Rapport de soutenance 1
Projet de programmation - Semestre 3

OCR Sudoku Solver

Créé par :
Olympe Cabande - Chef de projet
Celio Cucchiara
Nathan Auguie
Marin Godefroy

EPITA Toulouse

6 novembre 2023

Table des matières

1 Introduction **1**

2 Répartition des tâches **2**

3 Traitement-Image **3**

3.1 Conversion en gris 3

3.2 Filtrage Gaussien 4

3.3 Morphologie 5

3.4 Contraste/Gamma 6

3.5 Seuillage Otsu 7

3.6 Rotation de l'image 8

3.7 Détection des contours 10

4 Fractionnement-Image **13**

4.1 Détection de ligne 13

4.2 Détection et sélection du carré 16

4.3 Recadrage du carré/Fractionnement de la grille 18

5 Solveur de sudoku **19**

6 Réseau de neurones **20**

7 Planning **22**

8 Conclusion **23**

1 Introduction

La reconnaissance optique de caractère (de son abréviation anglaise OCR pour Optical Character Recognition), est un système qui permet de détecter, à partir d'une image standard, un texte (soit non éditable) pour pouvoir le sauvegarder et ainsi le modifier. Ce projet est composé de deux grandes parties : les éventuels traitements d'images ainsi que d'autres opérations sur ces dernières, puis la reconnaissance des caractères par un réseau de neurones. L'objectif du projet est de pouvoir coder un algorithme en langage C qui aura à la fin une utilisation facilitée grâce à une interface graphique. Ce projet est réalisé dans le domaine des études du cycle préparatoire de deuxième année à Epita.

Nous présentons, dans ce rapport de soutenance, tout le progrès réalisé dans notre projet dans cette première période de travail, comment les tâches ont été réparties, et les objectifs a atteindre pour la deuxième soutenance. Ainsi que les difficultés rencontrées.

2 Répartition des tâches

Voici la répartition prévue des tâches entre les étudiants du groupe.

Tâches	Olympe	Celio	Nathan	Marin
Traitement-d'image	√			
Fractionnement-Image				√
Solveur de sudoku			√	
Réseau neuronal XOR		√		

√ - Responsable

3 Traitement-Image

3.1 Conversion en gris

La première étape est de convertir la couleur de l'image en gris, pour celà on utilisera la fonction "grayscale()" ci-dessous :

```
static void _grayscale(Pixel *pixel, int brightness)
{
    pixel->r = pixel->g = pixel->b = clamp(
        (0.40 * pixel->r + 0.35 * pixel->g + 0.25 * pixel->b) + brightness, 0,
        255);
}
```

FIGURE 1 – *Fonction Grayscale*

Voici, ci-dessous un exemple de grille de sudoku convertit en gris :

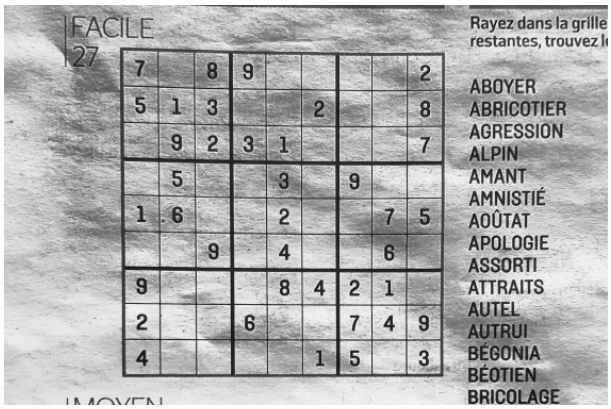

FIGURE 2 – *Exemple de conversion en gris*

3.2 Filtrage Gaussien

Nous devons maintenant atténuer les bruits parasites de l'image et uniformiser les couleurs pour faciliter le traitement ultérieur. Le processus de ce filtre est simple. Pour chaque pixel de l'image, nous calculons une nouvelle valeur issue de la somme du produit des pixels voisins et d'une matrice gaussienne. Cette matrice est générée à l'aide d'une fonction gaussienne, qui se présente avec la formule suivante :

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

FIGURE 3 – *Fonction gaussien pour deux dimensions*

On peut illustrer le filtrage gaussien à l'aide de la courbe ci-dessous :

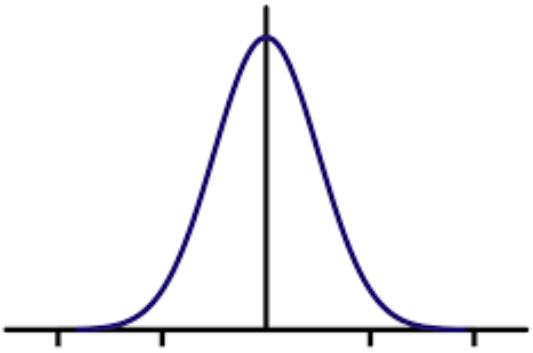

FIGURE 4 – *Filtre de Gauss*

3.3 Morphologie

Dans cette partie nous allons supprimer le bruit de l'image, pour rendre la reconnaissance des caractères et de la grille plus clair. Nous utiliserons les opérateurs Dilatation et Erosion. Nous les utilisons ici pour supprimer le bruit global des images et affiner les lignes et les chiffres des grilles. Ces opérateurs sont souvent utilisés simultanément. On commence par dilater l'image, ce qui agrandit les détails de l'image et rend ainsi les caractéristiques comme les lignes et les caractères plus clair. Nous l'érodons ensuite, pour supprimer la largeur ajoutée aux détails et restaurer leur taille d'origine. Le résultat est une image avec les mêmes caractéristiques principales, mais avec moins de bruit. Ces deux opérations nous permettent de préparer l'image pour l'étape suivante en nous donnant des images avec un arrière-plan uniforme et des traits nets par-dessus.

