start), &ui);
```

FIGURE 24 – Exemple de l'appel de la fonction on_start() quand l'utilisateur clique sur le bouton start.

Pour l'utilisation de G_CALLBACK, nous avons crée une structure UserInterface contenant toutes les variables nécessaires aux fonctions.

```
typedef struct UserInterface
{
    GtkWindow* window;              // Main window
    GtkButton* start_button;        // Start button
    GtkButton* left_rot_button;
    GtkButton* right_rot_button;
    GtkButton* auto_rot_button;
    GtkButton* training_button;
    GtkButton* save_button;
    GtkStack* stack;
    GtkImage* sudoku_image;
    GtkFileChooser* file_chooser_button;
    GtkWidget* page0;
    GtkWidget* page1;
    GtkLabel* process_label;
    GtkEntry* savename_entry;
    GtkEntry* name_ntw_entry;
    GtkEntry* custom_ntw_entry;
    GtkSpinButton* epoch;
    GtkSpinButton* nodes;
    GtkFileChooser* dataset;
    GtkSpinner* spin_nn;
    GtkLabel* success;
    char* filename;
    int rotation;

} UserInterface;
```

FIGURE 25 – Structure UserInterface

**Afficher une image**

Nous affichons une image lorsque l'utilisateur sélectionne une grille à résoudre et lorsque la grille à été résolue. Pour afficher une image nous avons besoin de garder en mémoire tampon les pixels de l'image dans un pixel buffer appelé GdkPixbuf. Ce dernier contient toutes les informations des pixels de l'image. Nous le créons depuis le fichier de l'image avec la fonction gdk_pixbuf_new_from_file(). Pour entrer dans notre interface l'image peut avoir besoin d'être redimensionner. Pour cela nous utilisons la bibliothèque SDL pour créer une surface et la modifier. Il ne faut pas oublier de free les surfaces puisque leurs créations allouent de la mémoire.
Enfin, quand l'image est prête, nous utilisons la fonction gtk_image_set_from_pixbuf() pour afficher le pixbuf sur la GtkImage.

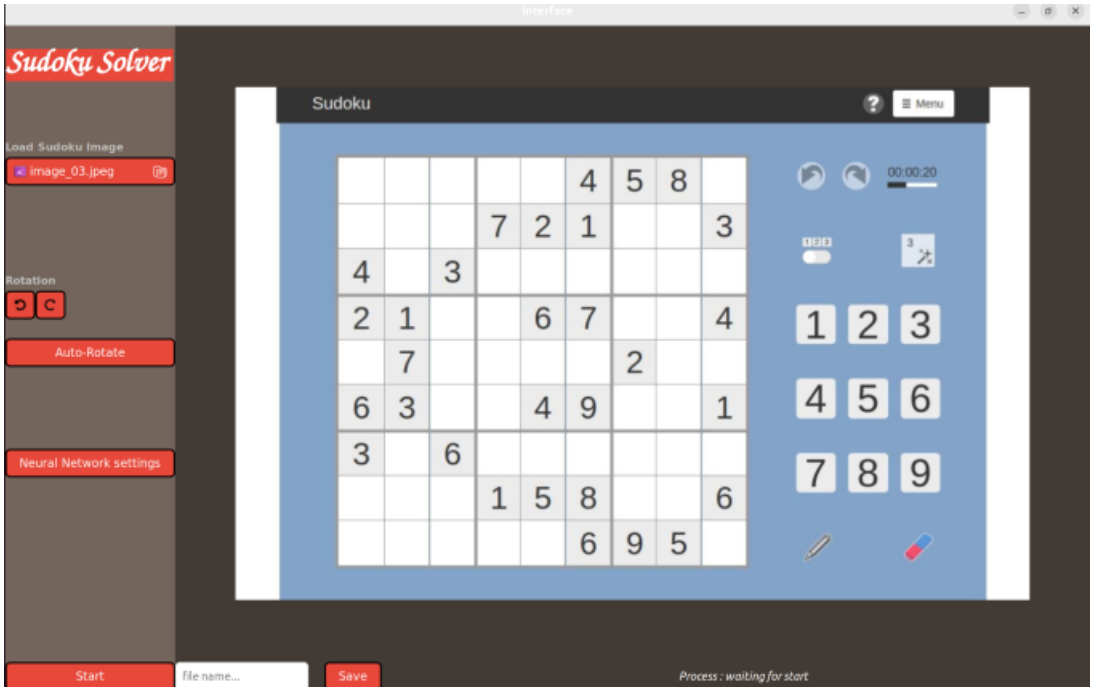


FIGURE 26 – Affichage de l'image_03 sur l'interface.

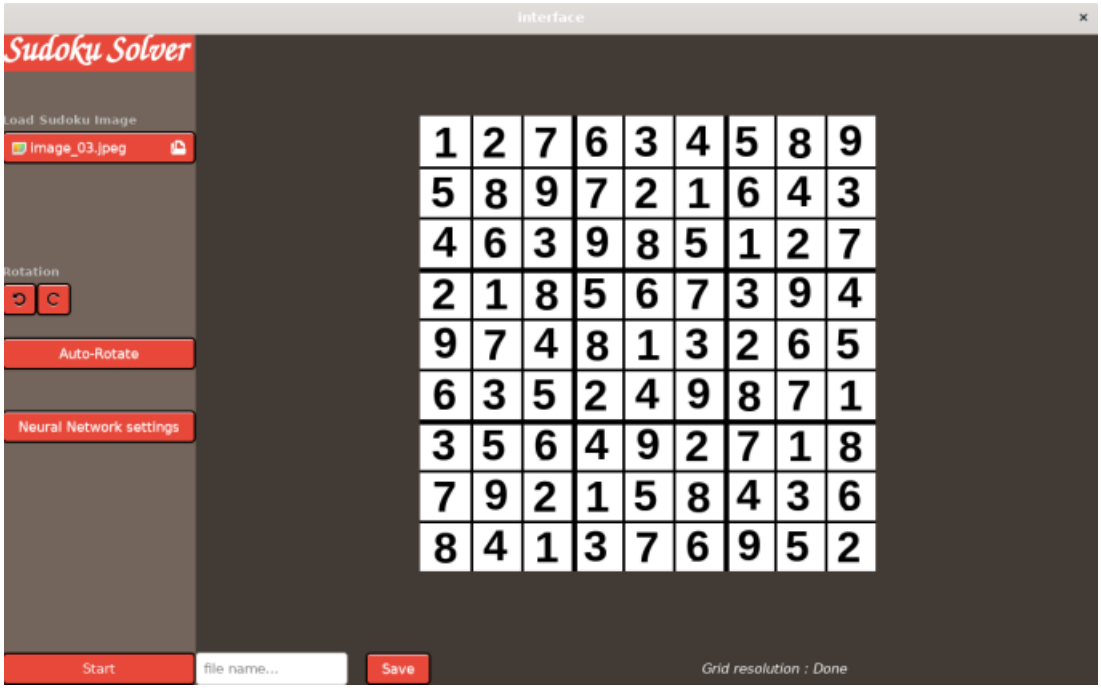


FIGURE 27 – Affichage de la solution de l'image_03 sur l'interface.

## 7.3 L'apparence

Pour mettre en forme l'apparence de mon interface, j'utilise un fichier .css. CSS est l'acronyme de « Cascading Style Sheets » ce qui signifie « feuille de style en cascade ».
Pour "connecter" notre fichier .css à l'interface, on utilise un GtkCssProvider que l'on associe à un GdkScreen.

