

# Rapport de soutenance Optical Character Recognition Sudoku

Louis Le Quellec, Antoine Thomas, Delphine Morvan  
et Méline-Aya Soumaire

10 Novembre 2022



# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Le groupe . . . . .	3
1.2	Le projet . . . . .	3
1.3	La répartition des charges . . . . .	3
<b>2</b>	<b>Aspects techniques</b>	<b>4</b>
2.1	Traitement d'image . . . . .	4
2.1.1	Niveaux de gris . . . . .	4
2.1.2	Flou gaussien . . . . .	6
2.1.3	Détection des coins avec l'algorithme de canny . . . . .	7
2.1.4	La transformation de Hough . . . . .	8
2.2	Découpage de l'image . . . . .	9
2.3	Réseau de neurones . . . . .	10
2.4	Solver . . . . .	11
<b>3</b>	<b>Avancement du projet</b>	<b>12</b>
3.1	Première soutenance . . . . .	12
3.2	Dernière soutenance . . . . .	12
<b>4</b>	<b>Conclusion</b>	<b>12</b>
<b>5</b>	<b>Bibliographie</b>	<b>13</b>

# 1 Introduction

## 1.1 Le groupe

Notre groupe est composé de 4 étudiants de deuxième année à l'EPITA Rennes : Louis Le Quellec, Antoine Thomas, Delphine Morvan and Méline-Aya Soumaire en classe R1.

## 1.2 Le projet

Le projet consiste en l'implémentation d'un OCR pendant le troisième semestre par groupe de 4. Il doit prendre en paramètre l'image d'un sudoku, traiter l'image afin de la rendre plus simple à interpréter. Puis il doit la découper en cases qui passent ensuite par un réseau de neurones permettant de reconnaître les chiffres de chacune des cases. Un programme résout alors le sudoku et celui-ci est renvoyé résolu.

## 1.3 La répartition des charges

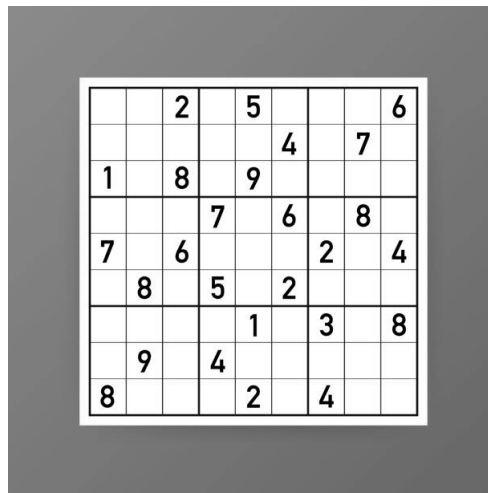
	Louis	Antoine	Delphine	Méline-Aya
Chargement d'une image et suppression des couleurs		x		
Suppression des bruits		x		
Rotation manuelle de l'image		x	x	
Détection de la grille et de la position des cases			x	
Découpage de l'image				x
La résolution de la grille				x
Mini réseau de neurones XOR	x			

## 2 Aspects techniques

### 2.1 Traitement d'image

#### 2.1.1 Niveaux de gris

Pour commencer le traitement d'image, il a fallu rendre l'image plus simple, avec moins de pixel parasite. Le but est donc pour commencer de supprimer toutes les couleurs que nous pouvions avoir dans l'image, pour laisser uniquement du blanc, du gris plus ou moins foncé et du noir. Nous commençons par parcourir tous les pixels et déterminons ses valeurs RGB, c'est-à-dire que chaque pixel est encodé sous 3 octets, qui pour chacun d'entre eux possède une valeur entre 0 et 255. Ce sont donc ces 3 octets qui donnent la couleur à nos pixels. Pour avoir une couleur qui se trouve entre blanc et noir, il faut que les 3 octets possèdent exactement la même valeur, qui se trouve être la moyenne des trois valeurs initiales. Nous obtenons donc, une image en noir et blanc.