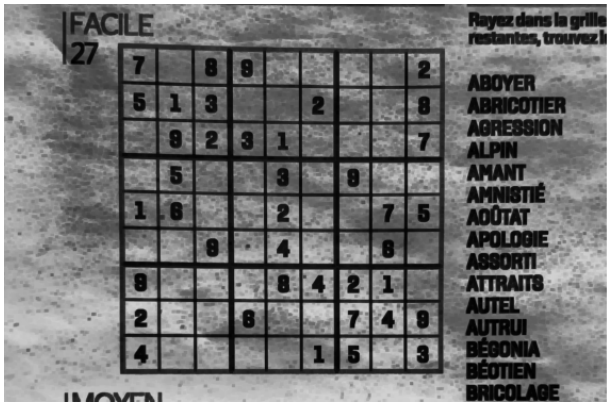

FIGURE 5 – *Dilatation*

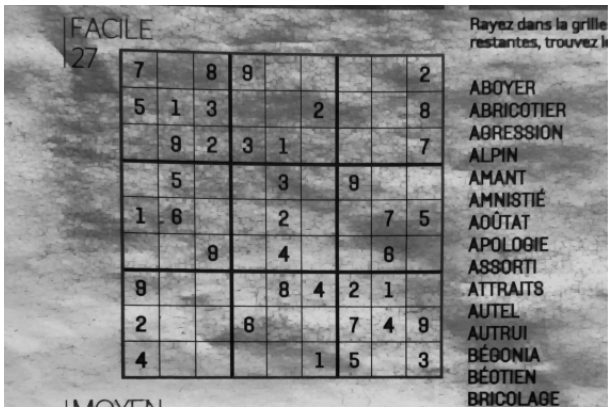

FIGURE 6 – *Erosion*

3.4 Contraste/Gamma

Le réglage du Gamma de l'image permet d'accentuer l'écart entre les parties claires et sombres de l'image. La formule utilisée est la suivante :

$$color = \frac{color^{\frac{\gamma}{128}}}{255}$$

FIGURE 7 – *Formule*

```
void apply_mask(Image *image, Image *mask)
{
    for (int x = 0; x < image->width; x++)
    {
        for (int y = 0; y < image->height; y++)
        {
            if (!mask->pixels[x][y].r)
            {
                image->pixels[x][y].r = image->pixels[x][y].g
                    = image->pixels[x][y].b = 255;
            }
        }
    }
}
```

FIGURE 8 – *Fonction d'application du masque*

3.5 Seuillage Otsu

Pour cette étape nous devons nous occuper de la binarisation de l'image. La méthode classique se base sur une valeur de seuil statique de 128. Si l'intensité est inférieure ou égale à 128, alors le pixel devient noir, dans l'autre cas il devient blanc. Mais cette valeur ne peut pas s'appliquer à toutes les images. Il faut donc calculer la bonne valeur de seuil en fonction de l'histogramme de l'image. C'est ici que Otsu intervient : L'algorithme suppose alors que l'image à binariser ne contient que deux classes de pixels (c'est-à-dire le premier plan et l'arrière-plan) puis calcule le seuil optimal qui sépare ces deux classes afin que leur variance intra-classe soit minimale. En d'autres termes, l'algorithme va chercher la meilleure probabilité de valeur efficace du seuil.