```
// Loads the CSS file
GtkCssProvider *cssProvider = gtk_css_provider_new();
gtk_css_provider_load_from_path(cssProvider, "./style.css", NULL);
GdkScreen *screen = gdk_screen_get_default();
gtk_style_context_add_provider_for_screen(screen,
                                          GTK_STYLE_PROVIDER(cssProvider),
                                          GTK_STYLE_PROVIDER_PRIORITY_USER);
```

FIGURE 28 – Lignes de code pour charger le fichier file et modifier l'apparence de l'interface

Cependant, cela ne suffit pas. Il faut également associer les classes aux éléments dans gtk. Cela se fait directement dans glade en indiquant le nom de la classe.


FIGURE 29 – Partie "Common" de la GtkBox de gauche

```
.left_p{
    background-color: #73655C;
}
```

FIGURE 30 – Classe .left_p dans le fichier style.css.

Pour modifier précisément l'apparence des boutons, il faut le préciser dans le code en ajoutant button au nom. Par exemple, dans notre interface, nous avons décidé de mettre les boutons en rouges, les bordures en noirs et l'écriture blanche avec la police Verdana.

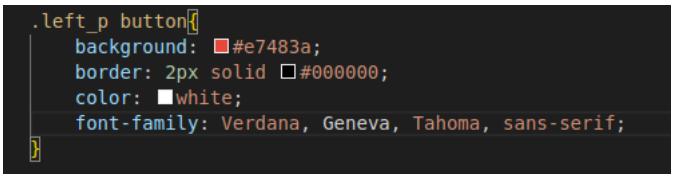


FIGURE 31 – Classe du style.css qui modifie l'apparence des boutons présents dans le conteneurs à qui la classe .left_p a été associée.

# 8 Conclusion

## 8.1 Conclusions individuelles

Yann :
Cette fin de projet m'a permis de complètement m'adapter aux spécificités du langage C. Effectuer le traitement de l'image m'a permis de me rendre compte de l'ampleur du lien entre les mathématiques et l'informatique et m'a aussi permis de découvrir de nouveaux domaines d'application de ces disciplines.

Ce projet m'a donc été très bénéfique dans son ensemble car il m'a permis de découvrir de nouvelles personnes. Il m'a aussi permit de découvrir que l'application pratique des maths passe aussi par l'informatique.

Vanina :
Je me suis heurté à des difficultés lors de ce projet, notamment concilier les cours et le projet. Cependant j'ai beaucoup appris. En effet j'ai pu renforcer des notions importantes en C tel que les pointeurs et les bibliothèques SDL et Gtk. De plus, j'ai également acquis de nouvelles connaissances comme la prise en main de Glade et les bases du langage CSS.

Jules :
C'est avec un sentiment de satisfaction que je conclus ce rapport. Je trouve que ce que nous avons produit pour ce projet reflète la qualité du travail apporté. Ce n'était pourtant pas gagné d'avance, en effet, nous sommes un groupe de 3, nous avons eu quelques retards sur la première soutenance et ce n'était pas toujours facile. Malgré ces quelques écueils, nous avons, je le pense, réussi à terminer ce projet jusqu'au bout.

## 8.2 Conclusion générale

Pour conclure sur ce projet, nous avons réussi à surmonter les difficultés. En effet, lors de la première soutenance nous étions en retard au niveau de la détection de la grille, mais grâce au travail d'équipe nous avons pu rattraper ce retard. Malgré que nous étions que 3 pour un projet de 4, nous avons réussi à finir et remplir tous les objectifs requis. De plus nous sommes fiers de notre travail et avons chacun appris beaucoup de choses.