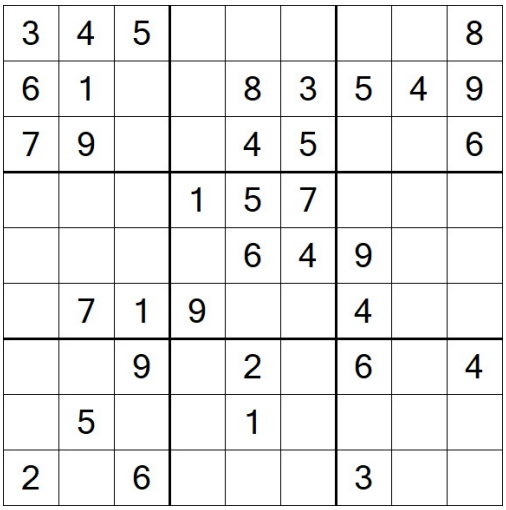
Rapport
OCR Sudoku Solver



Au pneu AI

Médéric LAVIROTTE
Ivan HUARD
Sacha WIZEN

Contents

1 Introduction

1.1 Le projet

Pour ce semestre à Epita, nous avons formé une équipe nommée Au pneu AI composé de trois codeurs dévoués, pour réaliser le projet OCR Sudoku Solver.

L'objectif de ce projet est de réaliser un logiciel de type OCR (Optical Character Recognition) qui résout une grille de sudoku. Notre application prendra donc en entrée une image représentant une grille de sudoku et affichera en sortie la grille résolue. Dans sa version définitive, notre application devra proposer une interface graphique permettant de charger une image dans un format standard, de la visualiser, de corriger certains de ses défauts, et enfin d'afficher la grille complètement remplie et résolue. Nous allons mettre en place un aspect d'apprentissage qui permettra d'entraîner notre réseau de neurones, puis de sauvegarder et de recharger le résultat de cet apprentissage. Ce document est le fruit de nos travaux et aura pour but de vous présenter l'étendue du projet créé.
Bonne lecture.

1.2 L'équipe

Médéric LAVIROTTE :
Je m'appelle Médéric LAVIROTTE, j'ai 19 ans, et je suis passionné par l'informatique. Ma vie a changé depuis que j'ai découvert Vim. Coder sur Vim devient tout un coup une expérience spirituelle, c'est sublime. J'adore également le C; je trouve que c'est un très beau langage, notamment grâce au contrôle total sur la gestion de la mémoire. Le projet d'OCR est très intéressant grâce à sa diversité. En effet on étudie beaucoup de sujets variés comme le réseau de neurones, la résolution d'un sudoku ou encore les traitements d'images. Je suis donc très motivé et je donnerais mon maximum pour la réussite de ce projet.

Ivan HUARD :
C'est moi qui ai choisi le nom de l'équipe et je ne comprends vraiment pas pourquoi personne ne l'aime. Je trouve ça pourtant très original et drôle. Sinon je suis motivé pour ce projet.

Sacha WIZEN :
Je suis nul en math. Voila comme ca c'est dit. Et on m'a quand même donné la partie ou il y a le plus de maths : la rotation et la détection de la grille. Ca c'est vraiment pas malin. Je commence déjà à avoir mal à la tête rien qu'en lisant la documentation sur wikipedia. Ça va être long. Et sinon, je ne comprends toujours pas le jeu de mots sur le nom de l'équipe. Enfin je suis comme d'habitude très motivé étant donné que coder c'est la vie.

1.3 La répartition des charges

	Mederic	Ivan	Sacha
Prétraitement		***	*
Rotation	**		***
Reconnaissance de la grille	**		***
Découpage de la grille	***		
Réseau de neurones		***	
Resolution du sudoku	***		
Interface Graphique			***

Le degré d'implication : *

2 Prétraitement

2.1 Filtre Sobel

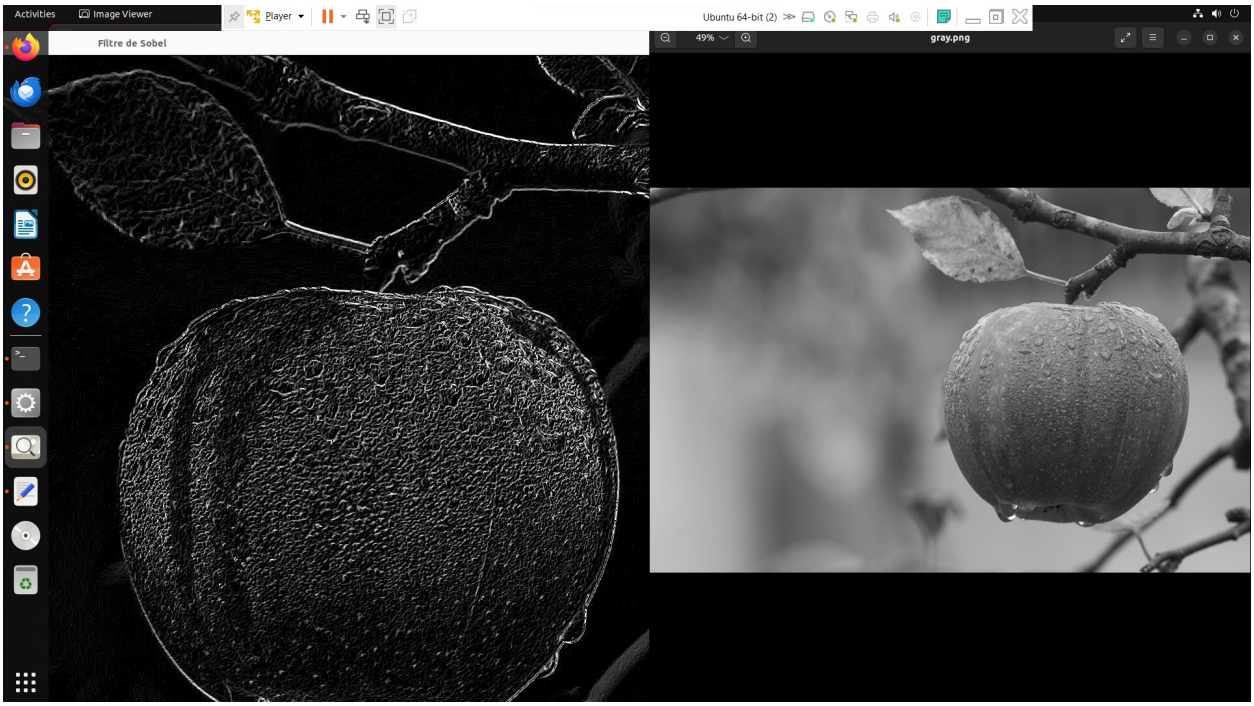
Le principe du filtre de Sobel est le suivant : détecter les bords des éléments dans une image en analysant les variations d'intensité des pixels. Afin de simplifier sa tâche, on convertit d'abord l'image en niveaux de gris, en utilisant le code développé (grayscale) lors d'un TP. On applique un filtre sur chaque pixel de l'image.

```
Uint8 r, g, b;
SDL_GetRGB(pixel_color, format, &r, &g, &b);
float average = 0.3*r + 0.59*g + 0.11*b;
Uint32 color = SDL_MapRGB(format, average, average, average);
return color;
```

Cette conversion en gris permet de se débarrasser des trois canaux de couleurs et de travailler directement avec l'intensité de l'image. Ensuite, on a créé une nouvelle "SDL_surface" (une nouvelle image) en recréant chaque pixel en fonction de ses pixels environnants. Pour déterminer s'il y avait une variation d'intensité, on compare le pixel en question avec les pixels voisins. Pour ce faire, il a utilisé des noyaux, qui sont en réalité des matrices pré-remplies.

```
int sobel_kernel_x[3][3] = {
        { -1, 0, 1 },
        { -2, 0, 2 },
        { -1, 0, 1 }
};

int sobel_kernel_y[3][3] = {
        { -1, -2, -1 },
        { 0, 0, 0 },
        { 1, 2, 1 }
};
```

Ces noyaux ont aidé à détecter les changements d'intensité dans un bloc de l'image. Si la somme des pixels environnants dépassait un certain seuil, cela signifiait qu'il y avait un changement d'intensité et donc un contour, et la fonction "dessine" le pixel en blanc. Sinon, on laisse en noir. Le résultat de la transformation de l'image en gris par la fonction Sobel :



2.2 Black and White

La conversion d'une image en noir et blanc, également connue sous le nom de binarisation, est une autre option de filtre dans le traitement d'images. Cette technique simplifie l'image en réduisant chaque pixel à une valeur binaire : noir ou blanc.

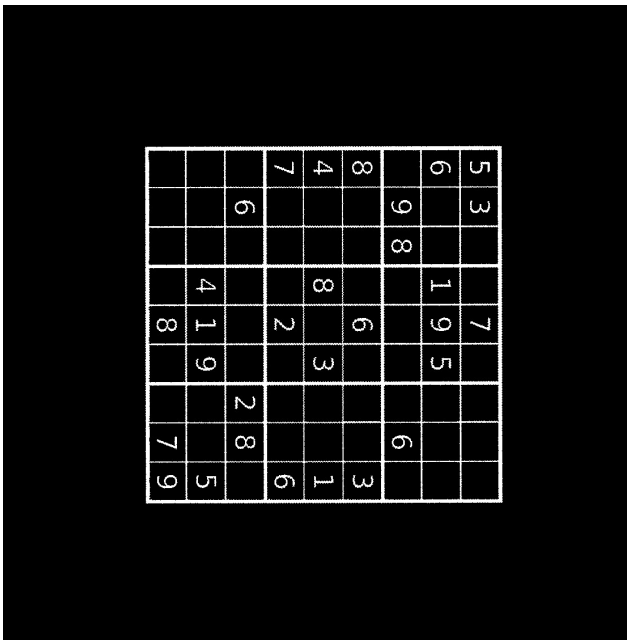
Le processus de binarisation commence par la conversion de l'image en niveaux de gris, suivie par l'application d'un seuil. Si l'intensité d'un pixel dépasse ce seuil, le pixel est converti en blanc (généralement représenté par la valeur 1), sinon il est converti en noir (représenté par 0).

