
Rapport de soutenance

Reconnaissance Optique des Caractères (ROC)

Optical Character Recognition (OCR)

Alexandra Wolter - Erwann Lesech - Guillaume Jolivalt - Valentin Gibbe

EPITA Lyon | Promo 2026



CONTENTS

1	Introduction	2
1.1	Qu'est ce qu'un OCR ?	2
1.2	Présentation du projet	2
1.3	Présentation du groupe de travail	3
2	La répartition des tâches	4
2.1	Tableau de répartition	4
3	Etat d'avancement du projet	5
3.1	Prévisions cahier des charges	5
3.1.1	Rappel des estimations dans l'avancement des tâches	5
3.1.2	Avancement final	6
4	Aspects techniques	7
4.1	Traitemet d'image	7
4.1.1	Chargement d'une image et suppression des couleurs	7
4.1.2	Rotation manuelle et automatique de l'image	7
4.1.3	Binarisation avec réduction de bruit	9
4.1.4	Détection de la grille et de la position des cases	10
4.2	Découpage de la grille	14
4.2.1	Découpage des cases	14
4.2.2	Nettoyage et sauvegarde des cases	14
4.3	Résolution du sudoku	15
4.4	Réseau de neurones	16
4.4.1	Preuve de concept (porte ou-exclusive)	18
4.4.2	Réseau de neurones principal	20
4.5	Interface graphique	24
4.6	Reconstruction de la grille	27
4.7	Site internet	28
5	Limites et améliorations possibles	30
6	Ressentis du projet	32
7	Conclusion	34
Acronyms		36

CHAPTER 1

INTRODUCTION

1.1 Qu'est ce qu'un OCR ?

Un Optical Character Recognition (OCR) ou Reconnaissance Optique des Caractères (ROC) correspond à la reconnaissance automatisée de textes imprimés ainsi qu'à la retranscription de ces textes écrits en fichiers numériques.

De fait, un OCR permet à un appareil numérique de "lire" le contenu d'un document papier et de le restituer informatiquement.

1.2 Présentation du projet

Dans notre cas, l'objectif de notre OCR est de scanner un sudoku imprimé, de numériser la grille ainsi que les chiffres déjà inscrits. Par la suite, notre logiciel devra être capable de résoudre ce sudoku pré-rempli pour enfin le restituer dans sa forme complétée via une interface utilisateur ergonomique et intuitive.

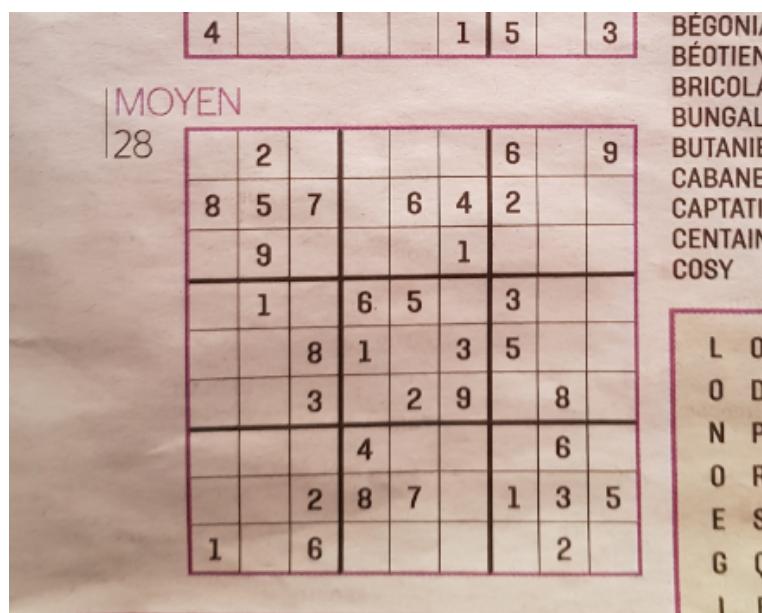


Figure 1.1: Sudoku imprimé

1.3 Présentation du groupe de travail

Erwann "R-One" LESECH (Chef de projet)

Roux et motard du groupe, j'ai été en charge principalement de la création du réseau de neurones dans le but de reconnaître les caractères.

Ce projet m'aura permis d'implémenter mon premier réseau de neurones et d'en apprendre plus sur les domaines du machine learning et du deep learning. De plus,似ilairement au projet de S2, nous avons été amenés à travailler en groupe sur un cahier des charges très précis ce qui m'a demandé en tant que chef de groupe, une organisation certaine.

Guillaume "Guimauve" JOLIVALT

Curieux du groupe, je me suis attelé à des tâches dans différents domaines telles que la résolution du sudoku, la réduction de bruits ou encore la recherche d'information sur le réseau de neurones. J'ai beaucoup apprécié travailler sur ces diverses parties avec ce groupe.

Ce projet m'a donc permis de découvrir et d'explorer de nouveaux univers plus captivants les uns que les autres.

Alexandra "Alex" WOLTER

Joueuse de Wow et seule fille du groupe, j'ai réalisé l'intégralité de l'interface graphique du projet. C'était une grande première pour moi et j'ai beaucoup aimé réaliser cette tâche. J'ai également implémenté la reconstitution de la grille mais aussi d'autres tâches qui m'ont beaucoup appris.

Ce projet était un nouveau challenge à relever pour moi et je suis contente du projet rendu par mes camarades et moi même !

Valentin "Nimu" GIBBE

Alcoolique du groupe, je suis globalement motivé pour tous types de projets informatique. Le projet OCR m'a permis de m'améliorer en informatique et en C en me permettant d'apprendre le traitement d'image.