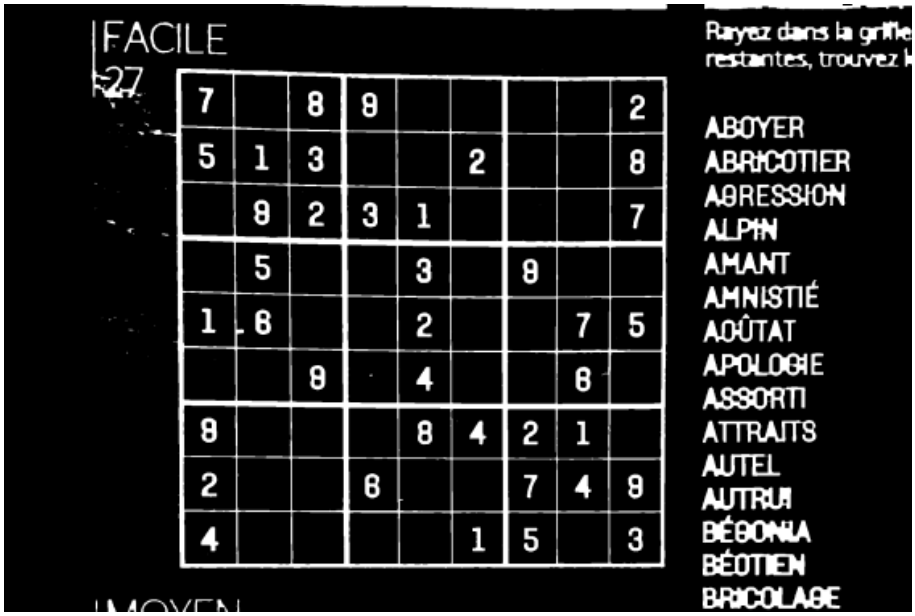

FIGURE 9 – *Grille après seuillage*

3.6 Rotation de l'image

Pour faire pivoter l'image, nous avons décidé d'utiliser une matrice de rotation. Cette méthode est rapide et précise, nous donnant des résultats sans bords irréguliers et nous permettant d'effectuer des rotations avec des angles très précis.