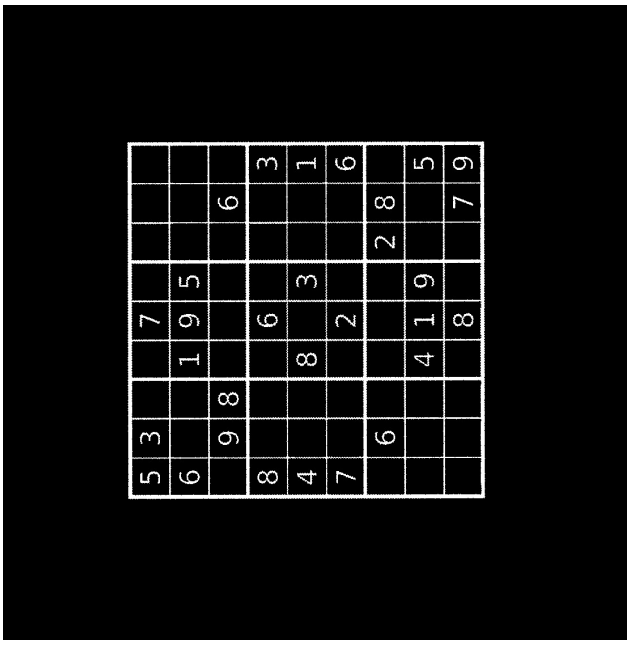
La binarisation facilite les étapes suivantes de notre projet, telles que la détection de la grille et la reconnaissance des chiffres, en réduisant la complexité des images et en mettant en évidence les caractéristiques essentielles.

2.3 Rotation manuelle

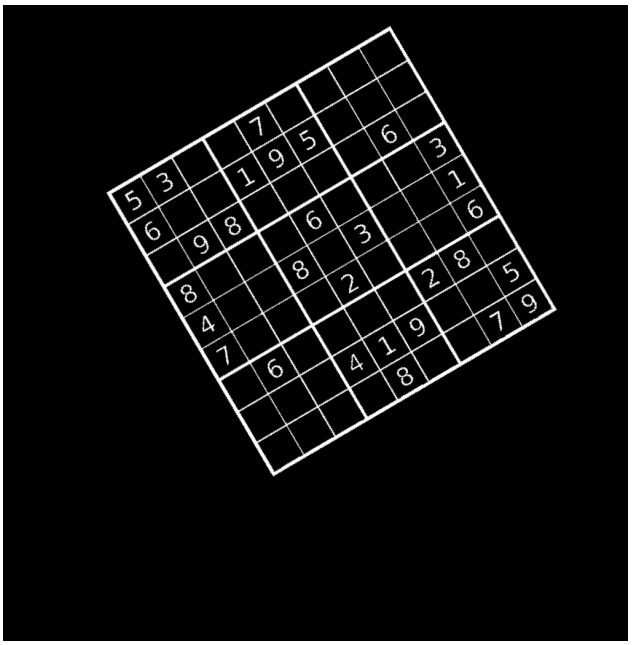
La rotation manuelle sera utilisée pour faire la rotation de l'image lorsque la rotation automatique ne se fait pas correctement. C'est en quelque sorte une roue de secours. La rotation automatique ou manuelle de l'image est une partie obligatoire dans le projet. En effet pour le découpage des cases on a besoin que

la l'image soit bien droite sachant que le découpage se fait de manière parallèle à l'image.
Afin d'implémenter une rotation qui satisfait tous nos critères, nous sommes passés par plusieurs étapes, la première était d'effectuer des rotations simples 90 – 180 – 270 (les multiples de 90). Pour implémenter ces opérations nous effectuions des calculs sur les coordonnées dépendant de l'angle afin de récupérer les pixels sur l'image d'origine et les placer à leur position à la suite de la rotation sur l'image de destination. Ces calculs individuellement était assez élémentaire, mais les effectuer tous en même temps demande un certain temps d'optimisation.
voici les sudokus avec les rotations simples à effectuer :





Toutefois, pour effectuer des rotations plus complexes nous avons décidé de changer d'approche. En effet, si nous effectuons une rotation avec un angle qui n'est pas un multiple de 90 alors l'image n'est plus alignée avec les axes du repère, cela veut dire que certaines parties de l'image peuvent déborder de celle-ci. par exemple celle ci:



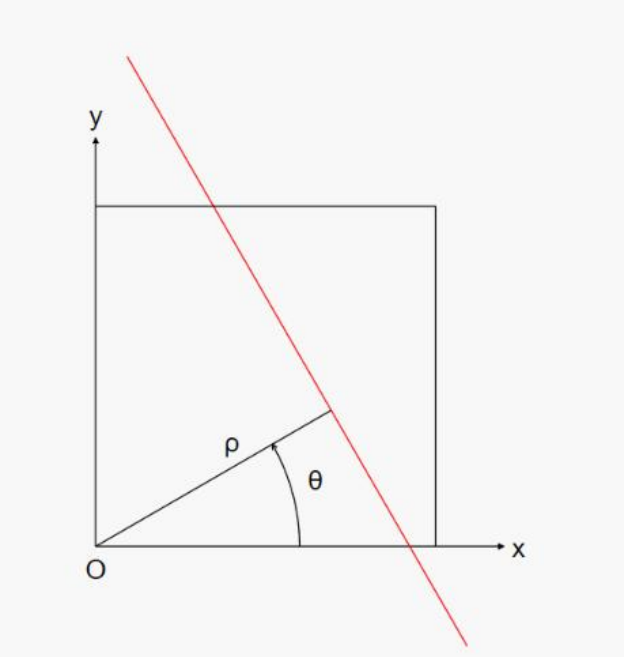
Afin d'éviter ces problèmes nous effectuons un calcul des dimensions de l'image de destination, ainsi si l'angle n'est pas un multiple de 90 alors l'image de

destinations sera plus grande que celle d'origine.Une fois cela effectué, nous effectuons ensuite une rotation autour de son centre qui ne change pas selon l'angle de rotation. Un des éléments importants des rotations d'un angle différent d'un multiple de 90 était de parcourir l'image de destination est non pas l'image de départ car comme la taille de l'image de destination est amené à changer cela peut créer des problèmes d'indices si on ne fait pas attention.

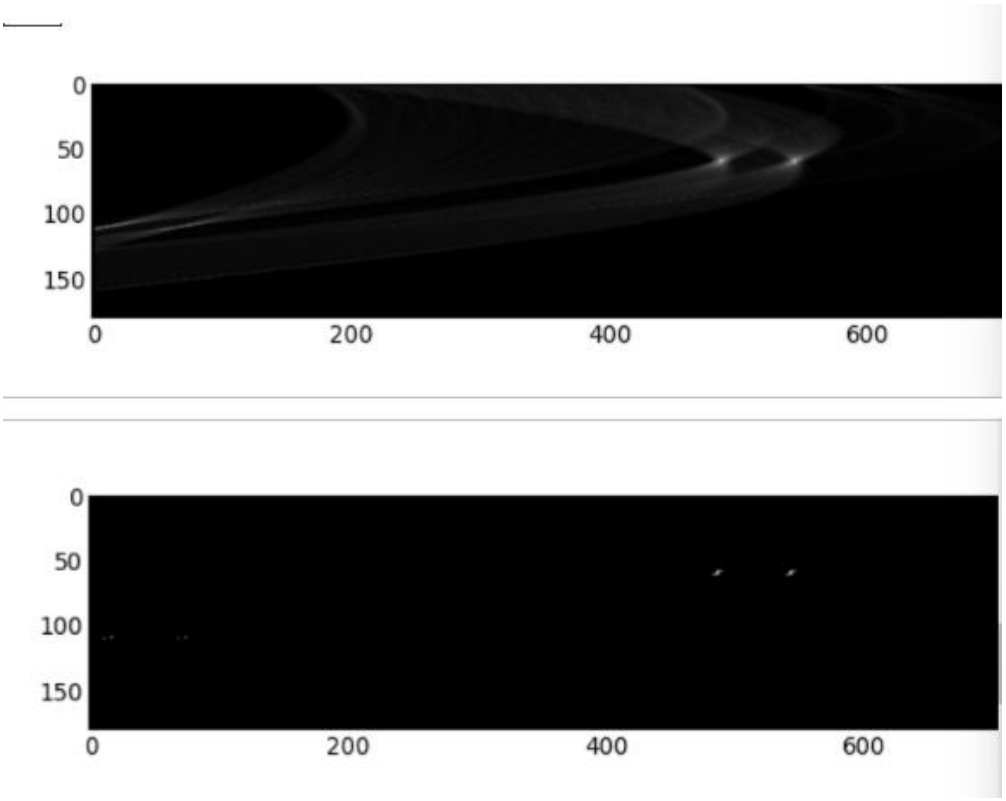
3 Détection

3.1 Hough

Nous avons implémenté un algorithme nous permettant de déterminer les lignes de la grille du sudoku, afin de faire cela nous avons utilisé l'algorithme de la transformation de hough qui nous permet de détecter de manière approximative les droites de l'image. L'image à traiter doit être binaire (noir et blanc) afin que cette fonction nous permette de créer une matrice qui correspond à un domaine rectangulaire dans laquelle chaque valeur représente le nombre de fois ou une droite a été considéré comme une ligne potentielle du sudoku. Plus précisément chaque pixel rencontré nous permet d'incrémenter d'une unité la probabilité qu'une droite existe. Cet algorithme met en application les propriétés mathématiques qui composent une droite, en effet une droite dans un plan peut être mise sous la forme : $ax + by + 1 = 0$, et pour chaque pixel de l'image une droite est tracée dans l'accumulateur. Cependant il est impossible dans notre cas de prendre en compte toutes les valeurs de coefficients (petite, moyenne, grande) c'est pour cela que la transformation de hough utilise les paramètres : $p = \cos(\rho) + \sin(\rho)$ qui correspondent aux coordonnées polaires.



Pour chaque pixel de l'image nous avons donc incrémenter dans la matrice qui est considéré comme un accumulateur les points de la courbe allant de 0 à 180 degrés. Une fois que nous avons traité tous les pixels composant l'image il faut récupérer la valeur maximal contenu dans l'accumulateur, grâce à lui nous allons pouvoir déterminer un seuil à partir duquel les valeurs supérieurs à celui-ci seront considérées comme des droits de l'image.



3.2 Rotation automatique

La rotation automatique est la dernière partie du prétraitement. Celle-ci se fera avant la détection des lignes pour pouvoir mieux détecter les lignes et ensuite bien découper. Pour la rotation automatique nous allons utiliser la fonction de la rotation manuelle. La rotation automatique va donc se faire en deux étapes : le calcul de l'angle de rotation et l'utilisation de la fonction de rotation manuelle avec l'angle calculé. Pour le calcul de l'angle de rotation nous avons utilisé la fonction de la transformation de hough expliquée précédemment. On va détecter les lignes horizontales du sudoku pour ensuite calculer la différence d'angle entre celle ci et celle des abscisses.

Une fois que nous avons effectué la transformation de hough est que nous avons déterminé un seuil à partir duquel nous considérons que les paramètres sont des droites nous avons implémenté une fonction nous permettant de tracer les droites. Le traçage de la grille nous permet de vérifier que la détection se fait correctement. Pour ce faire nous sommes d'abord partis sur une implémentation qui consistait à changer de couleur chaque pixel un par un pour chaque pixel

composant les droites, cette méthode était plutôt efficace au début sur des images parfaitement droite cependant cette méthode permet de tracer des lignes pas seulement droites mais également courbe. Ainsi nous avons donc changé notre méthode de traçage et avons opté pour l'utilisation de la méthode inclus dans la bibliothèque SDL : Drawline. Cette méthode nous à permis de tracer des lignes plus droites qu'à l'origine cependant cette méthode nous oblige à avoir un prétraitement de l'image optimal afin que l'algorithme de Hough et notre algorithme de traçage fonctionnent dans les meilleures conditions.