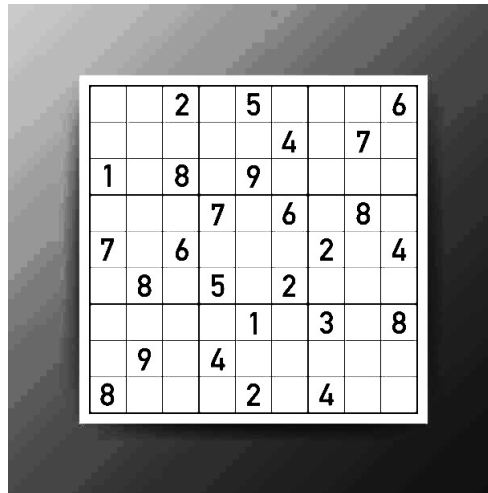


		2		5				6	
					4		7		
1		8		9					
			7		6		8		
7		6				2		4	
	8		5		2				
				1		3		8	
	9		4						
8				2		4			

Rendu de l'image après l'application du niveau de gris

Dans un deuxième temps, nous allons augmenter le contraste de l'image, en partant de la fonction "grayscale" (la fonction pour faire du noir et blanc). Le but du contraste, lorsqu'on l'augmente, est de d'augmenter la luminosité de chaque pixel qui sont clairs, donc plus proche du blanc que du noir et inversement. En d'autres termes, il faudra diminuer la valeur du pixel s'il est entre 0 et 127 et l'augmenter lorsque la valeur est entre 128 et 255. Nous avons donc cette fonction là, qu'il faudra appliquer à chaque pixel :

$$f(x) = \begin{cases} 127 * \left(\frac{2x}{255}\right)^n & \text{si } x \in [0 ; 127] \\ 255 - f(255 - x) & \text{si } x \in [128 ; 255] \end{cases} \quad (1)$$



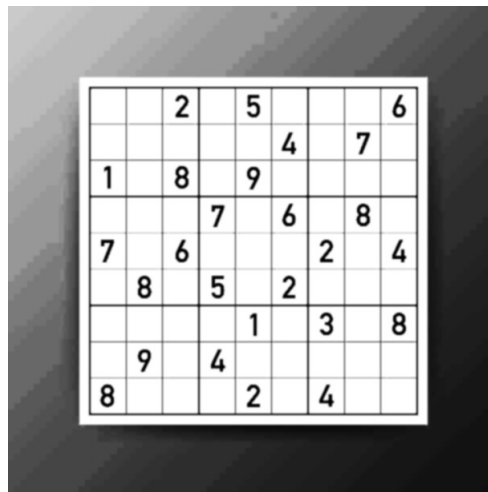
Rendu de l'image après l'application du niveau de gris avec une saturation plus puissante

### 2.1.2 Flou gaussien

Pour effectuer le flou Gaussien, on utilise une matrice de convolution représentée par la matrice suivante. Cette matrice est une matrice carrée de taille 5.

$$\frac{1}{256} \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

Cette matrice permet pour chaque pixel de notre image d'appliquer une convolution afin d'obtenir une 'moyenne' des pixels autour de notre pixel en utilisant la matrice : on prend donc les valeurs RGB de chaque pixels autour de notre pixel actuel (en vérifiant au préalable que les coordonnées sont bien dans notre liste) puis on remplace la valeur de notre pixel actuel par cette moyenne



Rendu de l'image après l'application du flou gaussien.

### 2.1.3 Détection des coins avec l'algorithme de canny

Pour détecter les contours, nous allons utiliser l'algorithme de canny qui utilise un filtrage linéaire avec un noyau gaussien pour lisser le bruit, puis calcule la force et la direction des contours pour chaque pixel de l'image lissée.

Pour implementer cet algorithme, il faut suivre plusieurs etapes :

- Filtre de Sobel :

le filtre de Sobel a besoin de deux matrices de Convolution de  $3 * 3$  , les deux matrices sont les suivantes :

$$Gx = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$
$$Gy = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