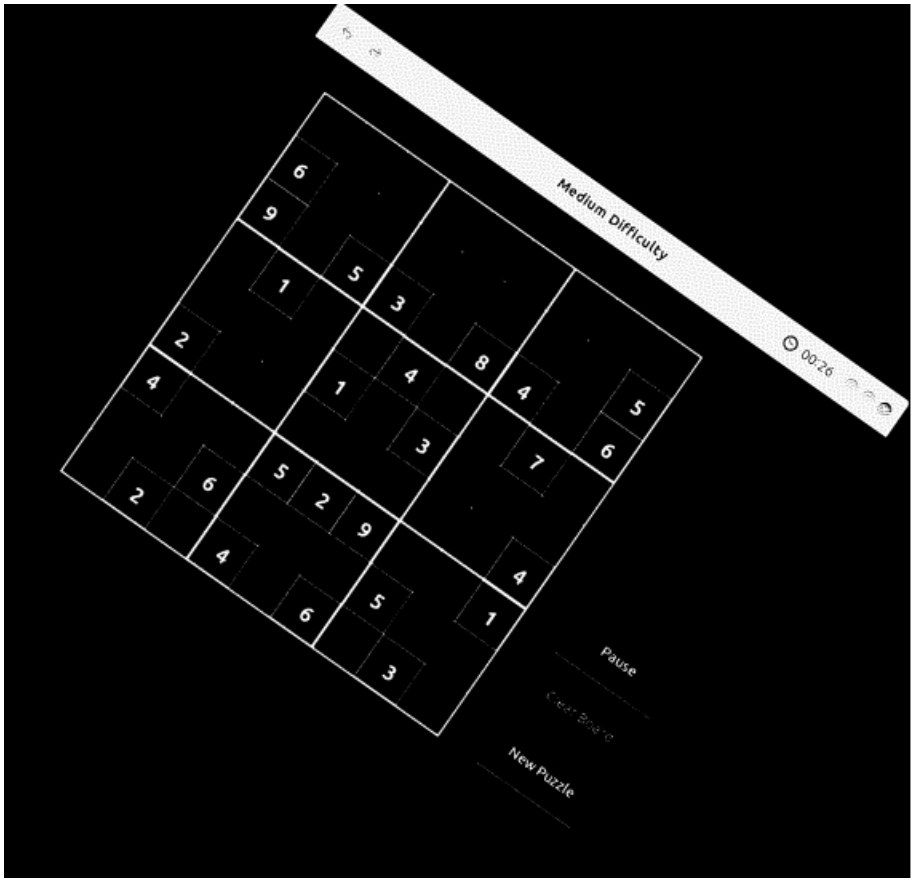

FIGURE 10 – *Avant rotation*

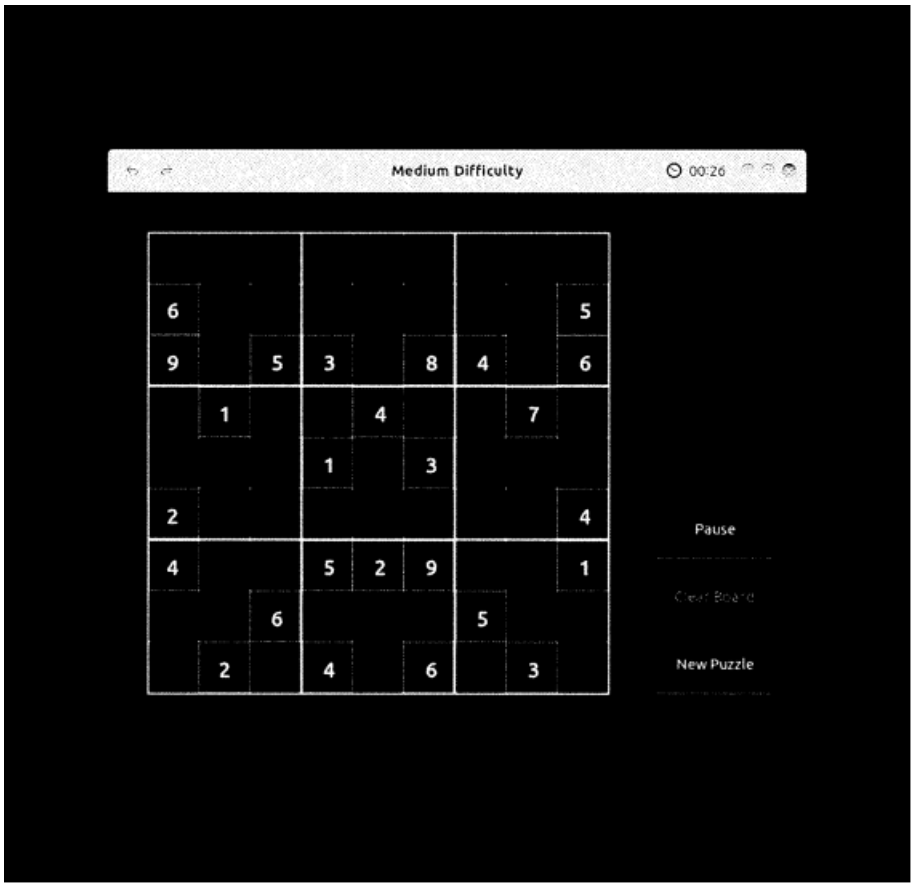

FIGURE 11 – *Après rotation*

3.7 Détection des contours

Pour cette partie nous allons utiliser un algorithme bien connu, le "Filtre de Canny" ou "Détecteur de Canny", qui est utilisé en traitement d'images pour la détection des contours. La première étape est la réduction du bruit qui a déjà été faite auparavant avec le filtre gaussien. Après le filtrage, l'étape suivante est d'appliquer un gradient qui renvoie l'intensité des contours. L'opérateur utilisé permet de calculer le gradient suivant les directions X et Y, il est composé de deux masques de convolution.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

FIGURE 12 – *Gradient en x (Gx)*

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

FIGURE 13 – *Gradient en y (Gy)*

Nous calculons ensuite le gradient de magnitude, qui représente la présence d'une arête sur l'image :

$$G = \sqrt{(G_x)^2 + (G_y)^2}$$

FIGURE 14 – *Formule du gradient de magnitude*

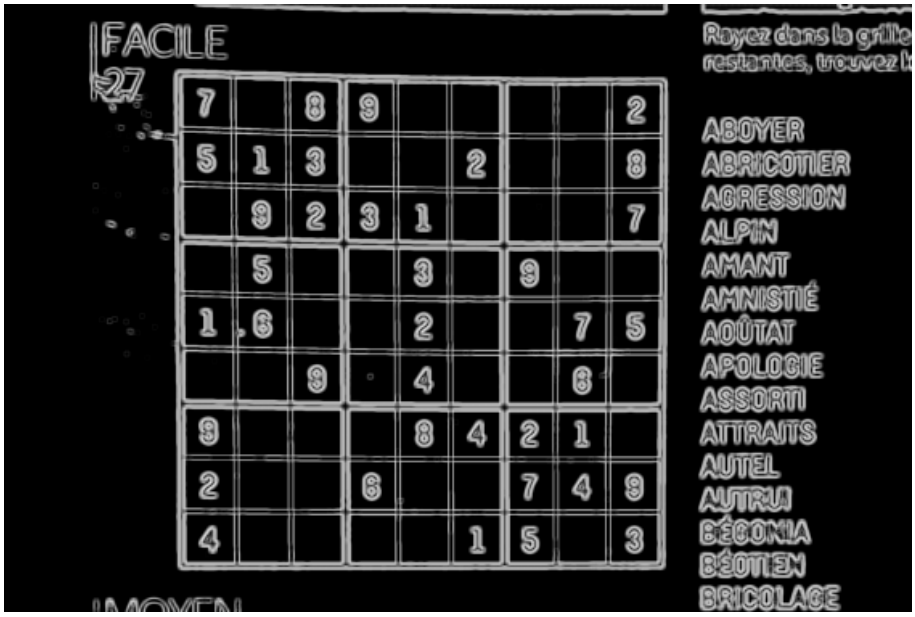

FIGURE 15 –

Nous allons maintenant affiner les bords trouvés dans le gradient de magnitude. Les algorithmes utilisés sont longs et peu intéressants, nous n'allons donc pas reprendre leur processus pour conserver le longueur de ce rapport.