Ce projet fut éprouvant et complexe par rapport à ce que l'on voit en cours. Cependant, nous avons réussi à produire un rendu très satisfaisant!

CHAPTER 2

LA RÉPARTITION DES TÂCHES

2.1 Tableau de répartition

Membres \ Tâches	Alexandra	Erwann	Guillaume	Valentin
Traitements d'images				
Chargement d'une image	S			R
Suppression des couleurs	S			R
Rotation manuelle de l'image	R			S
Rotation automatique de l'image		S		R
Réduction de bruit			R	S
Détection de la grille	S			R
Découpage de l'image	R		S	
Nettoyage de l'image	R	S		
Sauvegarde de la découpe	S		R	
Résolution du sudoku				
Programme de résolution		S	R	
Sauvegarde et présentation des résultats		S	R	
Réseau de neurones				
Recherches des différents algorithmes		R	S	
Réseau de neurones pour XOR		R	S	
Sauvegarde et chargement des poids		R	S	
Jeu d'image et importation		R	S	
Réseau de neurones complet		R	S	
UI / Présentation				
Reconstruction de la grille	R		S	
Affichage de la grille	R		S	
Sauvegarde du résultat	R	S		
Interface graphique	R	S		

Table 2.1: Tableau de la répartition des tâches, R = Responsable & S = Suppléant

CHAPTER 3

ETAT D'AVANCEMENT DU PROJET

3.1 Prévisions cahier des charges

3.1.1 Rappel des estimations dans l'avancement des tâches

Voici le tableau rappelant les estimations dans l'avancement des différentes tâches

Tâches \ Soutenances	Soutenance 1	Soutenance 2
Traitement d'image		
Chargement d'une image	100%	100%
Suppression des couleurs	50%	100%
Rotation manuelle de l'image	100%	100%
Rotation automatique de l'image	0%	100%
Réduction de bruit	0%	100%
Detection de la grille	100%	100%
Nettoyage de l'image	0%	100%
Découpe de l'image	75%	100%
Sauvegarde de la découpe	100%	100%
Résolution du sudoku		
Programme de résolution	100%	100%
Sauvegarde et présentation des résultats	100%	100%
Réseau de neurones		
Recherches des différents algorithmes	80%	100%
Réseau de neurones pour XOR	100%	100%
Sauvegarde et chargement des poids	50%	100%
Jeu d'image et importation	40%	100%
Réseau de neurones complet	10%	100%
UI / Présentation		
Reconstruction de la grille	0%	100%
Affichage de la grille	0%	100%
Sauvegarde du résultat	0%	100%
Interface graphique	0%	100%

Table 3.1: Estimation des avancées des différentes tâches

3.1.2 Avancement final

Voici le tableau récapitulatif du respect de notre avancement :

Tâches	Fait en S1	Prévision S2	Fait en S2
Traitement d'image			
Chargement d'une image	100%	100%	100%
Suppression des couleurs	50%	100%	100%
Rotation manuelle de l'image	100%	100%	100%
Rotation automatique de l'image	0%	100%	80%
Réduction de bruit	0%	100%	100%
Détection de la grille	80%	100%	100%
Découpage de l'image	75%	100%	100%
Sauvegarde de la découpe	100%	100%	100%
Résolution du sudoku			
Programme de résolution	100%	100%	100%
Sauvegarde et affichage des résultats	100%	100%	100%
Réseau de neurones			
Recherches des différents algorithmes	80%	100%	100%
Réseau de neurones pour XOR	100%	100%	100%
Sauvegarde et chargement des poids	80%	100%	100%
Jeu d'image et importation	40%	100%	100%
Réseau de neurones complet	10%	100%	90%
UI / Présentation			
Reconstruction de la grille	0%	100%	100%
Affichage de la grille	0%	100%	100%
Sauvegarde du résultat	0%	100%	100%
Interface graphique	5%	100%	100%
Bonus			
Site web	0%	50%	70%
Reconnaissance des caractères manuscrits	0%	50%	80%
Héxadoku solver	0%	100%	100%

Table 3.2: Respect des prévisions de l'avancement du projet

Comme le montre le tableau ci-dessus, nous avons respecté beaucoup de nos prévisions.

CHAPTER 4

ASPECTS TECHNIQUES

4.1 Traitement d'image

4.1.1 Chargement d'une image et suppression des couleurs

Pendant la première partie du projet, nous avions réalisé la partie grayscale et avons utilisé les connaissances acquises lors du TP06 : nous avons chargé l'image grâce à la librairie SDL2 Image qui permet de charger tous les formats (JPEG, PNG etc)).

Nous convertons ensuite l'image au format Red,Green,Blue (RGB)888, puis nous mettons l'image en gris en calculant la nuance de gris de chaque pixel à l'aide de cette fonction:

$$average = 0.3 * red + 0.59 * green + 0.11 * blue;$$

Et nous mettons finalement chaque composante (bleu,vert,rouge) à la valeur "average".

4.1.2 Rotation manuelle et automatique de l'image

Rotation manuelle de l'image

Nous avons légèrement modifié la rotation manuelle entre la première soutenance et la soutenance finale. Nous avons ajouté une étape d'interpolation bilinéaire afin d'améliorer la qualité de l'image après la rotation. La rotation manuelle se décompose donc en plusieurs étapes. La première étape est de créer les différentes surfaces nécessaires à la rotation de l'image à partir du fichier donné par l'utilisateur.

