

# Rapport1

MGJV

September 2023

## Contents

<b>1</b>	<b>Le groupe</b>	<b>3</b>
1.1	Victor AGAHI . . . . .	3
1.2	Mehdi AZOUZ . . . . .	3
1.3	Jules MAGNAN . . . . .	3
1.4	Gaspard TORTERAT SLANDA . . . . .	4
<b>2</b>	<b>Ressources humaines</b>	<b>4</b>
<b>3</b>	<b>Prétraitement</b>	<b>5</b>
3.1	Filtre grayscale . . . . .	5
3.2	Flou de Gauss . . . . .	5
<b>4</b>	<b>Traitement</b>	<b>6</b>
4.1	Canny Edge . . . . .	6
4.1.1	Calcul des gradients . . . . .	6
4.1.2	Double Treshold . . . . .	6
4.1.3	Hysteris . . . . .	7
4.2	Redressement de l'image . . . . .	7
4.2.1	Rotation de l'image . . . . .	7
4.2.2	Redressement automatique . . . . .	7
4.3	Transformée de Hough . . . . .	8
4.3.1	Accumulateur . . . . .	8
4.3.2	Maximums locaux . . . . .	9
<b>5</b>	<b>Découpage de la grille</b>	<b>9</b>
<b>6</b>	<b>Epurer les cases</b>	<b>10</b>
6.1	Colonnes . . . . .	10
6.2	Reconnaissance du chiffre . . . . .	10
6.3	D'image à tableau . . . . .	10
<b>7</b>	<b>Résolution du sudoku</b>	<b>10</b>
7.1	Lecture de fichier . . . . .	10
7.2	Solvabilité . . . . .	10
7.3	Résolution . . . . .	10
7.4	Ecriture . . . . .	11
7.5	Technique . . . . .	11

<b>8</b>	<b>Réseau de neurones</b>	<b>12</b>
8.1	Propagation avant . . . . .	12
8.2	Rétropopagation . . . . .	12
8.3	XOR . . . . .	13
8.3.1	Entraînement . . . . .	13
8.3.2	Affichage . . . . .	14

# 1 Le groupe

## 1.1 Victor AGAHI

Bonjour, je suis Victor Agahi, étudiant à l'EPITA. Attiré par l'univers de l'informatique dès mes premières années, je considère que notre projet actuel constitue un défi stimulant. Il nous offre l'opportunité d'approfondir nos compétences techniques en langage C, tout en nous permettant d'explorer de nouvelles thématiques passionnantes, notamment dans le domaine de l'intelligence artificielle. Ce projet, axé sur un logiciel OCR, renforce ma conviction quant à l'importance de l'informatique dans notre société, et j'aborde cette initiative avec sérieux et je suis prêt à relever les défis techniques en équipe.



Figure 1: Victor Agahi

## 1.2 Mehdi AZOUZ

Bonjour, je m'appelle Mehdi, étudiant en deuxième année à EPITA. Passionné d'informatique depuis mon plus jeune âge, je trouve ce projet très intéressant car il nous permet d'apprendre de nouvelles thématiques telles que l'intelligence artificielle.

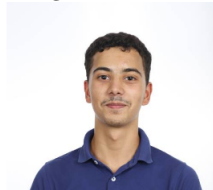


Figure 2: Mehdi AZOUZ

## 1.3 Jules MAGNAN

Bonjour, étudiant à l'Epita je me passionne pour les défis informatiques. Ce projet dans lequel nous nous lançons est très formateur il nous pousse à mener des défis techniques en C et nous forme à créer des projets en équipe.



Figure 3: Jules MAGNAN

## 1.4 Gaspard TORTERAT SLANDA

Bonjour, je suis Gaspard, un étudiant d'EPITA. Je suis convaincu que l'informatique est un domaine important de notre société qui regorge de défi. C'est avec sérieux et minutie que je compte réaliser ce projet centré sur un logiciel OCR.



Figure 4: Gaspard TORTERAT SLANDA

## 2 Ressources humaines

Ci-dessous la répartition des tâches pour la première période de travail.