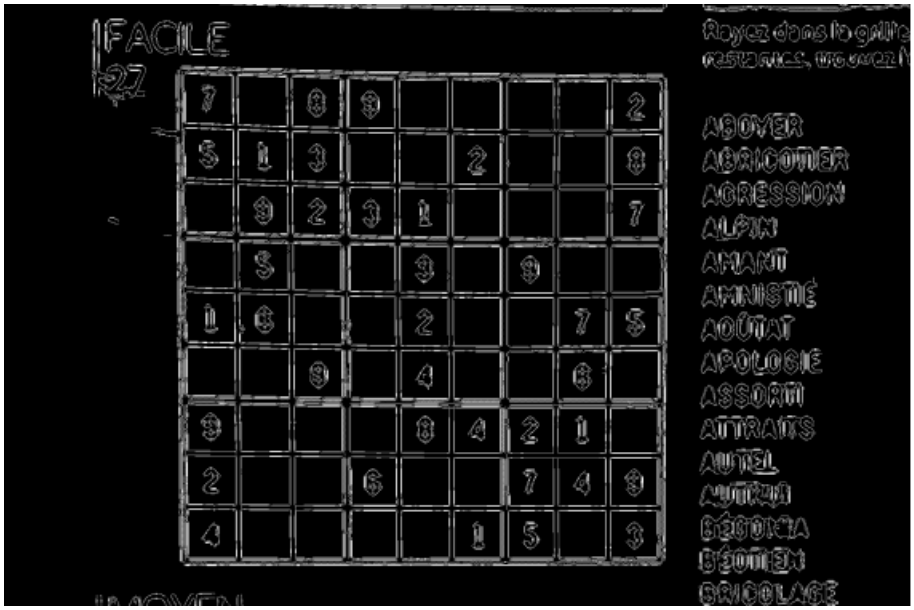

FIGURE 16 – *Filtre de Canny*

Le résultat de la détection astucieuse des contours est une image dans laquelle tous les contours possibles de l'image sont représentés par un pixel blanc. Cette image peut ensuite être utilisée avec l'algorithme de détection de ligne pour trouver les coordonnées de ces arêtes.

4 Fractionnement-Image

4.1 Détection de ligne

Pour détecter et récupérer les coordonnées de la ligne sur l'image, nous utilisons l'algorithme de transformation "Hough". L'algorithme comporte donc trois étapes :

- pour chaque point de contour détecté, détermination de la courbe correspondante dans l'espace des paramètres ;
- construction de la matrice d'accumulation à partir de ces courbes ;
- détection de pics dans la matrice d'accumulation.

Cet algorithme mappe les points de bord d'une image filtrée sur le Hough Espace pour représenter les lignes. Dans l'espace cartésien (l'espace de l'image du bord), les lignes sont représentées sous la forme de cette équation :

$$y = ax + b$$

FIGURE 17 –

Pour les besoins de l'algorithme, chaque point blanc sur l'image de bord est considéré comme une ligne. Chaque ligne sur l'image du bord crée un point sur l'espace de Hough, de coordonnées (rho,theta), avec :

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

FIGURE 18 –

theta allant de -90 degrés à 90 degrés. Cette formule est utilisée pour voter dans un accumulateur. Chaque point blanc de l'image de bord crée un vote dans la même région que les autres points situés sur les mêmes lignes. Ces votes sont ensuite utilisés pour trouver les lignes les plus dominantes sur l'espace Hough en trouvant les lignes locales maximums pour chaque région qui dépasse un seuil arbitraire (70 pourcents du nombre de voix maximum). L'accumulateur mentionné crée l'image suivante une fois tracée et normalisée. Les maximums locaux qu'il faut trouver pour avoir les droites dominantes sont représenté par les zones blanches où se rencontrent toutes les courbes grises.

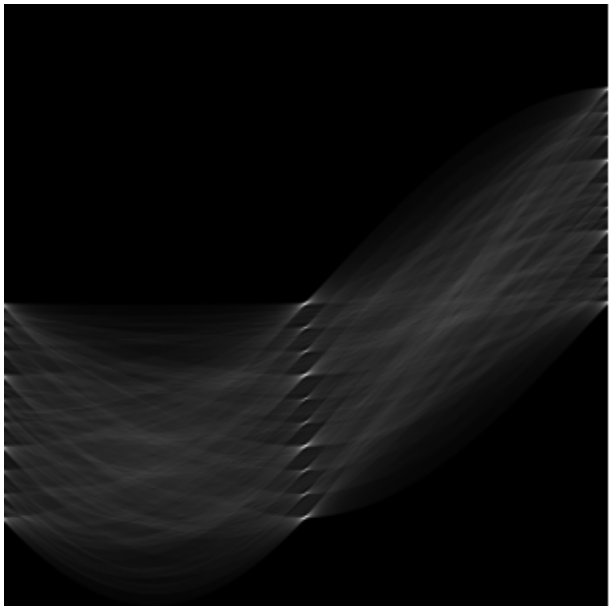

FIGURE 19 – *Accumulateur normalisé*

Lorsque les maximums locaux des coordonnées (rho, theta) sont trouvés, la formule suivante permet de trouver les coordonnées cartésiennes (x0, y0) d'un point situé sur la droite représentée au maximum :

$$x0 = \rho * \cos\theta$$
$$y0 = \rho * \sin\theta$$

FIGURE 20 –

Nous pouvons alors ajouter une distance arbitraire et le même theta pour trouver un deuxième ensemble de coordonnées situées sur la ligne :

$$x1 = x0 + d * (-\sin\theta)$$
$$y1 = x0 + d * \cos\theta$$

FIGURE 21 –

Enfin, lorsque les coordonnées des lignes sélectionnées sont trouvées, on peut les dessiner sur l'image. Comme vous pouvez le constater, les résultats sont assez précis. Cependant, il reste encore de nombreuses lignes et nous devons faire la moyenne pour n'en obtenir que quelques-unes dominantes.