Pour le calcul de l'angle nous allons calculer l'angle en radians, l'angle de la ligne Calcul de l'Angle en Radians :
La conversion de l'angle theta détecté par la transformation de Hough de degrés en radians se fait comme suit :

```
double angleRad = (t - 90) * M_PI / 180;
```

Ici, t représente l'angle thêta en degrés. Cette formule convertit cet angle en radians en ajustant l'angle par rapport à l'axe vertical et en utilisant la relation standard pour la conversion de degrés en radians. Cet angle en Radian va permettre de calculer les coordonnées x et y.

```
int y0 = (p - Rhos) / sin(angleRad);
int x0 = 1;
int x1 = width - 1;
int y1 = x1 * - cos(angleRad) / sin(angleRad) + y0;
```

Ensuite on calcule l'angle de la ligne tracée par rapport à l'axe horizontal en utilisant la fonction atan2 :

```
double lineAngleRad = atan2(y1 - y0, x1 - x0);
```

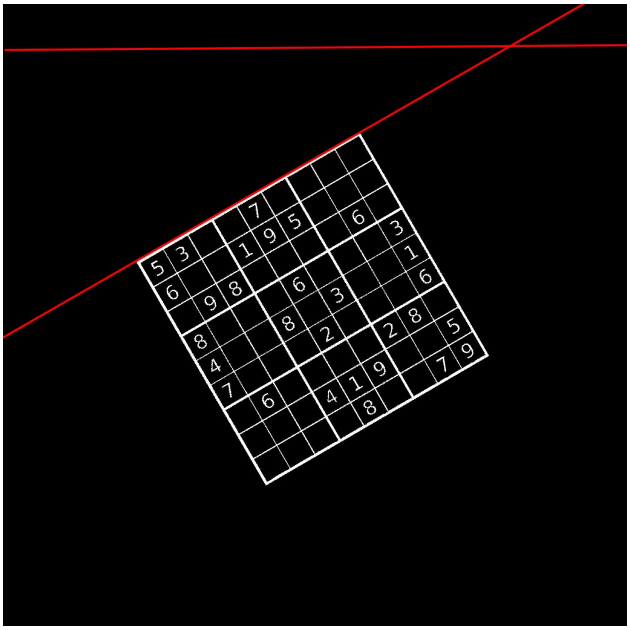
La fonction atan2 retourne l'angle entre l'axe des abscisses et le vecteur allant du point $(x0, y0)$ au point $(x1, y1)$, en radians.

L'angle en radians est ensuite converti en degrés :

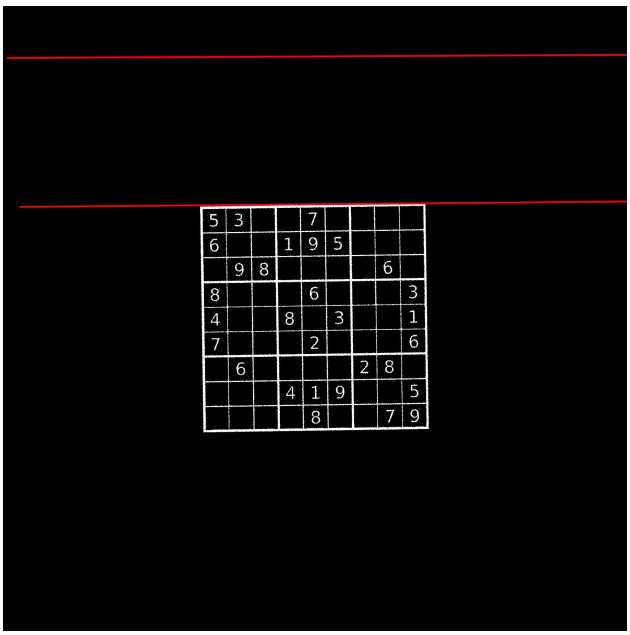
```
double lineAngleDeg = 90 - (-lineAngleRad * 180.0 / M_PI);
```

Cette formule transforme l'angle en radians en degrés et ajuste l'angle par rapport à l'axe vertical.

Enfin nous utilisons un filtre des angles pour détecter que les axes horizontaux de la grille de sudoku. En effet nous voulons que l'axe horizontales soit parallèles à l'axe des abscisse.
Ici nous avons une grille de sudoku black and white qui n'est pas droite. On voit que les lignes horizontales ne sont pas parallèles à l'axe des abscisses.



Ici la même grille est bien droite. En effet l'axe horizontal de la ligne de la grille est bien parallèles à l'axe des abscisses.

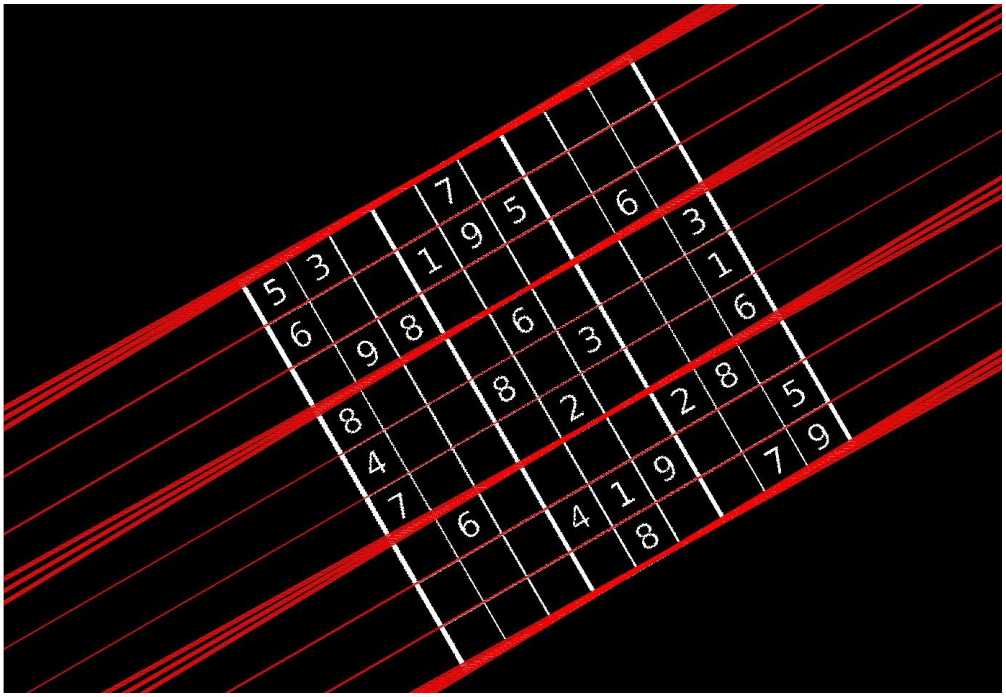


Pour filtrer les lignes horizontales nous comparons l'angle en radian à 90: Enfin on dessine une ligne pour vérifier le résultat

```
if(lineAngleDeg < 90){
SDL_RenderDrawLine(renderer, y0, x0, y1, x1);
```

```
}
```

Voici le resultat :



Nous remarquons que pour un seul même trait plusieurs lignes se sont tracées avec des angles légèrement différents. Pour augmenter la précision nous allons calculer tous les angles de toutes les lignes horizontales par rapport a l'abscisse pour en faire une moyenne. Enfin nous faisons une rotation de l'image avec la fonction de rotation manuelle avec l'angle trouve en moyenne. Sur l'exemple précédent nous trouvons une moyenne de 29.671918

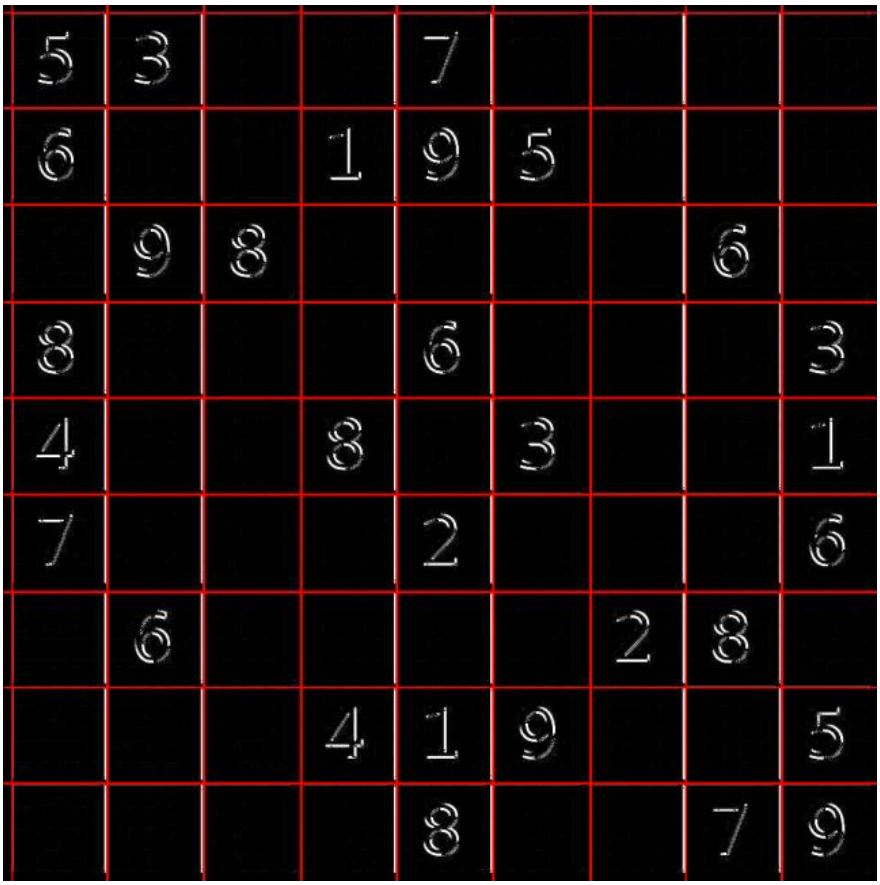
3.3 Détection de la grille après la rotation

Nous allons répéter le même processus c'est-à- dire refaire la détection de la grille avec la transformation de hough avec l'image qui sera droite. La différence ici est qu'on ne cherche pas l'angle des lignes de sudokus (car elles sont censé être horizontal), mais on cherche a trouver et stocker les coordonnées de ses lignes.