Dans la seconde étape, nous utilisions l'angle en degrés donné par l'utilisateur et nous le convertissons en radiant à l'aide de la formule :

$$angle_r = degrés * \pi / 180$$

Ensuite, nous opérons la rotation sur chaque pixel de la surface à l'aide des formules suivantes

$$X = x * \cos(angle_r) - y * \sin(angle_r)$$

$$Y = x * \sin(angle_r) + y * \cos(angle_r)$$

Dans la troisième étape, nous opérons une interpolation bilinéaire qui consiste à calculer avec les valeurs x et y obtenues ci-dessus la nouvelle valeur du pixel. Pour interpoler chaque pixel: nous calculons une moyenne entre les pixels voisins que nous pondérons selon la distance du pixel interpolé à chacun des pixels originaux. Voici un exemple avec le rouge du pixel (x,y):

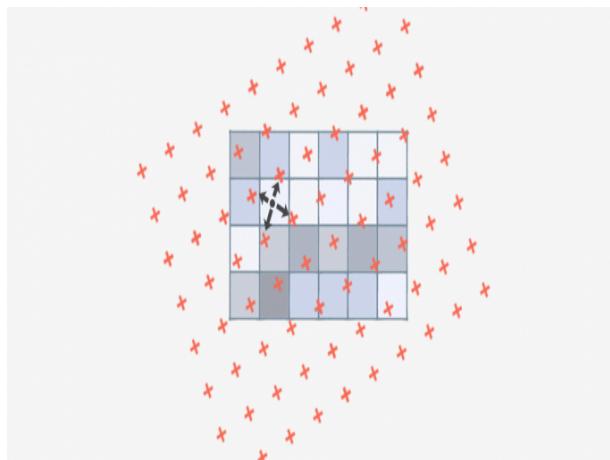
$$fr1 = r4 + ((double)r3 - (double)r4) * (x - (double)xprev)$$

$$fr2 = r1 + ((double)r2 - (double)r1) * (x - (double)xprev)$$

$$r = fr1 + (fr2 - fr1) * (y - (double)yprev)$$

(r1,r2,r3 et r4 étant les valeurs des 3 pixels voisins et le pixel (x,y) (xprev et yprev étant les composantes des pixels voisins)).

Sur un graphique, nous pouvons l'illustrer comme cela:



Nous faisons de même pour le pixel vert et bleu puis nous l'appliquons au pixel d'origine (avant la rotation).

Pour finir, la quatrième et dernière étape consiste à vérifier si le pixel tourné fait bien parti de la surface, ainsi nous opérons la rotation, sinon il n'y a aucun changement effectué.

Rotation automatique de l'image

La rotation automatique de l'image est un prolongement de l'algorithme Hough Transform réalisé pour la première soutenance. La rotation automatique est plutôt simple du fait que l'on ait tous les angles θ des lignes obtenues grâce à l'algorithme de Hough, nous construisons un histogramme à l'aide des angles θ et nous choisissons la valeur maximale. Cette valeur sera l'angle utilisé dans la rotation manuelle.

4.1.3 Binarisation avec réduction de bruit

Certaines images de sudoku que nous avons à traiter sont fortement ébruitées et donc par conséquent plus compliquées voir impossible à traiter, que ce soit pour la détection de la grille ou pour la reconnaissance des caractères après découpe. Pour nous permettre de traiter ces images, nous avons implémenté une réduction de bruit qui va améliorer la qualité de l'image et nous permettre de traiter certaines images dès lors inutilisables.

Pour ce faire plusieurs filtres peuvent être utilisés. Nous avons décidé d'utiliser d'implémenter plusieurs de ces filtres pour savoir lesquelles sont plus efficaces et plus utiles à utiliser. Il y a, tout d'abord, le filtre médian dont le principe est de faire une liste triée par ordre croissant des valeurs voisines d'une case et de remplacer cette case par la valeur médiane de la liste. Le but étant de faire cela pour tous les pixels de l'image. Par exemple, prenons le cas ci-dessous, nous avons une valeur beaucoup plus haute que les autres au centre, nous mettons alors ces valeurs dans une liste que nous trions. La valeur médiane est ici 7. Nous remplaçons alors la valeur du centre par 7. Ce filtre nous permet d'un peu réduire le bruit mais pas suffisamment pour pouvoir traiter la plupart des images correctement. Nous avons donc décidé de ne pas utiliser ce filtre.

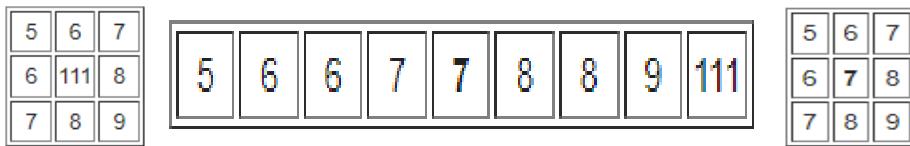


Figure 4.1: Principe du filtre médian

Le filtre que nous avons décidé de garder est le filtre Otsu. Le principe de ce filtre est assez simple. Tout d'abord, nous parcourons tous les pixels de l'image, dont les couleurs ont étéées supprimé au préalable, pour récupérer la valeur RGB de chaque pixel. Nous créons donc un histogramme à partir de ces valeurs de 0 à 255. Le but à partir de cet histogramme va être de trouver le seuil optimal pour notre binarisation. Nous allons donc appliquer le filtre d'Ostu, qui va séparer les pixels en deux catégories, ceux faisant parti de l'avant-plan (foreground) et ceux faisant parti de l'arrière-plan (background). Pour ce faire, nous allons utiliser d'abord 0 puis 1 puis 2, et ainsi de suite jusqu'à 255 comme seuil. Le seuil ayant la plus grande variance calculée entre les deux catégories (foreground et background), avec la formule ci-dessous. La formule de Otsu étant:

$$\sigma = Wb * Wf * (\mu_b - \mu_f) * (\mu_b - \mu_f)$$

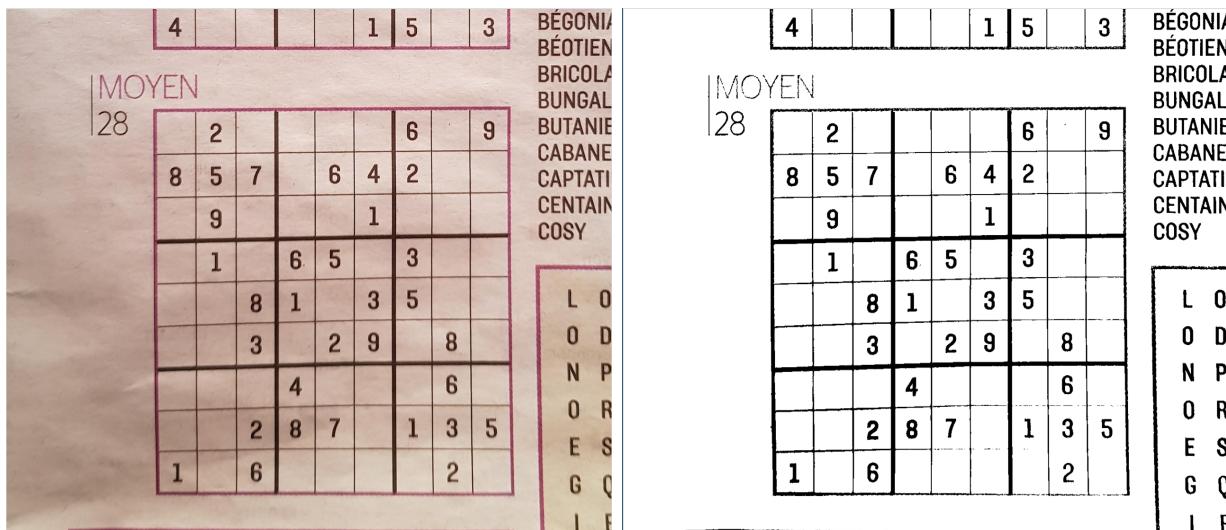
Wb étant le nombre de pixel en dessous du seuil que nous testons divisé par le nombre total de pixels de l'image.

Wf étant nombre de pixel au dessus du foreground divisé par le nombre total de pixels de l'image.

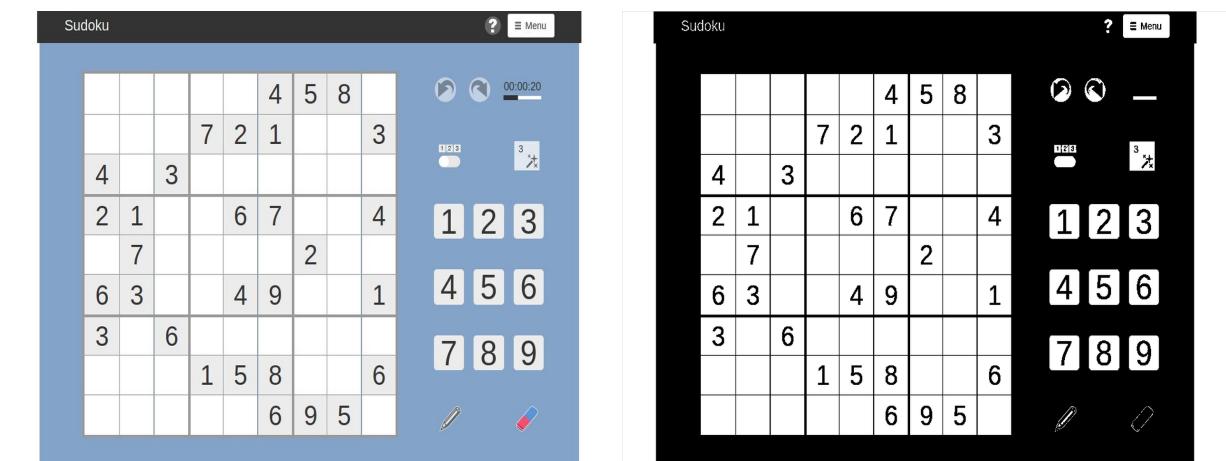
mu b étant la moyenne d'intensité du background donc chaque collone de l'histogramme multiplié par le niveau de gris, le tout divisé par le nombre total de pixels dans le background.

mu f étant la moyenne d'intensité du foreground donc chaque collone de l'histogramme multiplié par le niveau de gris, le tout divisé par le nombre total de pixels dans le foreground.

Nous avons maintenant le seuil grâce au filtre Otsu. Le principe de la binarisation va être de faire passer en blanc tous les pixels dont le niveau de gris est inférieur au seuil et de passer en noir les autres. Cela nous donne par exemple:



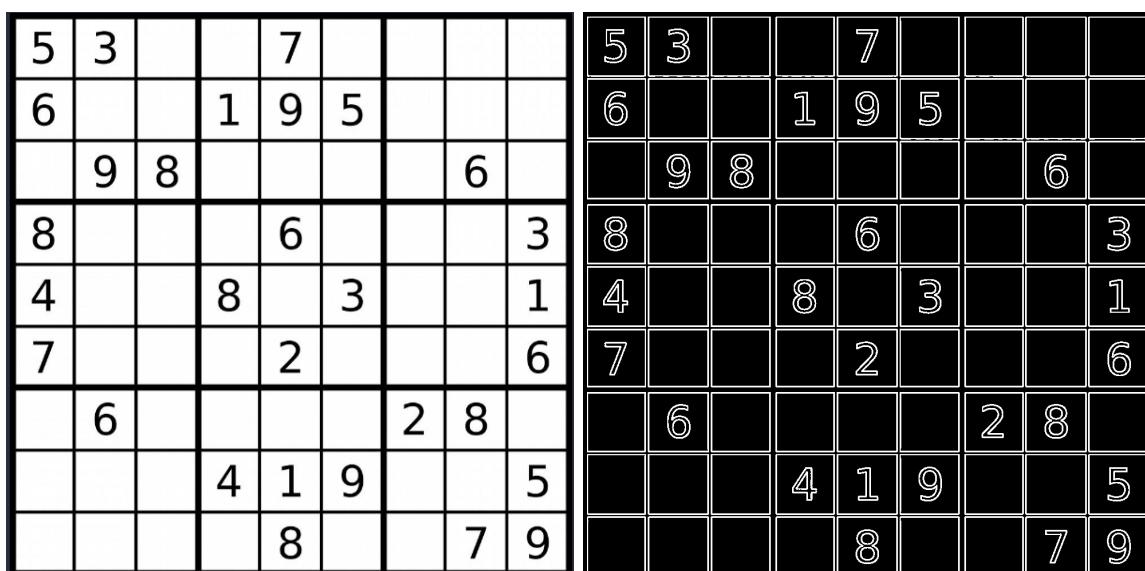
Exemple "Image 2" du filtre d'Otsu



Exemple "Image 3" du filtre d'Otsu

4.1.4 Détection de la grille et de la position des cases

Pour la première soutenance, nous avions commencé par reconnaître les contours de l'image afin de pouvoir détecter la grille par la suite. Voici un exemple sur le premier Sudoku:



Exemple du filtre de Sobel

Pour ce faire, nous avions utilisé le filtre de Sobel qui se base sur deux matrices de Convolution:

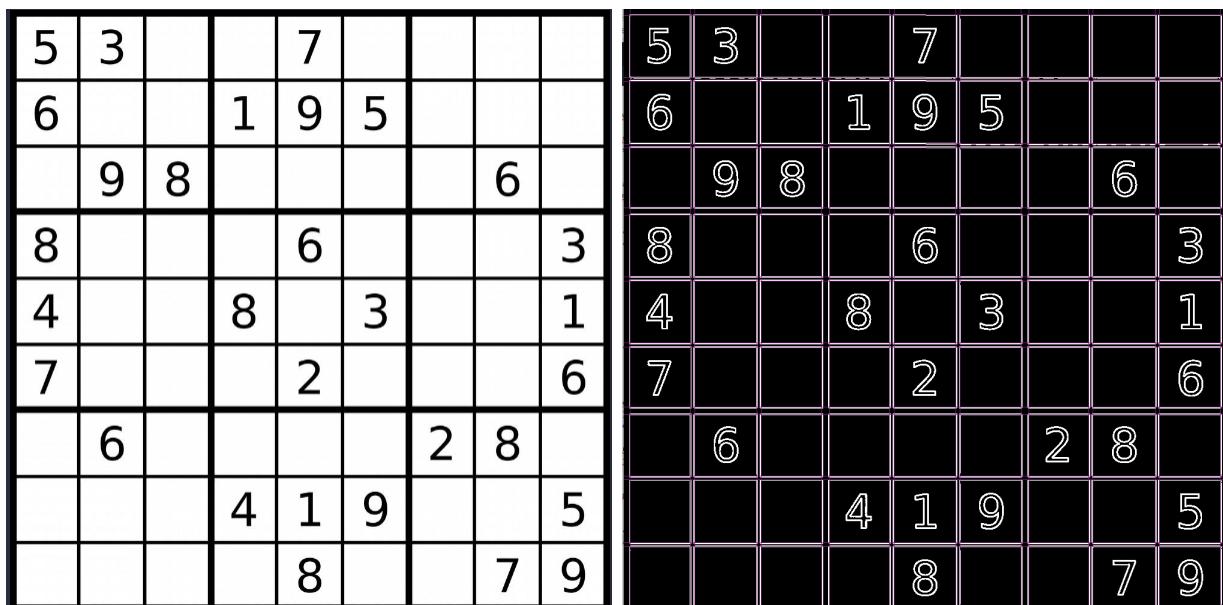
$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A} \quad \text{et} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{A}$$

Matrice de Convolution

Nous calculons par la suite les gradients horizontaux (pour chaque pixel, nous prenons tous ses pixels voisins puis nous les multiplions par la matrice G_x), et les gradients verticaux (même procédé mais avec la matrice G_y). Nous calculons ensuite la norme du gradient. Si celle-ci est plus grande que 128 nous mettons un pixel blanc sinon un pixel noir.

Une fois que nous avons détecté les contours, il fallait repérer les lignes du sudoku pour pouvoir faire le découpage. C'est sûrement l'une des parties les plus complexes du traitement de l'image car il faut réussir à repérer les lignes sans repérer les caractères.

Nous avons utilisé le Transformée de Hough. Voici ce que donne cet algorithme après l'implémentation sur le premier Sudoku:



Exemple du transformé de Hough

Comme nous pouvons le voir, les lignes violettes représentent ce que l'algorithme a réussi à "trouver" comme lignes. Nous pouvons voir qu'il arrive à repérer les lignes sans les caractères. Le transformée de Hough fonctionne de la façon suivante:

- Nous commençons par initialiser une matrice que nous nommons matrice d'accumulation. Nous allons aussi initialiser toutes les valeurs cosinus et sinus de 0 à 180 degrés (pour ne pas avoir à refaire le calcul à chaque fois).