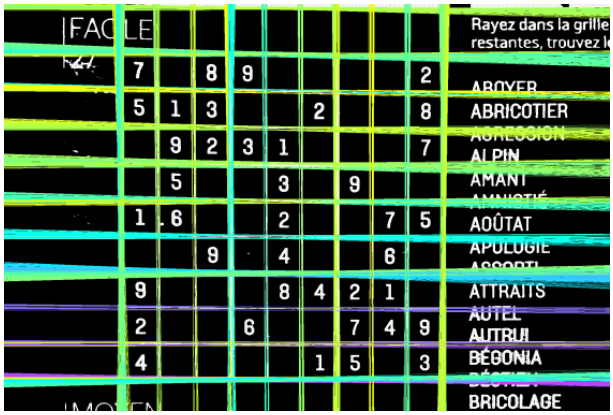

FIGURE 22 – *Lignes de transformation de Hough converties en coordonnées cartésiennes et dessinées sur l'image*

4.2 Détection et sélection du carré

Comme dit précédemment, nous devons faire la moyenne des lignes trouvées pour n'obtenir que les dominantes et accélérer nos calculs ultérieurs. Pour ce faire, nous avons décidé de faire la moyenne des lignes qui ont des coordonnées voisines. Nous commençons par ajouter la première ligne trouvée à une liste. On parcourt ensuite les lignes suivantes. Si la différence entre le point de début/fin de la ligne courante et le point de début/fin d'une des lignes contenues dans la liste est inférieure à un seuil arbitraire, on modifie l'entrée de la liste à la moyenne de ces deux lignes. Sinon, on ajoute la ligne courante à la liste Le résultat de la moyenne des lignes trouvées est présenté dans l'image suivante. Comme vous pouvez le voir, les seules lignes restantes suivent la grille et certaines des autres lignes dominantes caractéristiques de l'image.

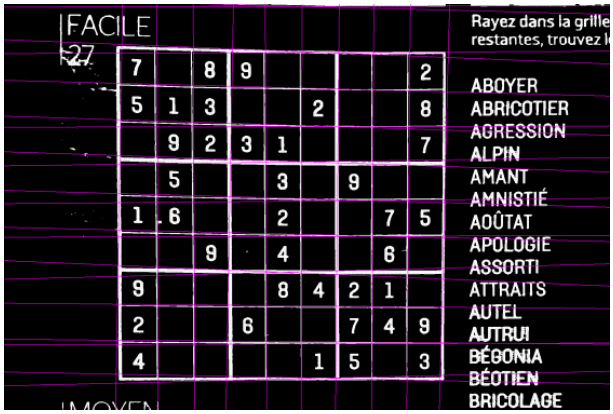

FIGURE 23 – *Réduction du nombre de bords détectés par moyenne*

La prochaine étape est la détection des carrés. Pour effectuer cette tâche, nous avons créé un algorithme simple. Nous commençons par parcourir toutes les lignes restantes. Pour la droite sélectionnée, on retrouve toutes les droites qui se croisent. Si la ligne sécante n'est pas la ligne sélectionnée, on continue. Si le nombre de droites sécantes trouvées avant de retrouver la première est 4, nous considérons que l'ensemble des droites trouvées est un carré Chaque carré trouvé est visible avec des bords verts sur l'image suivante. Nous devons ensuite sélectionner le bon carré qui suit le bord extérieur de la grille de sudoku. Pour ce faire, nous calculons un facteur pour chaque carré trouvé, qui est composé de : L'équerrage : la différence entre le bord le plus petit et le plus long du carré. On sélectionne alors le carré avec le meilleur facteur, et on considère qu'il s'agit de la délimitation de la grille. Ce carré est représenté par le carré rouge sur l'image suivante. Néanmoins, lors de cette étape nous rencontrons un problème majeur. Effectivement lors de la sélection des carrés, il semble que la première ligne, les deux dernières colonnes ainsi que la dernière ligne soit supprimé de notre rendu.

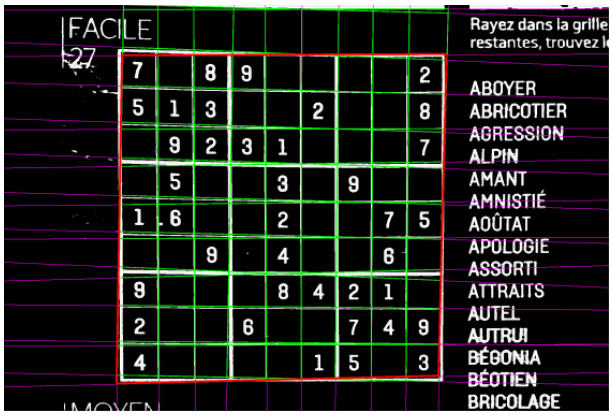

FIGURE 24 – *Détection et sélection du carré*

4.3 Recadrage du carré/Fractionnement de la grille

Pour que le recadrage de l'image suive le carré trouvé, on trouve le coin du carré le plus proche de l'origine, et on extrait les pixels situés aux coordonnées du coin jusqu'aux coordonnées du coin ajoutées à la longueur du plus long bord du carré. L'image suivante est le résultat du recadrage effectué suivant le carré trouvé.