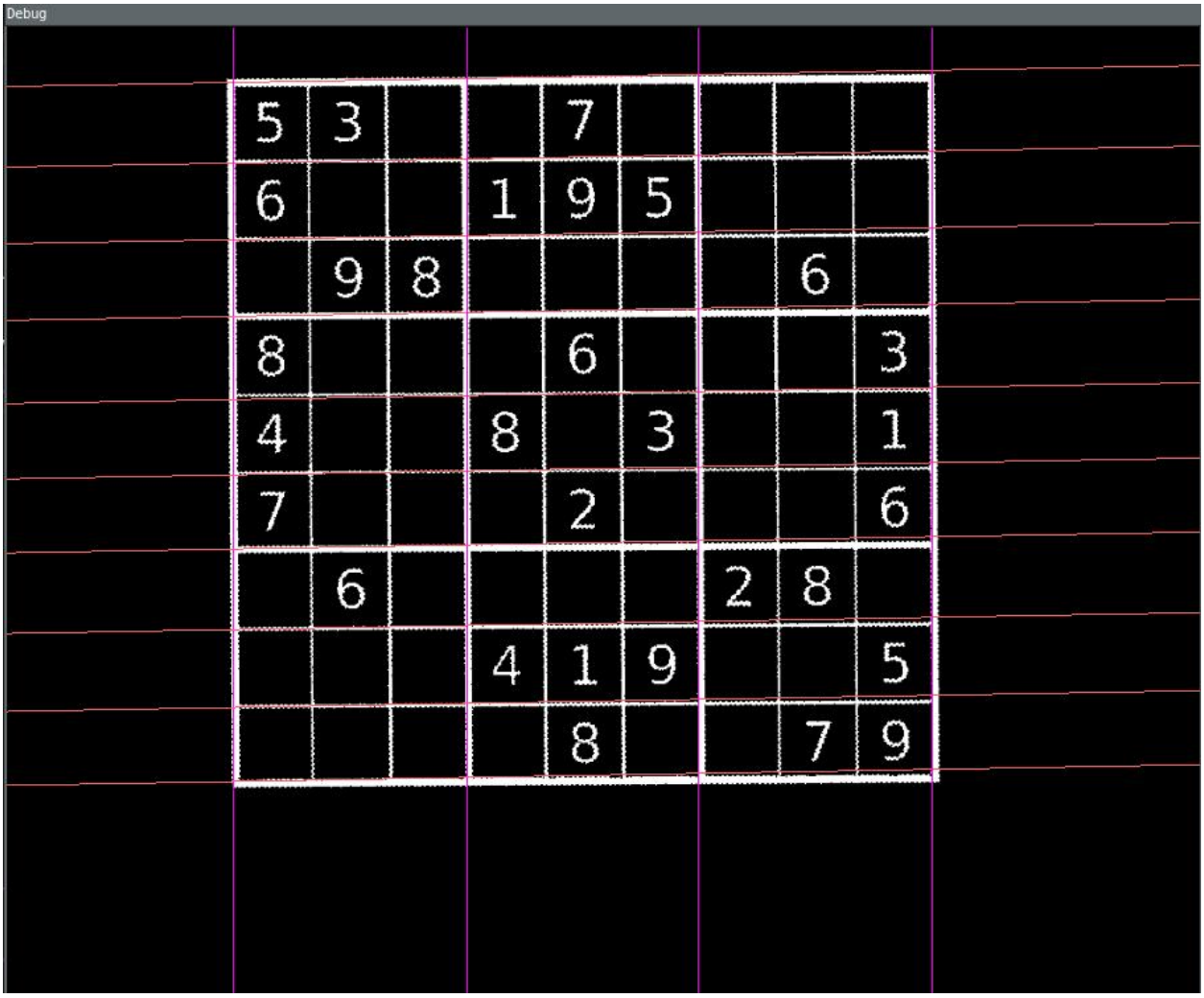
```
int y0 = (p - Rhos) / sin(angleRad);
```

```
int x0 = 1;
int x1 = width - 1;
int y1 = x1 * - cos(angleRad) / sin(angleRad) + y0;
int x2 = width - 1;  // Ending x-coordinate
int y2 = ((p - Rhos) / sin(angleRad)) + 0.3;
```

On ajoute les coordonnées x2 et y2 afin d'obtenir les coordonnées des lignes verticals de sudoku. On filtre également avec le même calcul d'angle, pour stocker séparément les coordonnées x et y. On considère que la rotation s'effectue de manière optimale donc on considère que les x1 (coordonnées x de début de segment) et x2 (coordonnées x de fin de segments)sont égales. De même y0 et y1 sont de même coordonne. On a donc besoin que de deux tableaux de tailles 10 (car 10 lignes) pour stocker les coordonnées de toutes les lignes. Le résultat du traçage des lignes sur la grille de sudoku sur le filtre de sobel:



un autre resultat pour le traçage des lignes avec le fitlre black and white :



les coordonnées x et y trouvées pour l'image ci-dessous.

```
y: 0
y: 43
y: 103
y: 161
y: 218
y: 280
y: 336
y: 392
y: 452
y: 510
y: 566
y: 0

x: 0
x: 171
x: 230
x: 289
x: 348
x: 406
x: 464
x: 523
x: 582
x: 641
x: 700
x: 0
```

3.4 Découpage des cases de la grille

Une fois que toutes coordonnées de chaque ligne horizontales et verticales de la grille sont trouvées on procède au découpage.
Pour le découpage on récupère les coordonnées x et y calculées précédemment et on trouve toutes les intersections grâce à une double boucle. Il faut également connaître la taille de chaque case et pour cela on soustrait les positions x y qui sont voisines.

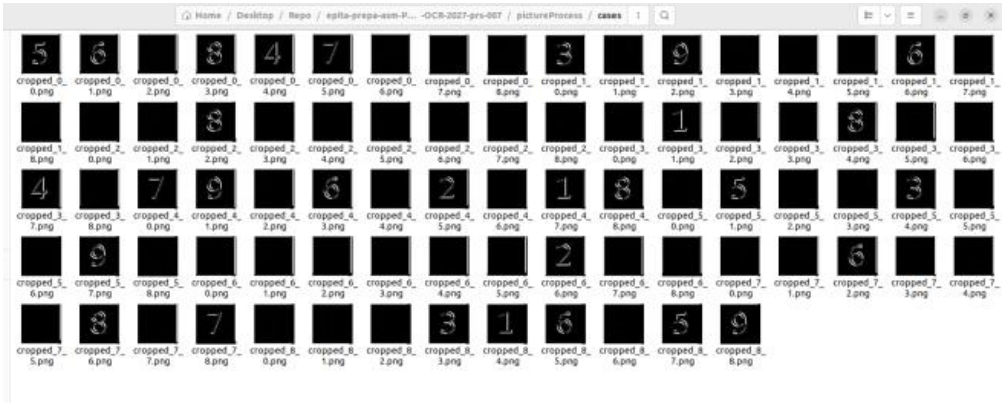
```
for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
            int x = X[j];
            int y = Y[i];

    //taille de la case
            int width = X[i + 1] - X[i];
            int height = Y[i + 1] - Y[i];

    //position de la case
            SDL_Rect rect;
            rect.x = x;
            rect.y = y;
```

Une fois la taille et la position définie on peut découper votre image en créant d'abord une surface puis en utilisant :

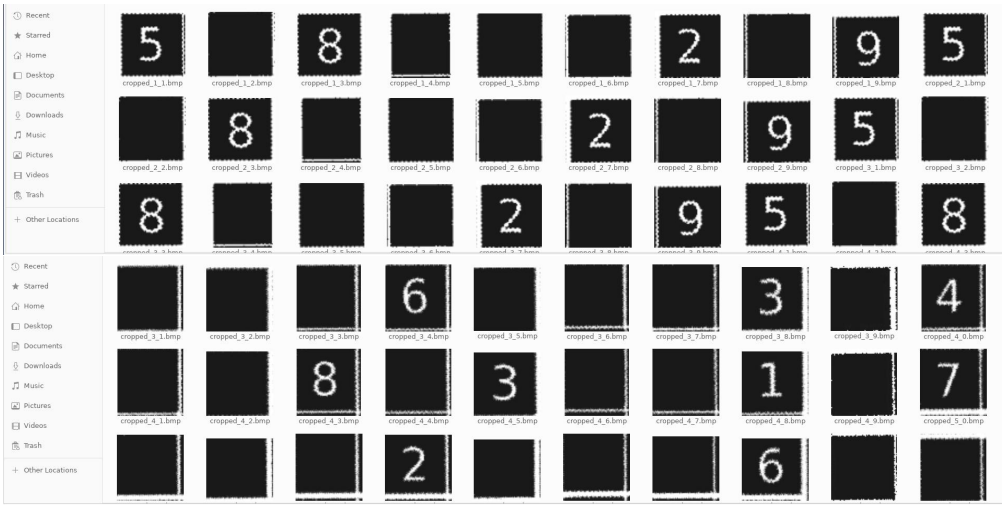
```
// Créer un rectangle découpé
SDL_Surface* croppedImage = SDL_CreateRGBSurface(0, width, height, image->format->BitsPerPixel, ima
SDL_BlitSurface(image, &rect, croppedImage, NULL);
```

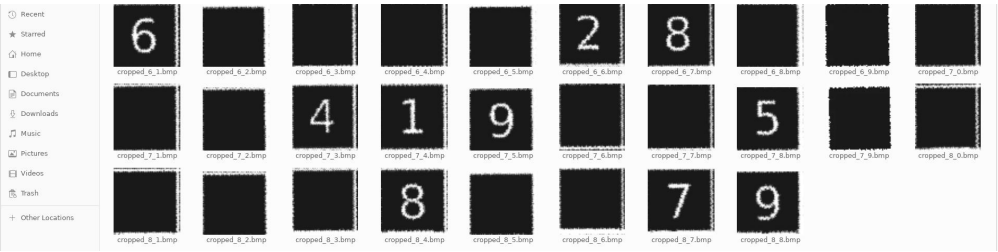
Enfin on enregistre le tout dans un dossier.

Le découpage de la grille une fois les coordonnées trouvées avec le filtre de sobel:



Le découpage de la grille une fois les coordonnées trouvées avec le filtre black and white:





On remarque sur le deuxième exemple qu'il y a encore quelques pixels blancs. Ce sont les lignes blanches de la grille qui sont sur l'image dû à l'imperfection de la rotation et la détection des lignes. Du fait qu'il y est un peu d'espace entre les lignes de la grille et les chiffres, nous allons donc utiliser une fonction nomme crop afin de rogner les images découpées pour être sur de ne pas avoir de pixels qui ne sont pas des chiffres et avoir de s'assurer les résultats pour la reconnaissance des chiffres, le réseau de neurones.