Membre \ Tâche	Gaspard	Jules	Mehdi	Victor
Traitement de l'image				
Prétraitement	Principal			
Rotation	Principal			
Détection de la grille	Principal			
Découpage de la grille		Principal		
Traitement des cases		Principal		
Transforme de Hough				Principal
OCR				
Réseau de neurones		Second	Principal	
Banque d'images			Principal	
Autre				
Site web		Principal		
Sudoku		Principal		

Table 1: Répartition des tâches

### 3 Prétraitement

Le prétraitement est l'étape qui doit nettoyer les impuretés de l'image pour que le réseau de neurones puisse performer au mieux.

$$\int_1^2 \sum_{\alpha}^8 1$$

#### 3.1 Filtre grayscale

Une image se découpe en pixels. Un pixel se décompose en trois composantes : le rouge, le vert et le bleu. Le but du filtre grayscale est de diminuer la quantité d'information. Nous appliquons la formule suivante sur chaque pixel de l'image :

$$\forall (x, y) \in \llbracket 0, height \rrbracket \times \llbracket 0, width \rrbracket$$

$$IMAGE(x, y) = 0.3 * R + 0.59 * V + 0.11 * B$$

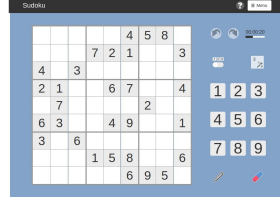


Figure 5: Avant filtre grayscale

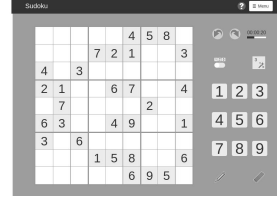


Figure 6: Après filtre grayscale

#### 3.2 Flou de Gauss

Une image peut être bruitée. Le bruit se traduit par l'apparition de pixels noirs ou de fort contrastes dans certaines zones de l'image. Pour pallier à ce premier problème, bien que peu courant en pratique, le filtre médian est idéal.

Pour le second, nous utilisons un flou de Gauss avec un kernel de 5x5. Cette étape uniformise la distribution des valeurs des pixels. Elle vise à améliorer les chances de détection des contours de la grille. Ainsi nous appliquons la convolution suivante à chaque pixel de l'image :

$$\forall (x, y) \in \llbracket 0, height \rrbracket \times \llbracket 0, width \rrbracket$$

$$\text{Soit } P(x, y) = IMAGE(x, y)$$

$$\text{Soit } M(x, y) = \begin{pmatrix} P(x-2, y-2) & P(x-2, y-1) & P(x-2, y) & P(x-2, y+1) & P(x-2, y+2) \\ P(x-1, y-2) & P(x-1, y-1) & P(x-1, y) & P(x-1, y+1) & P(x-1, y+2) \\ P(x, y-2) & P(x, y-1) & P(x, y) & P(x, y+1) & P(x, y+2) \\ P(x+1, y-2) & P(x+1, y-1) & P(x+1, y) & P(x+1, y+1) & P(x+1, y+2) \\ P(x+2, y-2) & P(x+2, y-1) & P(x+2, y) & P(x+2, y+1) & P(x+2, y+2) \end{pmatrix}$$

$$\text{Alors } IMAGE(x, y) = \frac{1}{273} \times \begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix} * M(x, y)$$



$$2. T1 \leq G(x, y) < T2$$

$$3. T2 \leq G(x, y)$$

Les pixels dont la magnitude est inférieure au premier seuil sont supprimés de l'image. Les pixels dont la magnitude est comprise entre le premier seuil et le deuxième seuil sont marqués en tant que pixels faibles, tandis que les autres pixels sont marqués pixels forts.

#### 4.1.3 Hysteris

Enfin, l'algorithme vérifie que pixel faible est voisin immédiat d'un pixel fort. Si la recherche est négative, le pixel est supprimé de l'image. Sinon, le pixel devient un pixel fort.