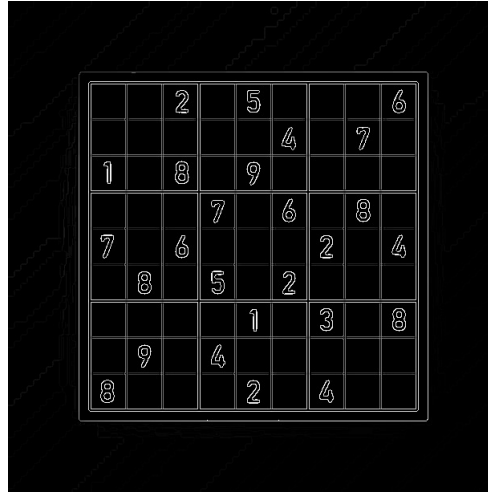
L'algorithme se deroule de la maniere suivante :

Pour chaque pixel de notre image, on fait une convolution avec Gx et Gy puis on recupere ses points pour faire une norme et enfin appliquer cette norme a  $\arctan2$  pour recuperer la direction du gradient pour pouvoir enfin appliquer la meme valeur obtenue dans les gradients pour chaque valeur RGB.

- NonMaxSuppression :

On prend la liste obtenue par Sobel puis avec l'angle obtenu, on parcourt tous les pixels et pour chaque pixel on cherche l'angle en degre et on change la valeur dans la liste. Et enfin pour chaque pixels on normalise la couleur (r = g = b)

Les deux derniers parties ne sont pas encore implements, il manque donc a Canny : Le double thersold ainsi que l'Hyperthesis.



Rendu de l'image après l'application de canny.

#### 2.1.4 La transformation de Hough

La transformation de Hough consiste à trouver les lignes de notre sudoku afin de pouvoir détecter les cases. Tout d'abord, il faudra parcourir tous les pixels de l'image sur laquelle nous travaillons. Notre but est donc de créer une image qui serait, dans un repère polaire, un graphique qui permettrait de pouvoir trouver les équations des différentes droites de notre image initiale. Pour faire cela, il va falloir tout d'abord récupérer la largeur et la hauteur, en nombre de pixel, de notre image. Grâce à cela, nous créerons une matrice de largeur 360 pixels et de hauteur :

$$\sqrt{\text{largeur}^2 + \text{hauteur}^2} \quad (2)$$

Une fois que cette matrice est créée, il faudra, pour chaque pixel non noir (ce qui signifie qu'il s'agit d'un pixel important), calculer :

$$r = x * \cos(\theta) + y * \sin(\theta) \quad (3)$$

Nous ferons varier theta entre 0 et 360. Nous ajouterons alors 1 à la position (theta,r) dans notre matrice. Par la suite, nous allons donc créer une image de la même taille que de notre matrice. Nous allons donc trouver le maximum de notre matrice afin de pouvoir faire que le maximum soit plus blanc que les autres points. Nous obtiendrons alors un graphique avec différentes courbes qui se rejoignent en un point, justement le maximum, qui est donc l'équation en coordonnée polaire.



## 2.2 Découpage de l'image

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

Image du sudoku en paramètre

Grâce aux étapes précédentes, nous aurons les coordonnées de chacune des cases et ainsi nous pourrions découper l'image du sudoku. Nous créons dans un premier temps une surface à partir de l'image du sudoku. Puis, nous créons une nouvelle surface composée de pixel noir de la taille de la case. La taille de la case est déterminée à partir de la position du pixel en haut à gauche et de celle du pixel en bas à droite. Sur la nouvelle surface, chaque pixel est remplacé par les pixels de la case placée sur la grille. Cela ressemble au fait de prendre une capture d'écran d'un bout de l'image du sudoku. Les cases récupérées seront ensuite triées en fonction de celles qui contiennent un chiffre et celles qui n'en contiennent pas. Les premières seront alors transformées en zéro et un, pour pouvoir être utilisées dans le réseau de neurones.



Rendu pour la case numéro 22

## 2.3 Réseau de neurones

Le réseau de neurones est le programme qui pourra permettre de différencier les différents chiffres du sudoku, il prend une suite de 0 et de 1 qui seront les pixels de l'image avec 0 pour blanc et 1 pour noir et calcule en passant par plusieurs opérations quel chiffre correspond a l'entrée.

Pour qu'il soit efficace, ce programme a besoin de s'entraîner avec une base de données dont il connaît les réponses pour en fonction de l'erreur corriger les calculs dans le but de trouver plus aisément ce même résultat si plus tard une image similaire lui est soumise.