4 Réseau de neurones

4.1 Qu'est-ce que c'est ?

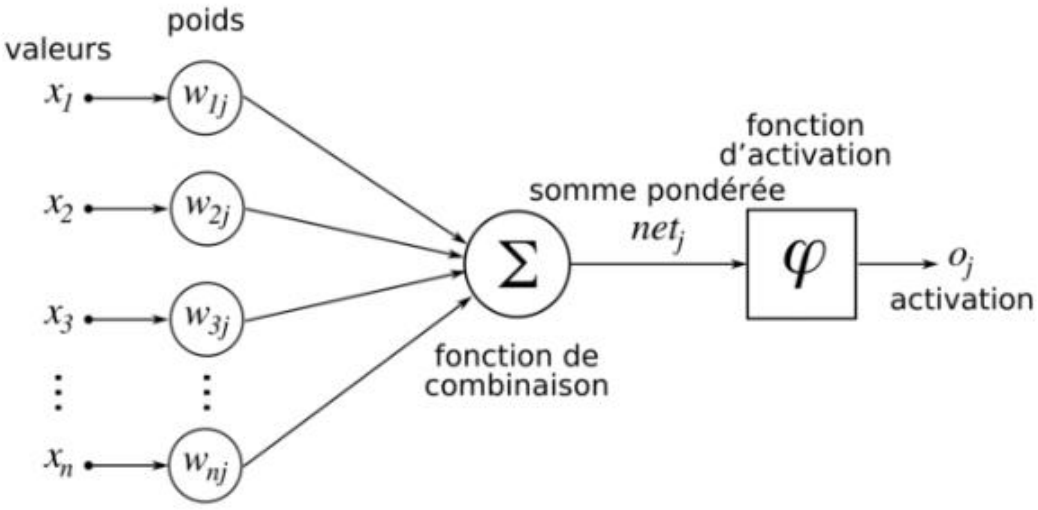
Le réseau de neurones joue un rôle crucial dans les systèmes de reconnaissance optique des caractères. En effet, c'est lui qui permet d'identifier les lettres et de reconstituer le texte désiré. La mise en place de cette composante est complexe, car le réseau fonctionne de manière semblable à un cerveau humain : il peut apprendre pour s'améliorer au fil du temps, produisant ainsi des résultats de plus en plus précis.

Les réseaux de neurones sont structurés en plusieurs couches, comprenant au minimum deux la couche d'entrée et la couche de sortie et une ou plusieurs couches cachées. Les neurones de la couche d'entrée contiennent les informations à tester. Les neurones de la couche de sortie fournissent le résultat de la résolution du problème par le réseau. Les couches intermédiaires servent principalement à effectuer des calculs. Leurs valeurs ne sont exploitées que par le réseau de neurones lui-même.

Les neurones de la couche d'entrée sont connectés à tous les neurones de la première couche cachée, s'il y en a une. Les neurones de cette première couche cachée sont à leur tour connectés à tous les neurones de la couche suivante, et ainsi de suite jusqu'à atteindre la couche de sortie.

Chaque connexion entre deux neurones est associée à un poids qui est utilisé pour effectuer divers calculs. En outre, chaque neurone possède une valeur de

sortie, généralement comprise entre 0 et 1, qui sert à la fois comme donnée d'entrée pour la couche d'entrée et comme résultat d'un calcul pour les autres couches. Cette valeur représente l'information transmise par chaque neurone aux neurones suivants.



4.2 Initialisation

On initialise la structure qui représentent les poids et biais, en créant des listes dynamiques, et les remplir de valeurs aléatoires (entre -1 et 1). Ainsi puisque les listes dynamique et l'instance de réseau sont créées avec des "malloc" on doit implémenter dans notre code une fonction "destroyNetwork" afin de libérer la mémoire des listes, puis de l'instance (l'ordre étant crucial).

```
typedef struct
{
    double* weightsInputHidden;
    double* weightsHiddenOutput;
    double* biasesHidden;
    double* biasesOutput;
    double* hiddenLayerValues;
    double* outputLayerValues;
    int* sizes;

}Network;
```

4.3 L'Apprentissage

Un réseau ne peut donner de bonnes réponses étant donné que ses valeurs initiales sont aléatoires. Le but de l'apprentissage est donc de permettre au réseau de neurones de s'améliorer et de se rapprocher des résultats optimaux.

Ainsi, un processus bien particulier entre en jeu. Tout d'abord on doit essayer de calculer à quel point le réseau est loin de son objectif, on va donc lui donner des données, le laisser les traiter puis regarder les résultats. D'ici on va calculer la marge d'erreur. Une fois la marge d'erreur acquise, le code va se charger d'ajuster les poids et biais de chaque neurones en fonction de leurs impact sur l'erreur. Cette ajustement se fait par deux étapes Calcul en avant et Retro-propagation:
Calcul en avant calcule les valeurs des neurones de la couche cachée et de sortie en utilisant les poids, les biais, et la fonction d'activation sigmoïde. Cela permet de produire les sorties du réseau en fonction des entrées fournies.
La rétropropagation ajuste les poids et les biais du réseau pour minimiser l'erreur entre les sorties réelles et attendues. Cela se fait en calculant les deltas (contributions à l'erreur) pour chaque neurone, puis en mettant à jour les poids et les biais en fonction de ces deltas. Cette mise à jour itérative des paramètres du réseau est au cœur de l'apprentissage des réseaux de neurones. Au fur et à mesure des corrections, le réseau de neurones va s'améliorer et réduire son erreur vis-à-vis de l'output "désirée". C'est à ce moment là seulement que l'on va pouvoir tester notre réseau et voir comment il a réussi à apprendre la fonction en regardant ses output en fonction des inputs données.

Après de longues heures d'entrainement, je me suis rendu compte que mon réseau détectait les images manuscrites et non numériques. A ce moment là, une sensation entre le dégoût et le soulagement m'indiquait qu'il fallait que je j'entraine mon réseau de nouveau sur des images plus adapté mais aussi plus simple à reconnaître. Je me suis alors lancé dans la reconnaissance d'image manuscrite, à succès.

4.4 XOR

Pour la fonction xor notre modèle d'entraînement est le suivant: double allInput[4][2] = 0, 0 | 0, 1 | 1, 0 | 1, 1 | double wantedOutput [4] = 0, 1, 1, 0 ;

Et on teste pour les 4 possibilités d'input avant l'entraînement et après l'entraînement:

```
navi@navi-virtual-machine:~/Desktop/Commun$ ./XORNN
-------------------- AVANT ENTRAINEMENT --------------------
Input: 0.000000 0.000000, Output: 0.089432
Input: 0.000000 1.000000, Output: 0.060031
Input: 1.000000 0.000000, Output: 0.064020
Input: 1.000000 1.000000, Output: 0.046022
-------------------- APRES ENTRAINEMENT --------------------
Input: 0.000000 0.000000, Output: 0.016455
Input: 0.000000 1.000000, Output: 0.978026
Input: 1.000000 0.000000, Output: 0.978140
Input: 1.000000 1.000000, Output: 0.025320
```

4.5 /le réseau de neurones

À la fin de la première soutenance, le réseau de neurones était fonctionnel et efficace pour l'entraînement et l'application de la fonction XOR. L'équipe envisageait alors de remplacer les entrées par des images et de ré-entraîner le réseau, en augmentant le nombre de neurones d'entrée (et de sortie), pensant que cela suffirait pour cette partie du projet.

Cependant, l'intégration d'images dans un réseau de neurones s'est avérée plus complexe en raison du nombre important d'entrées et de données à gérer. Pour le premier réseau, les entrées se limitaient à deux valeurs binaires. Pour le réseau traitant les images, 81 entrées étaient nécessaires, chacune représentant une image de résolution 28x28, soit 784 valeurs binaires. De nombreuses modifications étaient donc requises pour rendre le réseau efficace.

La première étape a consisté à ajuster le nombre de neurones dans le réseau, une tâche simplifiée grâce à une automatisation préalable. Le réseau initial, conçu pour résoudre un problème simple, ne nécessitait pas l'utilisation de biais et fonctionnait correctement uniquement avec des poids ajustés. Cependant, lors de l'entraînement avec des images, il est devenu évident qu'il fallait introduire et mettre à jour ces biais pour assurer un fonctionnement correct. L'ajout de biais et leur mise à jour, bien que simples, étaient cruciaux pour compléter le mécanisme des poids.

L'équipe a ensuite intégré la bibliothèque d'images MNIST, conformément aux exigences du cahier des charges. Après avoir extrait et préparé les données, elles ont été injectées dans une liste pour utilisation. Le nombre d'images à traiter, 60 000 au total, impliquait une série importante d'entraînements. Le traitement de chaque image, comprenant 784 pixels, ainsi que la répétition des opérations sur plusieurs "epochs", nécessitaient un temps considérable, soulignant l'ampleur et la complexité de l'intégration d'images dans le réseau de

neurones.

5 Résolution de la grille

5.1 Résolution

Pour la résolution de la grille, on a utilisé un algorithme qui teste un par un les cases. Pour ce faire, nous avons conçu une première fonction chargée de déterminer si une case peut être remplie par le chiffre "n". On rappelle les règles du sudoku : un chiffre ne peut être placé que s'il n'apparaît pas déjà dans la même ligne, la même colonne et le carré de 3x3 correspondant.

Nous avons créé une fonction (possible) qui vérifie si cette règle est respectée avec ce chiffre “n”. Cette fonction parcourt la ligne, la colonne et le petit carré de 3 fois 3 et vérifie la présence du chiffre “n”. Si ce chiffre est trouvé, alors cette case n'est pas valide avec ce chiffre “n” la fonction renvoie 0 (false). Sinon la case est compatible avec ce chiffre et la fonction renvoie 1 (true). Pour la résolution du sudoku nous avons utilisé une matrice 9 fois 9 pour stocker les valeurs de la grille. Les cases vides dans le sudoku auront comme valeur 0.

```
int possible(short n, size_t x, size_t y){
    for (size_t val = 0; val < 9; val+=1){
        //test sur la ligne
        if (sudoku[val][y] == n){
            return 0;
        }
        //test sur la colonne
        if (sudoku[x][val] == n){
            return 0;
        }
    }
    for (size_t Y = 0; Y < 3; Y += 1){
//calcul de la position de Ro et Co les premières valeurs
//dans le petit carré trois*trois dans laquelle
        //la valeur n se trouve
        size_t Ro = 6;
        size_t Co = 6;
        if(x < 3)
            Ro = 0;
        else
            if(x < 6)
                Ro = 3;
        if(y < 3)
            Co = 0;
        else
            if(y < 6)
                Co = 3;

        for(size_t X = 0; X < 3; X+=1){
            //test sur le carre de trois
            if (sudoku[Ro+Y][Co+X] == n){
                return 0;
            }
        }
    }
    return 1;
}
```

Ensuite nous parcourons le sudoku case par case, si la case est vide (c'est-

à-dire que sa valeur est égale à 0) alors la fonction test les valeurs de 1 à 9 en faisant appel à la fonction "possible". Si la fonction “possible” renvoie 1 alors on place ce chiffre dans la case. Si la fonction renvoie "faux" pour toutes les valeurs de 1 à 9, cela signifie que nous avons fait un mauvais choix précédemment. La fonction revient à la case précédente pour tester d'autres possibilités. Ce processus de retour en arrière s'effectue de manière récursive. On répète ces étapes jusqu'à ce que la grille soit résolue.

```
int solve(){
    for(size_t i = 0; i < 9; i++){       //parcour toutes les cases du sudoku
    for(size_t j = 0; j < 9; j++){

        if(sudoku[i][j] == 0){
            for(short num = 1; num <= 9; num++){
                if (possible(num,i,j)){       //test de la case avec le chiffre num
                    sudoku[i][j] = num;
                    if(solve()){
                        return 1; //appel recursif, si le sudoku est resolu
                        //alors le num est bon sinon le num n'est pas bon
                    }
                    sudoku[i][j] = 0; //la valeur ne marche pas donc
                    // on reniatilise a zero et on continue avec les autres nums
                }
            }
            return 0;
        }

    }
    }
    return 1;
}
```

5.2 Sauvegarde de la grille résolue

Pour sauvegarder la grille résolue, nous avons créé un nouveau fichier, parcouru la matrice résolue, et écrit les valeurs dans ce fichier à l'aide de la fonction "fprintf".
Nous avons également fait une fonction qui affiche la grille de sudoku résolue sous forme d'image. Pour cela nous avons pris une image d'une grille de sudoku vide puis 9 images de chaque chiffres et 9 images de ces chiffres avec un rectangle vert.