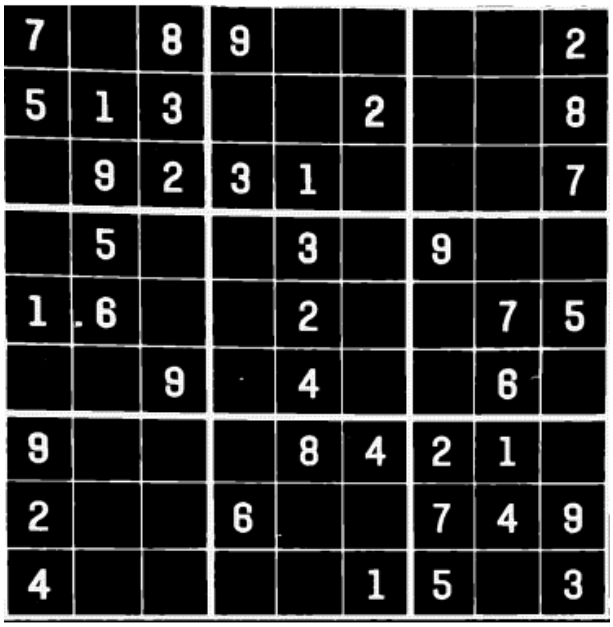

FIGURE 25 – *Recadrage du carré*

Enfin, nous divisons l'image en 9 parties sur l'axe x et 9 parties sur l'axe y pour extraire les cellules de la grille et les traiter plus facilement.

5 Solveur de sudoku

Ici, une méthode simple est utilisée. En effet toutes les possibilités sont générées et testées ainsi :

- Pour chaque cellule vide de la grille du Sudoku, un numéro entre 1 et 9 est attribué.
- Une fois toutes les cases vides remplies, le Sudoku est testé selon les règles du Sudoku : chaque case doit contenir un seul chiffre, seuls les chiffres de 1 à 9 peuvent être utilisés, chaque carré de 3 x 3 ne peut contenir que un nombre de 1 à 9, chaque colonne verticale ne peut contenir que chaque nombre de 1 à 9 et chaque ligne horizontale peut contenir un nombre de 1 à 9.
- Nous vérifions d'abord si tous les nombres ne sont présents qu'une seule fois dans chaque ligne. Puis pareil la vérification est effectuée dans chaque ligne. Finalement cette vérification est effectuée dans chaque carré 3 par 3 dans la grille du Sudoku.
- Si toutes ces vérifications sont correctes, le Sudoku résolu est écrit dans un fichier. Sinon, un message d'erreur s'affiche.

Comme spécifié, l'algorithme de résolution du Sudoku est une application qui s'exécute uniquement à partir de la ligne de commande. Le solveur prend en entrée un fichier dans lequel un la grille incomplète est enregistrée. L'algorithme résout la grille. Si une solution est trouvée et que le fichier contenant le résultat est généré. Le nom du fichier généré est le nom du fichier d'entrée avec l'ajout de l'extension ".result".

FIGURE 26 – *Application du solveur de sudoku*

6 Réseau de neurones

Réaliser le réseau neuronal pour le XOR était un travail très intéressant. D'abord la théorie, apprendre ce qu'est un réseau, de quoi est-il constitué a été très instructif, notamment la rétro-propagation qui est la partie la plus importante du travail.

x	0	0	1	1
y	0	1	0	1
z	0	1	1	0

FIGURE 27 – *la fonction XOR*

Le principal probléme était d'imiter la bibliothèque Numpy. Créer une structure appelée Matrix semblait être la meilleure réponse. Dans cette structure sont définies de nombreuses propriétés sur la matrice comme la multiplication, la transposition par exemple. Voici ci dessous la fonction entête ou toute les fonctions Matrix sont expliqués :

```c
// initilize a matrix
Matrix *m_init(Matrix *a, int rows, int cols);

// indentity matrix of size: size
Matrix *m_identity(Matrix *a, int size);

// free the matrix from the memory
void m_free(Matrix *a);

// set the value at a given index
Matrix *m_setIndex(Matrix *a, int r, int c, double val);

// map all value of a row according to the function given in parameter
Matrix *m_map_row(Matrix *a, int r, double (*f)(double));

// map all value of a column according to the function given in parameter
Matrix *m_map_col(Matrix *a, int c, double (*f)(double));

// map all value of the matrix according to the function given in parameter
Matrix *m_map(Matrix *a, double (*f)(double));

// add matrix b to a
Matrix *m_add(Matrix *a, Matrix *b);

// substract matrix b to a
Matrix *m_subtract(Matrix *a, Matrix *b);

// the matrix resulting is the matrix where indexes of same place are multiplied (it's just named hadamard product...)
Matrix *m_hadamard(Matrix *a, Matrix *b);

// multiply a matrix by a single double
Matrix *m_scalar_mult(Matrix *a, double x);

// add a double to each value of the matrix
Matrix *m_scalar_add(Matrix *a, double x);

// return a new matrix being the product of a and b
Matrix *m_mult(Matrix *a, Matrix *b, Matrix *dest);

// return a new matrix being the transpose
Matrix *m_transpose(Matrix *a, Matrix *dest);

// return a new matrix being the copy of the given in parameter
Matrix *m_copy(Matrix *src, Matrix *dest);
```

FIGURE 28 –

Nous devons maintenant mettre en place le réseau. Nous avons pour celà utilisé le site donnée dans le cahier des charges "http ://neuralnetworksanddeeplearning.com". Le réseau prend différents paramètres, d'abord le nombre d'entrées, le nombre de sorties, nombre de neurones cachés (il n'y aura toujours qu'une seule couche cachée), nombre d'époques (sorte de nombre de génération d'IA) et nombre de tests de formation. Définir les paramètres est une étape cruciale. Trop de neurones cachés diminueront la précision, pareil pour le nombre d'époques. Cela a nécessité de nombreux tests pour trouver les meilleures combinaisons de paramètres. Voici donc une photo du réseau :