Pour cette soutenance l'objectif était d'implémenter ce principe a l'opération xor (ou exclusif) l'entrée est donc 2 chiffres étant chacun soit un 0 soit un 1 et la sortie est une valeur a virgule dont l'objectif sera de faire le plus converger vers le bon résultat et ce pour chacune des 4 combinaisons possibles.

L'entraînement du réseau de neurones peut malheureusement rencontrer des difficultés comme des "paliers" pour lesquels les valeurs peuvent stagner rendant impossible la suite de l'entraînement alors que les résultats ne sont pas encore bons. Ce problème aura été celui le plus compliqué a aborder étant donné que les valeurs de départ du réseau sont aléatoires il y a toujours des chances pour qu'il se rapproche de ces paliers en fonction de ses valeurs.

## 2.4 Solver

Le solver est un programme qui doit résoudre un sudoku. Dans notre cas, il prend en paramètre un fichier qui contient les chiffres déjà présent et des points à la place des case vides.

```
... ..4 58.  
... 721 ..3  
4.3 ... ..  
  
21. .67 ..4  
.7. ... 2..  
63. .49 ..1  
  
3.6 ... ..  
... 158 ..6  
... ..6 95.
```

Contenu du fichier en paramètre

Le programme résout la grille à l'aide de ce qu'on appelle le "backtracking", le retour en arrière. Il s'agit d'un programme qui teste toutes les possibilités pour chacune des cases tant que la grille n'est pas résolue. Par exemple, pour chaque case vide, il testera un chiffre en regardant s'il n'est pas déjà présent dans la même colonne, ligne ou dans le même carré. Si ce n'est pas le cas, le chiffre est "validé". Le programme testera alors la case vide suivante. Si le programme arrive à 9, c'est qu'il n'y avait pas de solution. Alors, il fait un retour en arrière et retourne à la case précédente anciennement vide et teste le chiffre suivant.

Une fois l'algorithme complété, le programme renvoie un fichier avec le sudoku résolu.

```
127 634 589  
589 721 643  
463 985 127  
  
218 567 394  
974 813 265  
635 249 871  
  
356 492 718  
792 158 436  
841 376 952
```

Contenu du fichier en sortie

## **3 Avancement du projet**

### **3.1 Première soutenance**

Pour la première soutenance, nous avons réalisé la majorité des tâches obligatoires, c'est-à-dire le chargement d'une image et la suppression des couleurs, la suppression des bruits, la rotation manuelle de l'image, le découpage de l'image, la résolution de la grille et le mini réseau de neurones XOR.

### **3.2 Dernière soutenance**

Nous finirons le projet d'ici la dernière soutenance. Nous nous sommes déjà réparti les tâches restantes et comptons tout faire pour réussir ce challenge. Si le temps nous le permet, nous serons ravis d'ajouter d'autres fonctionnalités au projet ou de créer un site web.

## **4 Conclusion**

Même si nous n'avons pas réussi à finir tous ce que nous devons faire pour la première soutenance, nous avons fait de notre mieux. De plus, notre groupe fonctionne très bien et nous espérons que cette cohésion nous permettra d'aller au maximum de nos capacités pour finir ce projet et rattrapper notre retard.

## 5 Bibliographie

Traitement de l'image :

<https://zestedesavoir.com/tutoriels/pdf/1014/utiliser-la-sdl-en-langage-c.pdf>

[https://www.researchgate.net/publication/282075920\\_Implementing\\_Hough\\_transformation\\_with\\_C\\_language\\_of\\_programming](https://www.researchgate.net/publication/282075920_Implementing_Hough_transformation_with_C_language_of_programming)

[https://rosettacode.org/wiki/Example:Hough\\_transform/C](https://rosettacode.org/wiki/Example:Hough_transform/C)

[https://en.wikipedia.org/wiki/Hough\\_transform](https://en.wikipedia.org/wiki/Hough_transform)

Réseau de neurones :

<https://www.youtube.com/watch?v=hfMk-kjRv4c&t=2022s>

Solver :

<https://www.geeksforgeeks.org/sudoku-backtracking-7/>