- Une fois cela fait, si nous rencontrons un pixel blanc, nous allons calculer toutes ses droites polaires correspondantes en faisant varier une variable Θ de 0 à 180 degrés. L'équation de la droite polaire est la suivante:

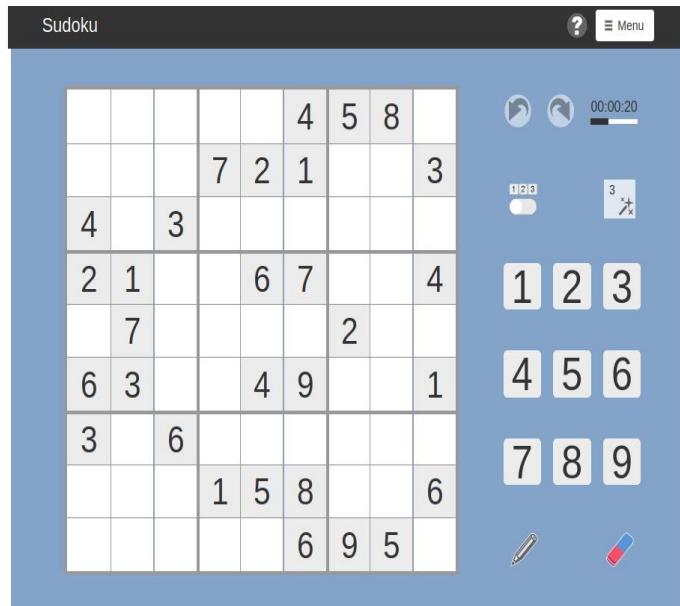
$$r = (x - (largeur/2)) * \cos(\Theta) + (y - (longueur/2)) * \sin(\Theta)$$

Nous incrémentons ainsi de 1 la matrice d'accumulation: matrice accumulation[r][theta] + 1. Avant cela nous ajoutons la diagonale de l'image à r pour éviter les valeurs négatives. En même temps que cela, nous cherchons la valeur maximum de cette matrice.

- Par la suite, nous allons reparcourir notre matrice d'accumulation afin de retenir les "pics" de cette dernière. La première chose à faire et de ne retenir que les valeurs de la matrice supérieures au max*0.42. Ensuite, nous allons chercher le pic local maximum sur 9*9 pixels. Uniquement celui-ci sera retenu.

- Enfin, il ne nous reste plus qu'à reconvertis les rho et theta que nous avons retenus en coordonnées cartésiennes et de tracer les lignes.

Nous nous étions arrêtés là pour la première soutenance, cependant il nous manquait encore à repérer uniquement la grille finie. Par exemple sur l'image 03 donnée dans le sujet nous devions passer de ça:



Pour arriver à ce résultat, nous avons dû travailler sur les lignes du Hough transform. Nous commençons par prendre les lignes horizontales et verticales (en utilisant les angles θ des lignes). Ensuite, on calcule pour chaque ligne, on calcule les intersections de ces lignes (et on vérifie si il y a une intersection sinon on passe à la prochaine ligne). On fait ça pour 4 lignes (en retenant à chaque fois les intersections des lignes). Une fois les 4 points obtenus, on vérifie si ces 4 points forment un carré (on vérifie si toutes les longueurs sont égales à plus ou moins 2 pixels d'écart). Il est inutile de calculer le degré car nous prenons uniquement les lignes horizontales et verticales.

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

Nous obtenons donc un carré, il nous suffit juste de calculer le point avec les coordonnées le plus petit et le point avec les coordonnées le plus grand, et on reconstruit une image avec tous les pixels dans ce carré. Il nous reste plus qu'à enregistrer cette image !

4.2 Découpage de la grille

4.2.1 Découpage des cases

Pour le découpage de l'image, nous utilisons une image prétraitée afin qu'il y ait le moins d'espace possible autour de la grille. Nous découpions en surfaces égales la surface, de façon à avoir une surface par case de la grille, pour ce faire nous utilisons les pixels de la surface.

Nous procédons de la même manière pour toutes les cases de la grille, tant que le pixel appartient à la case, c'est à dire tant que sa position est inférieure en largeur et en longueur de la case alors :

- si le pixel appartient à l'image : nous utilisons le pixel correspondant sur la surface de base
- si le pixel n'appartient pas à l'image, : celui-ci devient un pixel noir

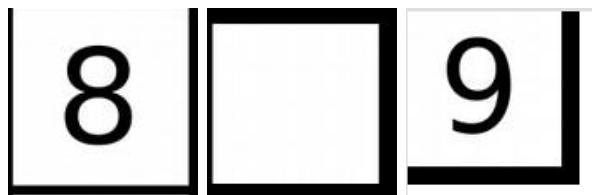


Figure 4.2: Exemples de découpes de l'image

4.2.2 Nettoyage et sauvegarde des cases

Nettoyage des cases

Lors de la découpe, nous pouvons apercevoir sur certaine case les lignes de la grille du sudoku qui rendent la détection des caractères plus difficile pour le réseau de neurones. Ainsi il nous a fallut effectuer un "nettoyage" de chaque case afin de conserver uniquement le chiffre s'y trouvant. Ainsi, nous prenons une à une chacune des cases récupérées lors de la découpe et pour chaque pixel se trouvant sur le contour de l'image, nous le changeons en un pixel blanc.