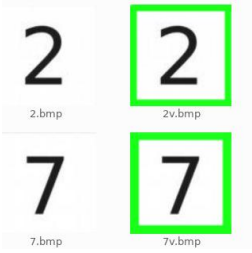
On stocke le nom des images dans une variable et la position de chaque image dans la grille du sudoku vide.

```
const char* imageFiles[] = {"0.bmp","1v.bmp", "2v.bmp", "3v.bmp","4v.bmp", "5v.bmp", "6v.bmp","7v.bmp", "8v.bmp", "9v.bmp"};
const char* imageFiles2[] = {"0.bmp","1.bmp", "2.bmp", "3.bmp","4.bmp", "5.bmp", "6.bmp","7.bmp", "8.bmp", "9.bmp"};

    // Array of coordinates for each image
    int x[9] = {13,120,230,343,450,560,670,780,890};
    int y[9] = {13,120,230,343,450,560,670,780,890};
```

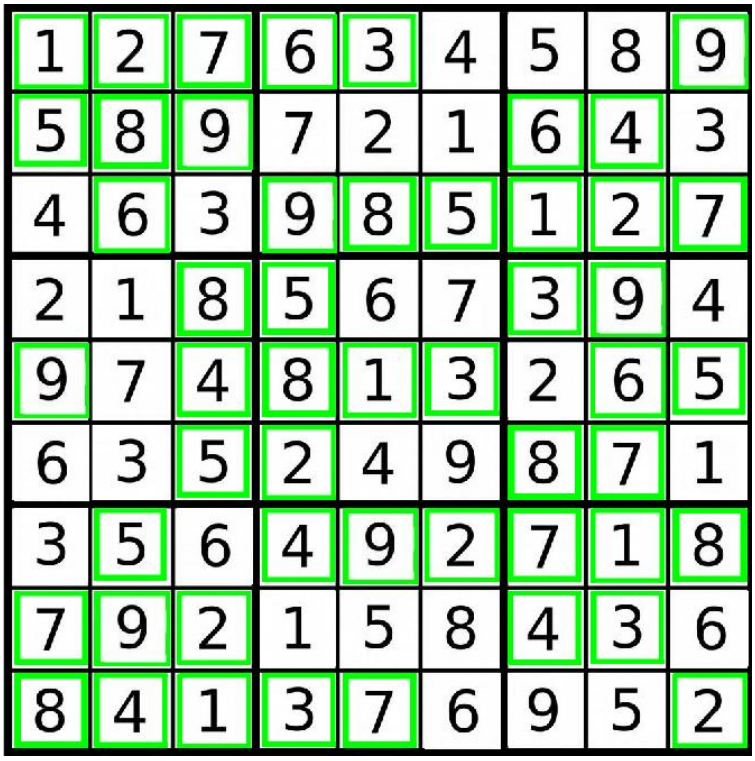
On rappelle que lorsque la fonction solver résout le sudoku, il stocke le résultat dans une matrice. Nous parcourons donc cette matrice et pour chaque chiffre on prend la bonne image du chiffre à l'index du chiffre dans le tableau et on le place dans l'image de la grille vide grâce aux coordonnées stockées dans les deux tableaux.

```
                if(sudoku2[j][i] == 0){

                        image = SDL_LoadBMP(imageFiles[sudoku[j][i]]);
                        // Rectangle source (entire image)
                        SDL_Rect srcrect = {0, 0, image->w, image->h};

                        // Rectangle destination (where to paste the image)
                        SDL_Rect dstrect = {x[i], y[j], 0, 0};

                        // Overlay the image on the background
                        SDL_BlitSurface(image, &srcrect, background, &dstrect);

                        // Free the loaded image surface
                        SDL_FreeSurface(image);
```

On a créé deux types de cases : des cases basiques et d'autres entourées de vert.



La case en vert sera utilisée pour afficher les cases ajoutées à la grille de base. C'est les cases de la "réponse". Pour afficher l'un ou l'autre nous avons copié la matrice non résolu. On parcourt cette matrice. Si elle est égale à 0 c'est a dire case vide alors on place le chiffre de la matrice résolu en vert sinon en basique. voici le resultat de l'affichage du soduku resolu :



6 Interface graphique

6.1 Interface graphique

La fonction `on_solve` est un gestionnaire d'événements associé à un bouton de l'interface graphique, orchestrant une suite d'opérations complexe comprenant le traitement d'images, la résolution de grilles de sudoku et l'utilisation d'un réseau de neurones. Voici un aperçu détaillé de son processus :

La fonction débute par vérifier si l'image chargée (`image_surface`) est nulle. Si une erreur survient lors du chargement de l'image, un message d'erreur est affiché, les bibliothèques SDL et IMG sont fermées, et la fonction se termine.

Ensuite, la fonction applique un algorithme de transformation de Hough sur l'image chargée pour détecter des lignes. Ces lignes sont dessinées sur l'image grâce à la fonction `draw_lines`. L'image résultante est redimensionnée et affichée dans un widget image de l'interface graphique. Parallèlement, l'image est sauvegardée au format BMP sous le nom `"test.bmp"`. La fonction procède ensuite à la découpe de l'image sauvegardée (`"test.bmp"`) et arrête l'environnement SDL.

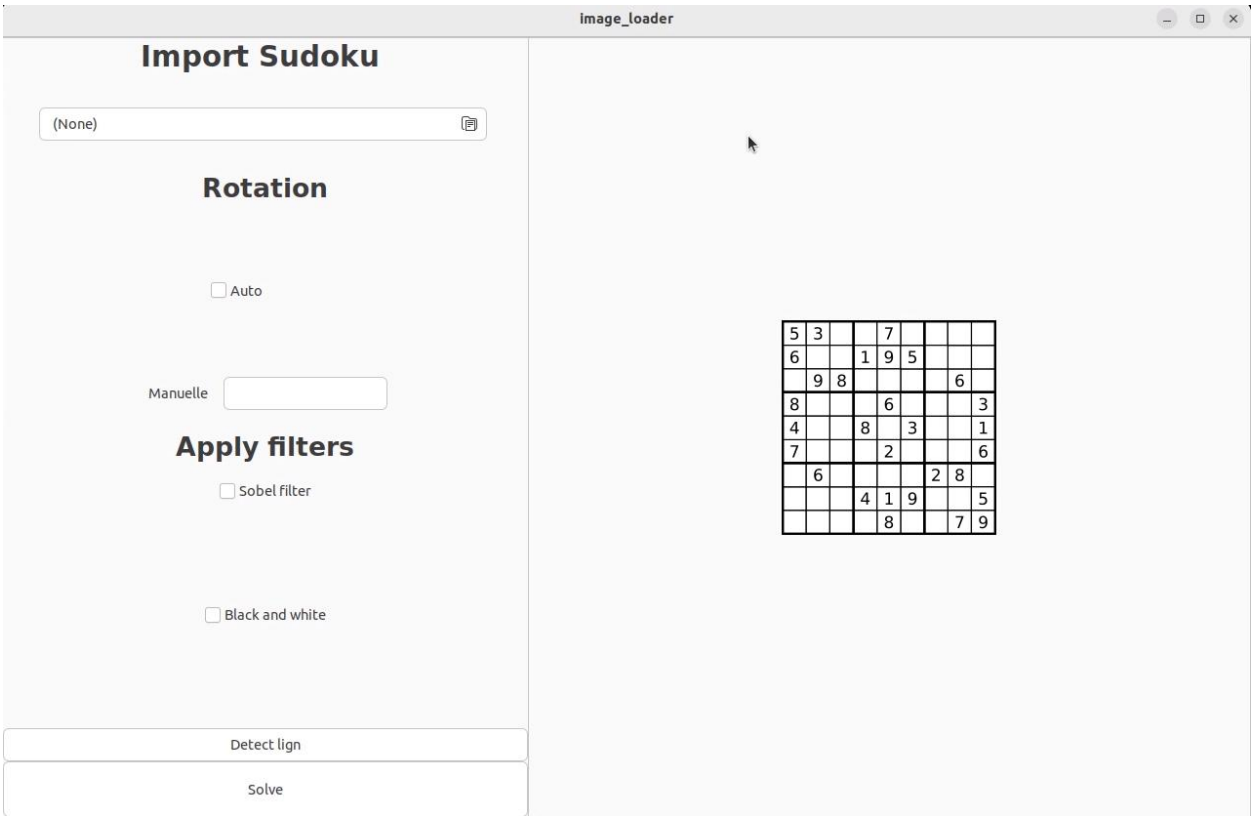
Puis, des structures de données sont initialisées pour créer et entraîner un réseau de neurones. Un tableau nommé `sizes` définit l'architecture du réseau, et différentes fonctions sont appelées pour traiter des images, initialiser le réseau et afficher la grille du réseau.

Après avoir libéré la mémoire allouée aux données du réseau, la fonction continue en lisant une grille de sudoku depuis un fichier (`"grid_00"`) pour remplir une matrice. La grille est résolue par la fonction `solve`, et si la résolution est un succès, l'image résultante est affichée dans l'interface graphique. En conclusion, un message indiquant la validité de la grille de sudoku résolue est affiché, et l'environnement SDL est fermé.