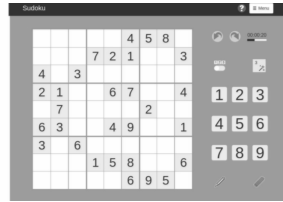


Figure 9: Avant Canny Edge

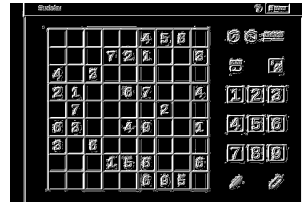


Figure 10: Après Canny Edge

## 4.2 Redressement de l'image

Conformément au cahier des charges, nous implémentons la rotation de l'image de sudoku. Nous tournons l'image en appliquant une matrice de rotation à deux dimensions.

### 4.2.1 Rotation de l'image

Nous appliquons les trois matrices de rotation suivantes aux coordonnées de chaque pixel : Soit  $\theta$  l'angle de rotation et  $IMG(x, y)$  un pixel de coordonnées  $(x, y)$ .

$$\text{Soient } P = \begin{pmatrix} 0 & -\tan(\frac{\theta}{2}) \\ 1 & 0 \end{pmatrix} \quad Q = \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \quad \text{et } R = \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix}$$

Alors les nouvelles coordonnées  $(x', y')$  du pixel sont données par:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \times P \times Q \times R$$

### 4.2.2 Redressement automatique

Nous avons implémenté la rotation automatique de l'image. Nous utilisons dans un premier temps la transformée de Hough, puis nous détectons le plus mince angle formé par une droite de l'image et l'axe horizontal.

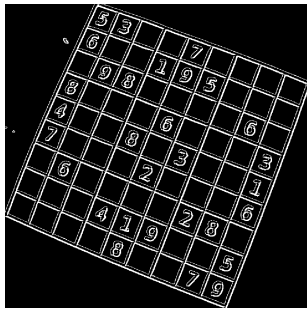


Figure 11: Avant rotation

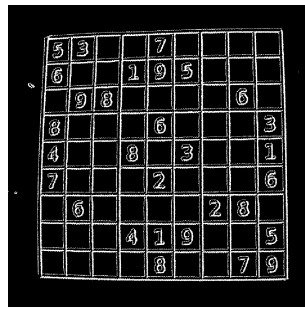


Figure 12: Après rotation

### 4.3 Transformée de Hough

Pour pouvoir détecter des formes dans une image, nous utilisons un algorithme répandu : la transformée de Hough. Nous centrons cette méthode sur la détection de lignes dans l'image. Elle utilise un accumulateur dans l'espace polaire qui entrepose les droites. Cette méthode s'utilise sur une image traitée au préalable par Canny Edge ou autre un algorithme dont le but est semblable.

#### 4.3.1 Accumulateur

Pour chaque pixel  $M(x,y)$ , il faut calculer la distance  $\rho$  à l'origine, donnée en fonction de l'angle  $\theta$  par :

$$\forall \theta \in [0, 180] \quad \rho = x * \cos(\theta) + y * \sin(\theta)$$

Ensuite il faut incrémenter la position  $[\rho] [\theta]$  de l'accumulateur.

Puis, il est possible d'enregistrer l'accumulateur sous la forme d'une image ci-dessous, en veillant à normaliser les valeurs pour qu'elles se situent entre 0 et 255. Pour visualiser les contours identifiés, nous parcourons notre accumulateur et sélectionnons chaque point de coordonnées  $(\rho, \theta)$  situé au-dessus d'un seuil préalablement défini. Nous récupérons ensuite les coordonnées polaires et en déduisons les coordonnées cartésiennes  $(x_0, y_0)$  à l'aide des formules suivantes :

$$x_0 = \rho \cdot \cos(\theta) \quad \text{et} \quad y_0 = \rho \cdot \sin(\theta)$$