Ainsi, tous les pixels se trouvant à une coordonnée inférieure à un sixième de la largeur de la surface ou supérieure à cinq sixièmes de la largeur de la surface seront remplacés par un pixel blanc. Le même procédé est effectué pour la longueur de l'image. Pour créer le pixel blanc nécessaire, nous modifions le code RGB du pixel en mettant toutes les valeurs à 255. Le résultat obtenu est tel :

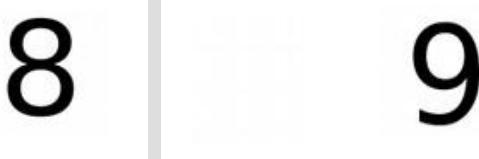


Figure 4.3: Exemples de découpe après le nettoyage

Sauvegarde de la découpe

Lors de la découpe du sudoku, nous avons sauvegardé une nouvelle image pour chaque case découpée. Chaque image a pour nom "*son_indexage.png*". L'indexage étant compris entre 0 et 80 et commence en haut à gauche pour finir en bas à droite. Ces images seront récupérées et utilisées ultérieurement par l'intelligence artificielle pour détecter les caractères.

Lors de la seconde partie du projet, nous avons modifié cette sauvegarde pour qu'elle s'effectue dans un dossier Cells qui se créer lors de la découpe, afin que toutes les images soient regroupées dans un même dossier à part.

```
[epita@localhost:~/OCR/src/cells]$ ls
00.png 09.png 18.png 27.png 36.png 45.png 54.png 63.png 72.png
01.png 10.png 19.png 28.png 37.png 46.png 55.png 64.png 73.png
02.png 11.png 20.png 29.png 38.png 47.png 56.png 65.png 74.png
03.png 12.png 21.png 30.png 39.png 48.png 57.png 66.png 75.png
04.png 13.png 22.png 31.png 40.png 49.png 58.png 67.png 76.png
05.png 14.png 23.png 32.png 41.png 50.png 59.png 68.png 77.png
06.png 15.png 24.png 33.png 42.png 51.png 60.png 69.png 78.png
07.png 16.png 25.png 34.png 43.png 52.png 61.png 70.png 79.png
08.png 17.png 26.png 35.png 44.png 53.png 62.png 71.png 80.png
```

Figure 4.4: Fichiers résultant de la découpe

4.3 Résolution du sudoku

Sudoku

Pour résoudre le sudoku, nous récupérons une grille contenue dans un fichier. Après avoir récupéré celle-ci, nous parcourons chaque case de la grille en essayant de la remplir tout en vérifiant qu'il n'y a pas plusieurs fois le même nombre dans la même ligne, colonne ou diagonale. Si la grille n'est pas valide, nous retournons sur la dernière grille valide et testons avec d'autres chiffres jusqu'à en trouver une. Après avoir résolu la grille, nous l'affichons pour bien montrer qu'elle a été résolue et nous la sauvegardons dans un fichier.

```
main_solver: Grid before solver
53. .7. ...
6.. 195 ...
.98 ... 6.

8... 6. ... 3
4.. 8.3 ... 1
7... 2. ... 6

.6. ... 28.
... 419 ... 5
... 8. ... 79

main_solver: Grid solved
534 678 912
672 195 348
198 342 567

859 761 423
426 853 791
713 924 856

961 537 284
287 419 635
345 286 179

[epita@localhost:~/OCR/Solver]$ make
gcc -Wall -Wextra -MM -c -o main.o main.c
gcc -Wall -Wextra -MM -c -o solver.o solver.c
gcc -o solver main.o solver.o

[epita@localhost:~/OCR/Solver]$ ./solver 01 -solve test_grid_01
main_solver: solve -solve.

[epita@localhost:~/OCR/Solver]$ cat test_grid_01
53. .7. ...
6.. 195 ...
.98 ... 6.

8... 6. ... 3
4.. 8.3 ... 1
7... 2. ... 6

.6. ... 28.
... 419 ... 5
... 8. ... 79

[epita@localhost:~/OCR/Solver]$ cat test_grid_01.result
534 678 912
672 195 348
198 342 567

859 761 423
426 853 791
713 924 856

961 537 284
287 419 635
345 286 179
```

Figure 4.5: Affichage et sauvegarde de la grille résolue

BONUS : Hexadoku

La résolution de l'hexadoku fonctionne sur le même principe que celle du sudoku mais sur une grille de 16 par 16 donc avec des caractères allant de 0 à F.

Hexadoku to solve:	Hexadoku solved:
.... 29.. ...8	61FC BA74 29DE 5038
0D2B ..C5 A..F ..49	0D2B E8C5 A31F 6749
5... 9F.. ...4 1...	5A37 9F62 0B84 1DCE
.8.E ..D. ...6. F2B.	984E 01D3 576C F2BA
.59. 8.4. ...B ..D3	F592 864E 7CAB 01D3
E.6. .BA. .F.2 .C..	E36D 1BA7 4F02 8C95
.0.. F.9. 7...	B0A8 F59C 3DE1 7624
C714 2... .85. EF..	C714 203D 9856 EFAB
D.CF 7E.. 859. .4.6	DBCF 7E20 8593 33A4
.67. ..8. B..A .E5.	1626 79CD 8FB1 4A3E
.... ...B C..D	503E 50A4 1BF6 27C9
...A DE.0 2.F.	8D14 8A63 59DE C02B
42.3 5CF. .AB. .87.	F742 035C F6EA BD98
..E. D9B. C...	71AF E5D9 B1C0 7843
7... 1... .E..	6279 B632 0814 F5DA
.CD. ..EA ..39	EC8C D147 EA62 39B5

Figure 4.6: Résolution de l'hexadoku

4.4 Réseau de neurones

Dans l'objectif de reconnaître les caractères écrits dans chacune des cases, nous utilisons un Réseau de neurones (RN) qui à pour but de nous dire via l'image d'une case et du chiffre écrit à l'intérieur de quel chiffre il s'agit afin de numériser le sudoku présenté pour le résoudre. Lors de la première soutenance, l'objectif principal était de comprendre le fonctionnement de ces RN afin d'en user de la meilleure des façons par la suite.