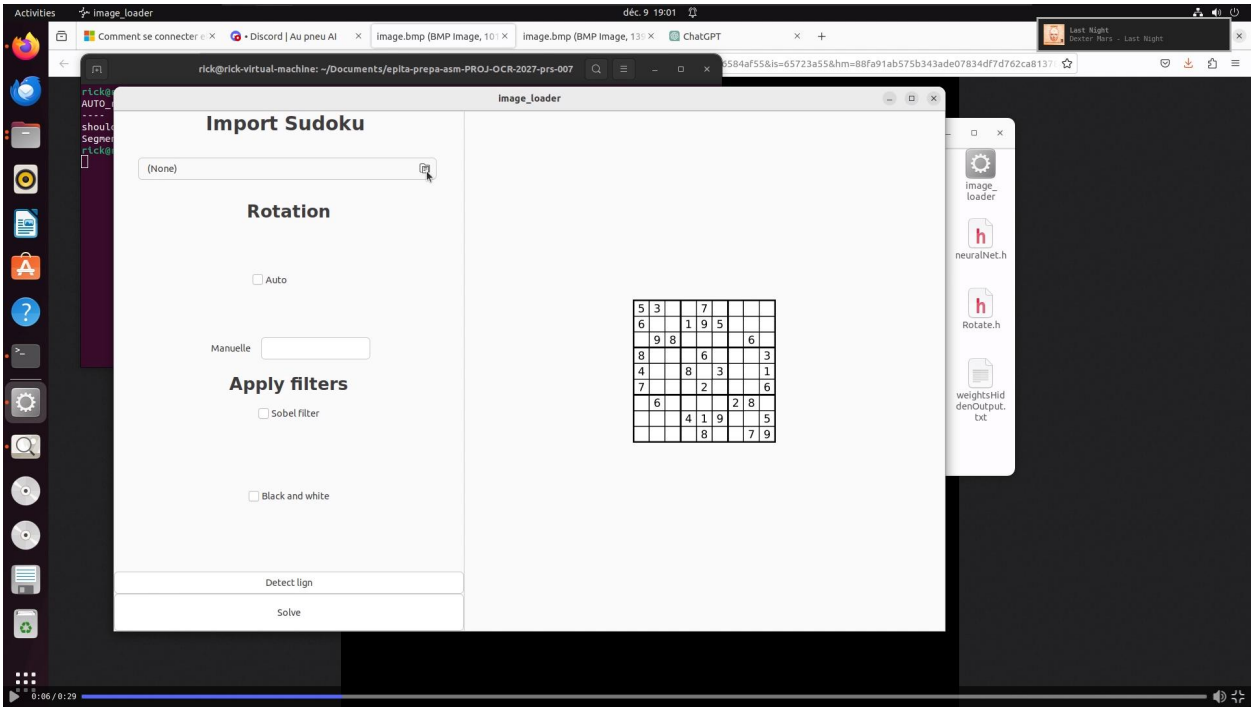
Pour résumer, la fonction `on_solve` englobe une suite d'opérations complexes, allant du traitement d'images et la détection de lignes, à la résolution de grilles de sudoku, en passant par l'utilisation d'un réseau de neurones, jusqu'à l'affichage des résultats dans l'interface graphique GTK.

6.2 Le résultat en imgage :

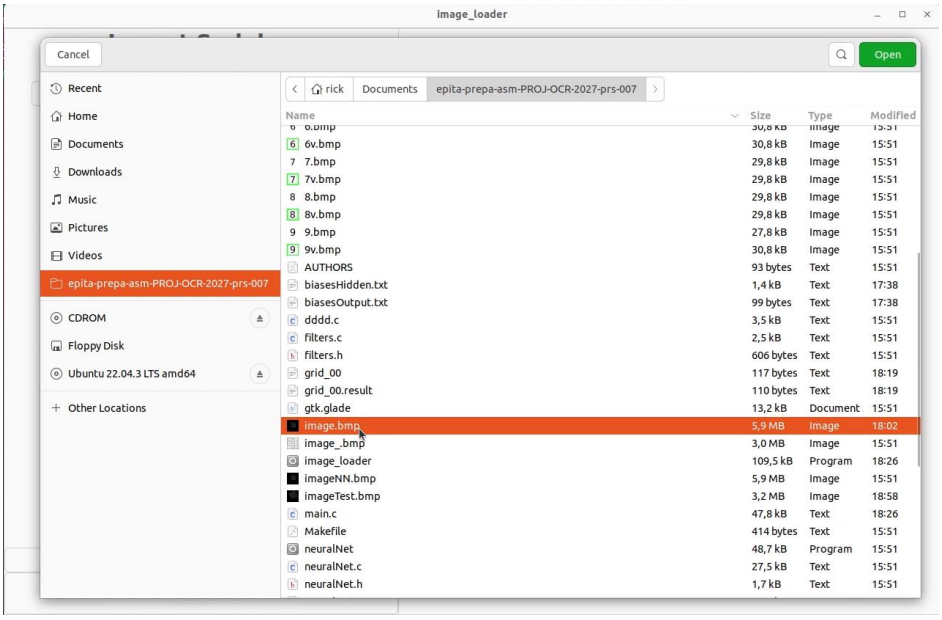
Tout d'abord voici a quoi ressemble l'interface :



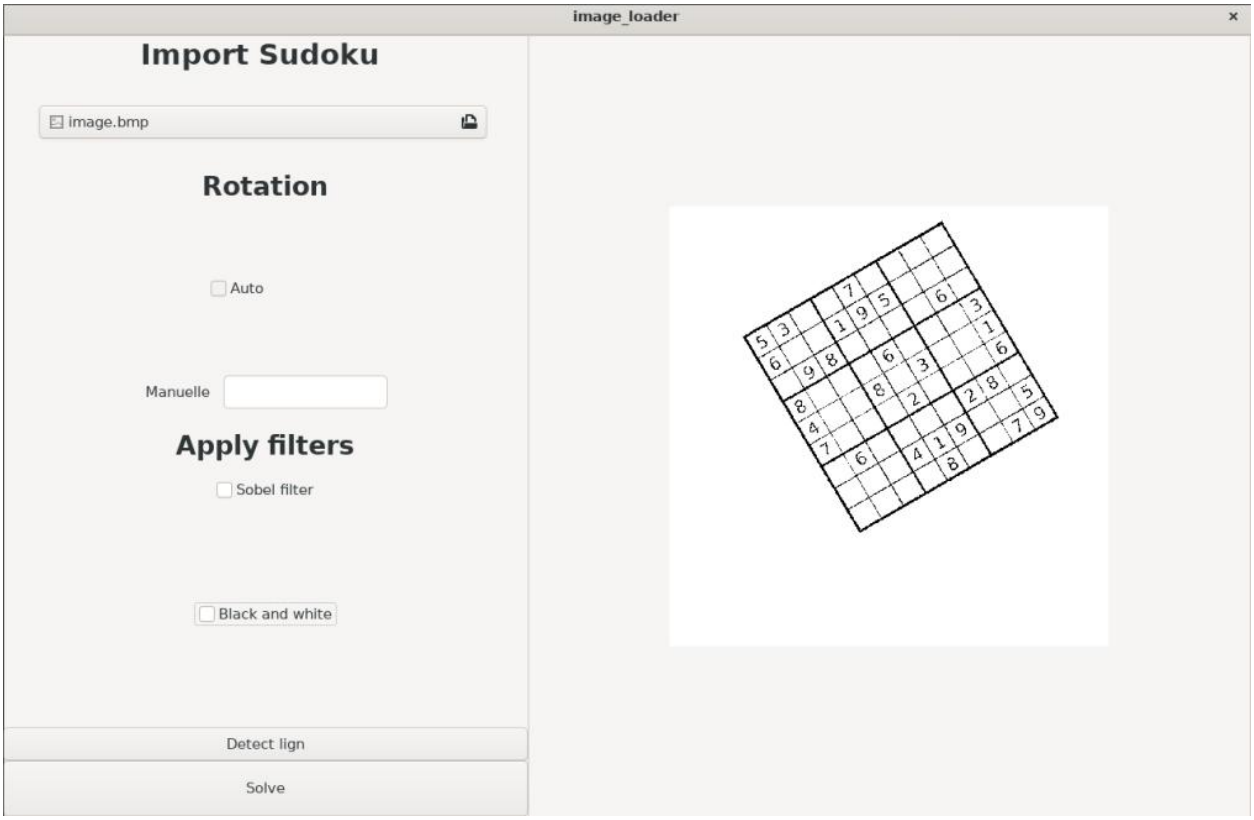
pour charger une image on clique sur l'icone tout en haut :



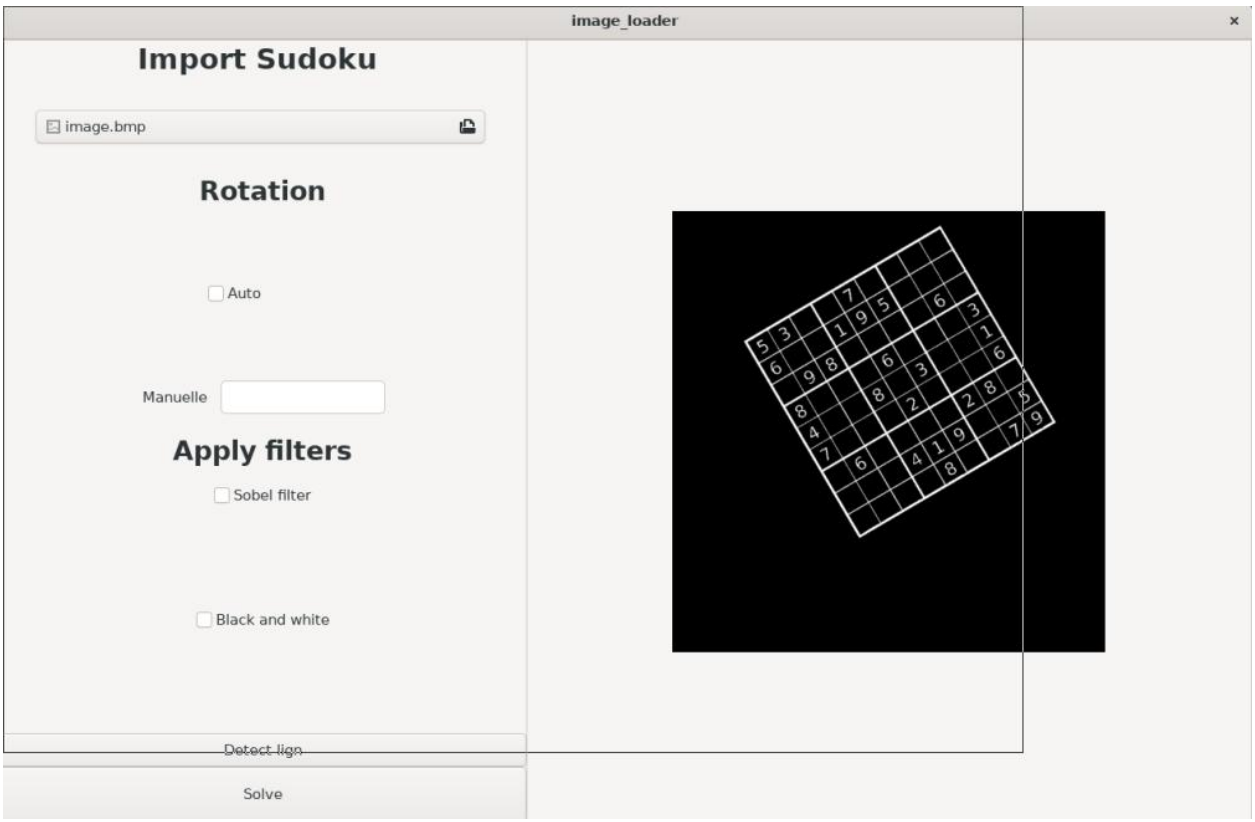
on selectionne l'image du sudoku :