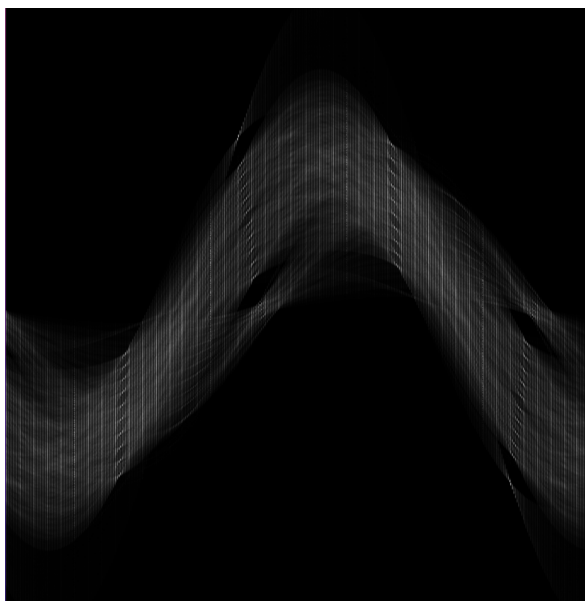


Figure 13: Accumulateur normalisé

#### 4.3.2 Maximums locaux

Dans cette étape, nous récupérons les pics de l'accumulateur et supprimons leurs valeurs voisines. Nous sélectionnons ces pics s'ils sont supérieurs à un seuil défini au préalable.

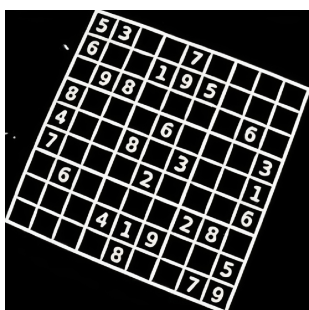


Figure 14: Avant Hough

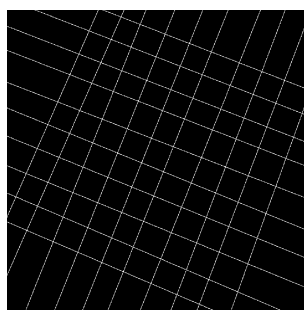


Figure 15: Après Hough

## 5 Découpage de la grille

Pour que les caractères présents sur la grille puissent être lus il faut dans un premier temps découper l'image. Pour cela nous utilisons un algorithme qui découpe l'image en 81 cases de 28 pixels par 28 pixels. Ces cases seront donc enregistrés dans un autre dossier pour que nous puissions les épurer.

## 6 Epurer les cases

### 6.1 Colonnes

L'épuration des cases est nécessaire pour qu'elles puissent par la suite être lues par notre réseau de neurones. Pour enlever les lignes l'algorithme passe d'abord faire le contour des cases automatiquement de façon à ce que le plus gros morceau restant soit le chiffre.

### 6.2 Reconnaissance du chiffre

Après l'effacement des colonnes, un algorithme va se déplacer sur l'image de façon à trouver quel est l'amat de pixel le plus gros qui sera alors forcément le chiffre puisque nous avons enlevé les bordures précédemment. Lorsque ceci est fait nous pouvons redessiner les pixels du chiffre sur une image que nous allons enregistrer pour qu'elles puissent être analysées par le réseau de neurone dans la suite du processus.

### 6.3 D'image à tableau

Maintenant que nous sommes en moyen d'obtenir chaque cases indépendamment il faut que nous soyons capables de les retranscrire en tableaux de chiffres. Pour cela nous récupérons la matrice de pixels de l'image grâce à SDL puis nous regardons sur chaque pixels s'il est noir ou blanc. Puis nous mettons dans une matrice la valeur du pixel pour qu'il puisse être lu par le réseau de neurones.

## 7 Résolution du sudoku

### 7.1 Lecture de fichier

Pour que la résolution du sudoku soit conforme nous avons implémenté un algorithme de lecture de fichier. Le fichier du sudoku initial comprend donc les caractères déjà présents sur le sudoku et représente les cases vides par des points. Lors de la lecture du fichier les caractères sont donc ajoutés dans une matrice et les points sont remplacés par des 0 pour la résolution. Pour simplifier l'utilisation de notre tableau il sera alloué à l'aide de calloc.

### 7.2 Solvabilité

Avant de commencer à résoudre le sudoku pour des questions de performance et d'utilisation il est indispensable de vérifier que le sudoku est soluble. Lorsque l'exécution de résolution est donc appelé nous avons d'abord la vérification de chaque ligne, colonne et carré de la grille pour vérifier qu'il est possible de résoudre le sudoku.