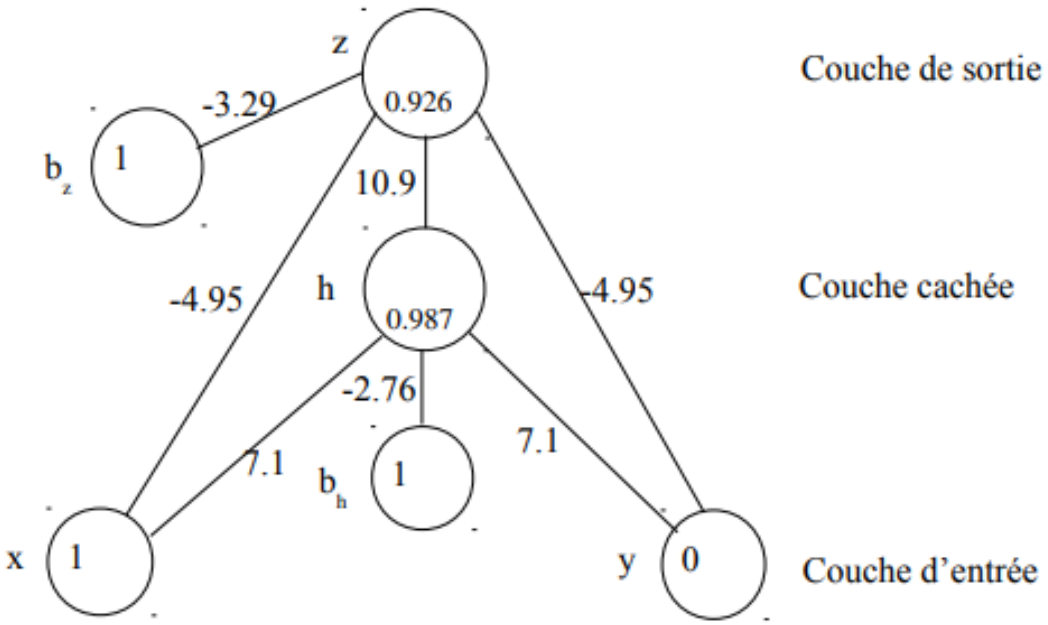

FIGURE 29 – *Un réseau à 3 couches pour résoudre le problème du XOR.*

7 Planning

Voici le planning d'avancement des tâches par période (temps séparant deux soutenances). Les deux prochaines périodes sont réparties ainsi :

— - Soutenance 1 : Semaine du 6 novembre 2023
— - Soutenance 2 : Semaine du 6 décembre 2023

Tâches	Soutenance 1	Soutenance 2
Chargement d'une image	100%	100%
Suppression des couleurs	100%	100%
Prétraitement	100%	100%
Détection de la grille	100%	100%
Détection des cases de la grille	50%	100%
Récupération des chiffres présents dans les cases	50%	100%
Reconnaissance de caractères	50%	100%
Reconstruction de la grille	0%	100%
Résolution de la grille	100%	100%
Affichage de la grille résolue	0%	100%
Sauvegarde de la grille résolue	0%	100%

8 Conclusion

Le développement de ce projet représente pour nous une grande première. Certains d'entre nous ont déjà un antécédent de programmation, mais cela semble dérisoire face à tous les besoins de ce projet. Beaucoup de temps et de travail a été et sera nécessaire, mais l'objectif est important. En effet, il est inévitable que nous sortirons tous beaucoup plus expérimentés que nous ne le sommes maintenant, ce qui est le but recherché. Nous espérons donc mettre à bien toutes les idées proposées dans ce document et que le résultat de ce projet soit de bonne qualité. Pour nous, avoir un résultat intéressant sera un accomplissement personnel car le développement sera une épreuve.

Enfin, l'esprit de groupe sera également un point important de l'apprentissage c'est à dire savoir s'entraider, communiquer, s'organiser et planifier nos objectifs. Ce sont des points que l'on attend d'un ingénieur et qui nous seront demandés durant nos carrières professionnelles. Il est donc évident que la qualité de ce projet sera un reflet de nos capacités à évoluer au sein d'un groupe pour aller vers un même but.

Quant à l'avancement du projet, nous sommes sur la bonne voie. Le traitement de l'image est la partie la plus compléte, nous avons pris de l'avance avec des fonctionnalités tels que la rotation automatique. Nous allons améliorer l'algorithme pour permettre le traitement d'images plus variées et plus dures. Nous avons aussi encore quelques problèmes au niveau de la sélection des cases de la grille. Ce qui pause problème pour la suite, cette partie la sera donc essentielle au bon déroulé du projet. Le solveur de Sudoku est terminé. Pour la seconde soutenance nous travaillerons sur l'affichage d'une image du résultat au lieu du simple affichage sur la console. Enfin, la structure de base d'un réseau neuronal est réalisée et fonctionne dans le Xor.