image_loader
Cancel
Open
Recent
Home
Documents
Downloads
Music
Pictures
Videos
epita-prepa-asm-PROJ-OCR-2027-prs-007
CDROM
Floppy Disk
Ubuntu 22.04.3 LTS amd64
Other Locations
rick
Documents
epita-prepa-asm-PROJ-OCR-2027-prs-007
Name
Size
Type
Modified
6v.bmp 30,8 kB Image 15:51
7.bmp 29,8 kB Image 15:51
7v.bmp 29,8 kB Image 15:51
8.bmp 29,8 kB Image 15:51
8v.bmp 29,8 kB Image 15:51
9.bmp 27,8 kB Image 15:51
9v.bmp 30,8 kB Image 15:51
AUTHORS 93 bytes Text 15:51
biasesHidden.txt 1,4 kB Text 17:38
biasesOutput.txt 99 bytes Text 17:38
dddd.c 3,5 kB Text 15:51
filters.c 2,5 kB Text 15:51
filters.h 606 bytes Text 15:51
grid_00 117 bytes Text 18:19
grid_00.result 110 bytes Text 18:19
gtk.glade 13,2 kB Document 15:51
image.bmp 5,9 MB Image 18:02
image_.bmp 3,0 MB Image 15:51
image_loader 109,5 kB Program 18:26
imageNN.bmp 5,9 MB Image 15:51
imageTest.bmp 3,2 MB Image 18:58
main.c 47,8 kB Text 18:26
Makefile 414 bytes Text 15:51
neuralNet 48,7 kB Program 15:51
neuralNet.c 27,5 kB Text 15:51
neuralNet.h 1,7 kB Text 15:51

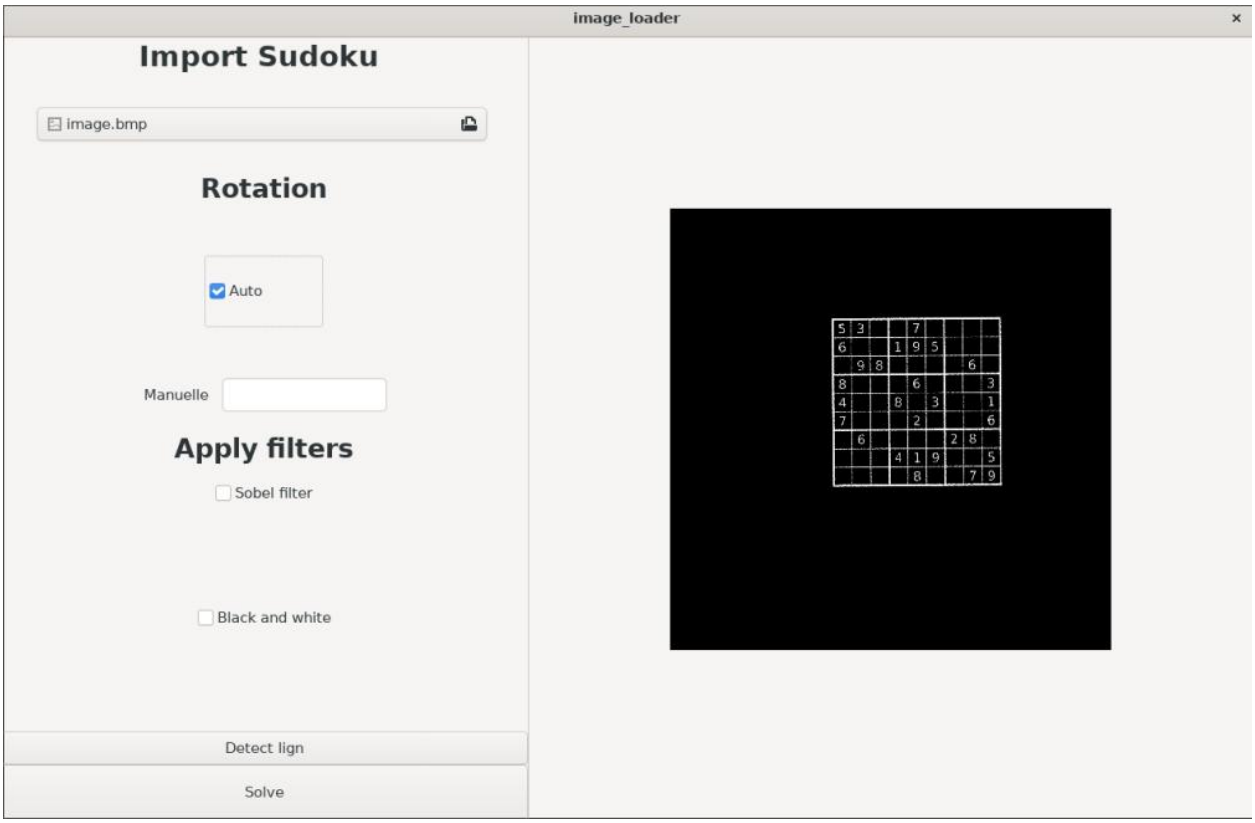
quand on charge l'image on obtient :



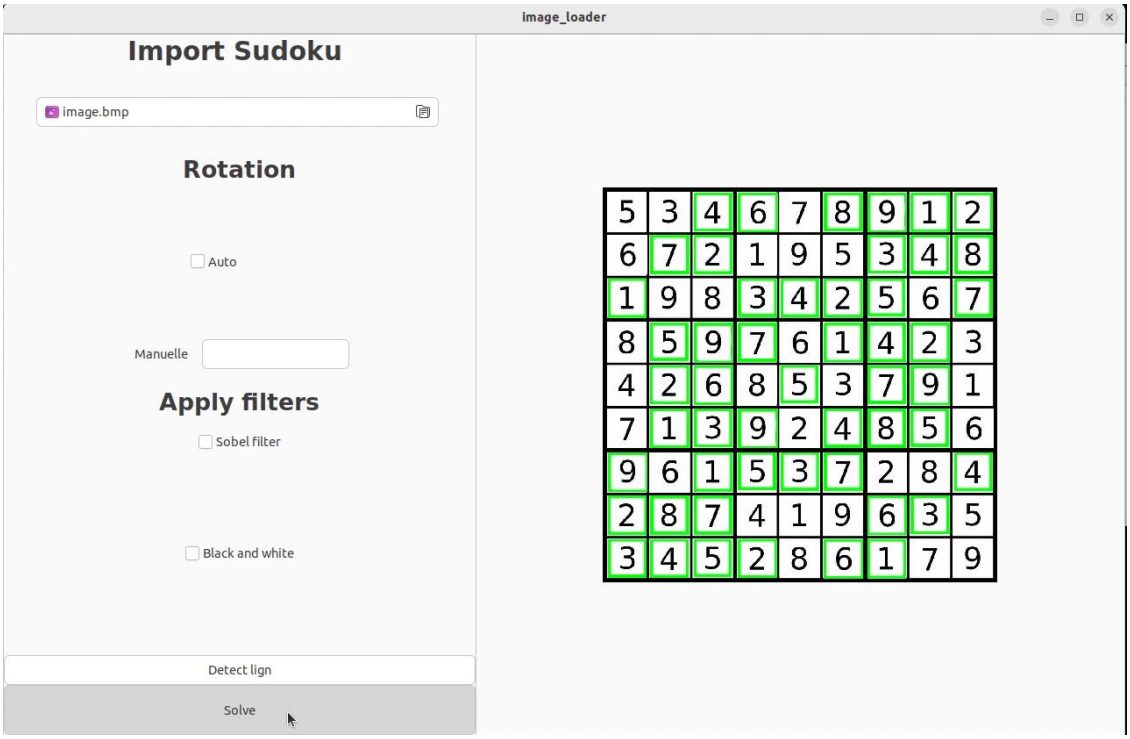
Lorsqu'on appuie sur black and white :



pour faire la rotation automatique on coche l'icone auto pour la rotation manuelle, on entre l'angle de rotation dans l'input Lorsqu'on appuie sur auto :



puis enfin on appuie sur solve et hop le resultat s'affiche! On peut egalement cliquer sur detect line pour verifier le decoupage.



7 Conclusion

7.1 Tableau d'avancement

	Intermediaire	Final
Prétraitement	70%	90%
Rotation	100%	100%
Reconnaissance de la grille	60%	70%
Découpage de la grille	95%	100%
Réseau de neurones	20%	100%
Resolution du sudoku	95%	100%
Interface Graphique	0%	90%

7.2 Conclusion

Ce document présente les réalisations atteintes au cours de ce semestre pour le projet de résolution de Sudoku en langage C. Grâce à ce projet nous avons acquis de nouvelles connaissances intéressantes d'un point de vue algorithmique. Nous avons constaté avec satisfaction les avancées réalisées jusqu'à présent.

Ce document présente les réalisations atteintes durant ce semestre pour le projet de résolution de Sudoku en langage C. Ce projet, bien plus qu'un simple exercice de programmation, s'est révélé être une aventure enrichissante.
Grâce à ce projet nous avons acquis de nouvelles connaissances intéressantes tels que les concepts de traitement d'images, de reconnaissance optique de caractères (OCR) et d'intelligence artificielle.

Nous avons exploré diverses algorithmes, tels que l'application des filtres de Sobel pour la détection des bords, les rotations manuelles et automatiques des images, et les approches de binarisation pour simplifier les images. L'intégration de ces techniques a été cruciale pour préparer les images pour la reconnaissance des chiffres, une tâche réalisée grâce à un réseau de neurones entraîné.

Le développement de l'interface utilisateur avec GTK a été un autre aspect significatif de notre apprentissage.

Enfin, ce projet a renforcé notre capacité à travailler en équipe et à résoudre des problèmes.

En conclusion, le projet OCR Sudoku Solver a été une expérience formatrice, nous dotant de compétences techniques avancées et de précieuses leçons en matière de travail d'équipe et de gestion de projet. Nous sommes fiers des réalisations accomplies malgré les quelques points qu'on aurait pu mieux faire.

Merci pour la lecture.