### 7.3 Résolution

Ensuite l'algorithme de résolution est assez intuitif. A la première occurrence d'un 0 dans le tableau nous allons le remplacer par un 1 puis vérifier si sa présence commet une erreur dans le sudoku (donc nous vérifions si le sudoku

reste correct). Si le remplacement n'a pas posé de problème alors nous continuons sur les prochaines cases, sinon nous remplaçons le 1 par un 2 et ainsi de suite jusqu'à trouver une combinaison qui fonctionne. Evidemment si arrivé à 9 la résolution ne fonctionne pas l'algorithme se relance du début en changeant la configuration de départ jusqu'à trouver une combinaison fonctionnelle.

## 7.4 Ecriture

Lorsque le sudoku est résolu nous n'avons donc plus qu'à réécrire la matrice dans un fichier sous le meme format que l'initial avec pour extension ".result". Ce fichier contient donc le résultat du sudoku.

## 7.5 Technique

Pour que la résolution du sudoku soit conforme le dossier est munit d'un Makefile permettant de créer un exécutable. Ensuite à l'aide de cet exécutable il suffit de passer en paramètre le fichier comprenant le sudoku et dans le dossier de la grille sera créé le résultat avec l'extension ".result".

## 8 Réseau de neurones

Un réseau de neurones est un modèle informatique organisé en plusieurs parties. La première partie, appelée **propagation avant (ou front propagation)**, consiste à faire prédire au réseau de neurones une valeur de sortie en utilisant les données d'entrée. Cette prédiction est basée sur les connexions pondérées entre les neurones et les fonctions d'activation appliquées à ces connexions.

Ensuite, il y a la **rétropropagation (ou backpropagation)**, qui intervient après la prédiction. Son rôle est de corriger d'éventuelles erreurs faites par le réseau. Pour ce faire, elle utilise un algorithme qui ajuste les poids des connexions entre les neurones en fonction de la différence entre la prédiction du réseau et la valeur attendue. Cette rétropropagation permet au réseau de s'améliorer progressivement au fil de l'apprentissage, en minimisant les erreurs entre les prédictions et les valeurs réelles.

### 8.1 Propagation avant

Nous avons décidé d'utiliser la fonction sigmoïde pour notre fonction d'activation elle permet de faire prédire à notre réseau de neurones une valeur de sortie.

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

### 8.2 Rétropropagation

Nous avons décidé d'utiliser la fonction dérivée de sigmoïde afin de corriger les éventuelles erreurs du réseau de neurones. Cette fonction d'apprentissage permet au réseau de neurones de comprendre ses erreurs.

Fonction dérivée de sigmoïde:  $f'(x) = x \cdot (1 - x)$

### 8.3 XOR

Notre réseau de neurones est organisé selon la structure suivante : il comprend 2 entrées, chacune pouvant prendre des valeurs comprises entre 0 et 1. Le réseau est également doté d'une couche cachée composée de 3 neurones. En outre, notre réseau de neurones possède une sortie qui produit une valeur comprise entre 0 et 1, visant à se rapprocher le plus possible de 0 ou de 1.

Voici ci-dessous un schéma représentant notre réseau de neurones:

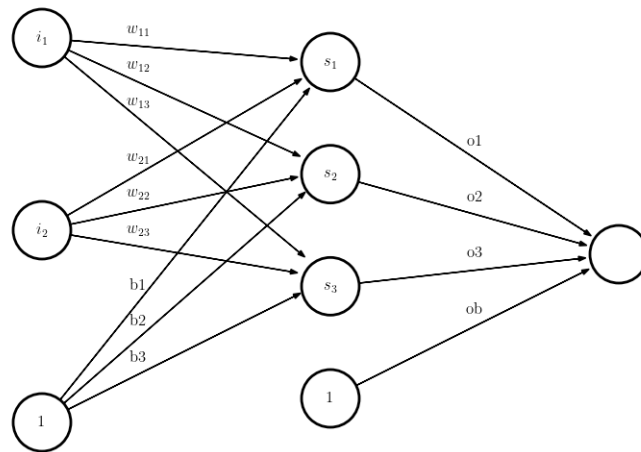


Figure 16: Schéma réseau de neurones XOR

#### 8.3.1 Entraînement

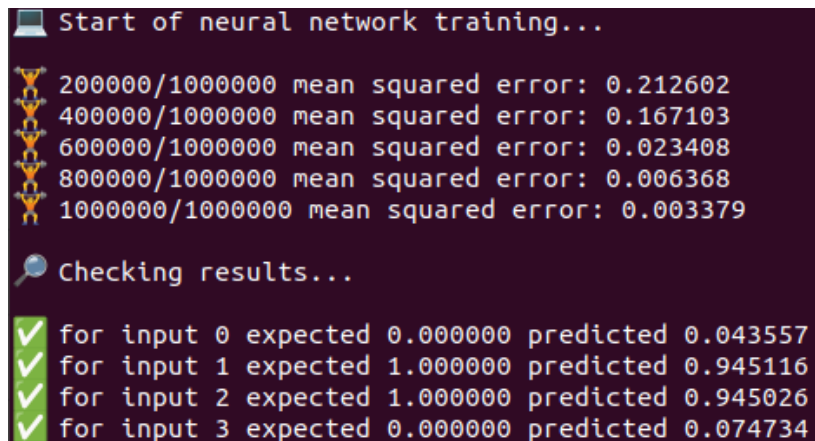
Pour entraîner notre réseau de neurones, nous testons à de nombreuses reprises les 4 combinaisons possibles des entrées XOR. Cela permet au réseau de neurones d'ajuster les biais et les poids de manière à minimiser les erreurs au fil des tests. Nous avons décidé de réaliser 1 000 000 de tests pour réduire au maximum la marge d'erreur lors des évaluations.

### 8.3.2 Affichage

Nous avons choisi d'implémenter un affichage sur le terminal lors de l'exécution du programme afin de faciliter la compréhension du fonctionnement de chaque composant du réseau de neurones. Cela nous permet de visualiser la progression du réseau au fil de son apprentissage.

Lorsque vous lancez l'exécutable, les informations affichées dans le terminal vous donnent un aperçu détaillé des opérations effectuées par chaque partie du réseau. Cette fonctionnalité offre une meilleure transparence sur les calculs effectués par les neurones, la manière dont les poids et les biais sont ajustés, ainsi que la manière dont le réseau se rapproche des valeurs attendues.

Ce suivi en temps réel dans le terminal offre une opportunité précieuse pour analyser le comportement du réseau de neurones et ajuster les paramètres si nécessaire, permettant ainsi d'optimiser les performances du modèle. Cette visibilité accrue contribue à une meilleure compréhension du processus d'apprentissage et facilite le débogage en cas d'erreurs ou d'incohérences dans les résultats obtenus.



```
Start of neural network training...  
200000/1000000 mean squared error: 0.212602  
400000/1000000 mean squared error: 0.167103  
600000/1000000 mean squared error: 0.023408  
800000/1000000 mean squared error: 0.006368  
1000000/1000000 mean squared error: 0.003379  
Checking results...  
✓ for input 0 expected 0.000000 predicted 0.043557  
✓ for input 1 expected 1.000000 predicted 0.945116  
✓ for input 2 expected 1.000000 predicted 0.945026  
✓ for input 3 expected 0.000000 predicted 0.074734
```

Figure 17: Affichage du terminal XOR

## List of Tables

1	Répartition des taches . . . . .	4
---	----------------------------------	---

## List of Figures

1	Victor Agahi . . . . .	3
2	Mehdi AZOUZ . . . . .	3
3	Jules MAGNAN . . . . .	4
4	Gaspard TORTERAT SLANDA . . . . .	4
5	Avant filtre grayscale . . . . .	5
6	Après filtre grayscale . . . . .	5
7	Avant Flou de Gaus . . . . .	6
8	Après flou de Gauss . . . . .	6
9	Avant Canny Edge . . . . .	7
10	Après Canny Edge . . . . .	7
11	Avant rotation . . . . .	8
12	Après rotation . . . . .	8
13	Accumulateur normalisé . . . . .	9
14	Avant Hough . . . . .	9
15	Après Hough . . . . .	9
16	Schéma réseau de neuronesXOR . . . . .	13
17	Affichage du terminal XOR . . . . .	14

## References

- [1] John Canny (1986) *A Computational Approach To Edge Detection*, IEEE.