Fonctionnement d'un réseau de neurones

Afin de détecter de quel chiffre il s'agit, le RN doit apprendre en amont, via une base de données plus ou moins grande et diversifiée, à faire la différence entre chaque chiffre écrit. Pour ce faire, un réseau de neurones imite les réseaux cérébraux humains.

Forward Propagation

Il s'agit de la propagation des calculs partant de la couche d'entrée (input layer) passant à travers chaque poids et neurones des couches cachées intermédiaire (hidden layers) jusqu'aux différents neurones de sortis (output layer).

Chaque neurone utilise une fonction d'activation. La plus standard et celle que nous avions utilisé pour le XOR est la fonction sigmoïd.

Fonction d'activation : Sigmoïd

Elle se définit ainsi :

$$\frac{1}{1 + \exp(-x)}$$

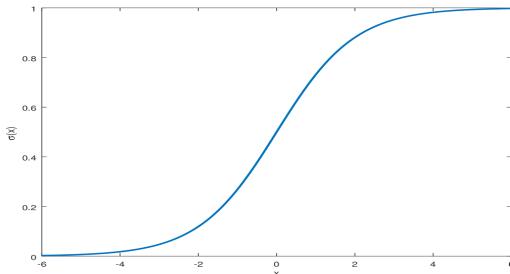


Figure 4.7: Tableau GD Courbe de la fonction sigmoid

L'intérêt premier de cette fonction est sa capacité à faire "fuir" les résultats, sa courbe permet de ne pas avoir de résultat "floatant" vers 0.5 mais permet de rapidement se rapprocher de 1 ou 0 dès lors que l'on s'éloigne de la médiane contrairement à une fonction linéaire. De plus, la fonction Sigmoïde donne une valeur entre 0 et 1, une probabilité. Elle est donc très utilisée pour la classification binaire.

Back Propagation

La back propagation a pour but de calculer le taux d'erreur du réseau afin d'essayer de réduire ce nombre d'erreur en ajustant les poids et biais du réseau.

Pour ce faire, nous utilisons l'algorithme de la GD qui permet de trouver le minimum dans toutes fonctions convexes et nous permet ici de trouver le taux minimum d'erreurs que le réseau puisse faire.

Deux paramètres entrent en jeu. Le "learning rate" qui multiplie l'action de la GD et permet théoriquement d'atteindre le minimum plus rapidement mais un learning rate trop grand peut avoir l'effet inverse :

A contrario, un learning rate trop faible nous demandera davantage d'itérations pour trouver l'objectif optimal. C'est alors qu'intervient le second paramètre qui est le nombre d'epochs (d'itération) d'apprentissage. Ainsi, il est de notre ressort de trouver les paramètres idéaux pour un apprentissage efficace et rapide.

Pour user de la GD, nous utilisons la dérivé de la fonction d'activation : la sigmoïd dérivée.

Dérivé de la fonction d'activation : dSigmoid

Elle se définit comme suit :

$$x * (1 - x)$$

Mise à jour des poids et biais

A chaque fin d'itération, il est primordial de mettre à jour tous les poids et tous les biais du réseau dans le but d'optimiser les prochaines forward propagations. Pour ce faire, nous utilisons les résultats de la GD qui nous permet de diminuer le taux d'erreurs.

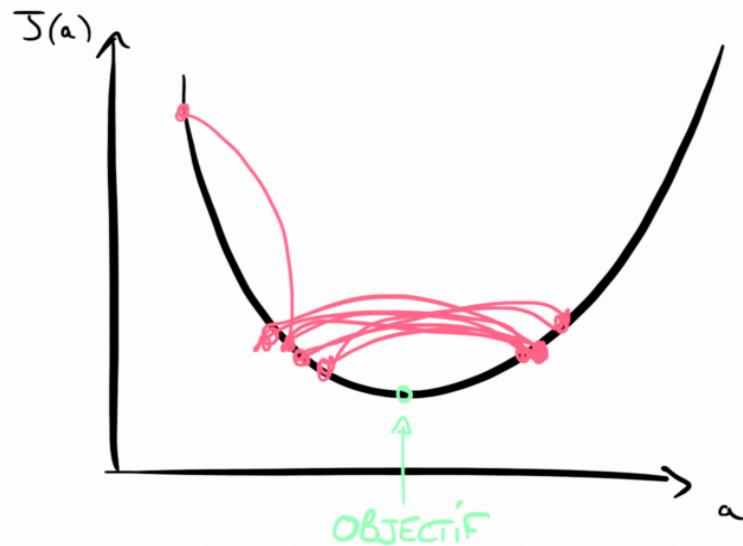


Figure 4.8: Tableau GD avec learning rate trop grand

4.4.1 Preuve de concept (porte ou-exclusive)

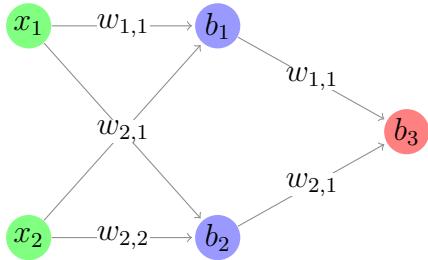
Pour la soutenance intermédiaire, il nous a été demandé une preuve de concept d'un réseau de neurones basique permettant d'apprendre le comportement de la Porte ou-exclusive (XOR) qui se définit comme suit :

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4.9: Tableau XOR

Paramètres du réseau

Pour l'apprentissage, nous avons utilisé un réseau de neurones basique dont les paramètres peuvent être représentés ainsi :



2 inputs(entrés) notées (x_1, x_2)

1 couche cachée avec 2 neurones (en bleu)

1 output(sortie) en rouge.

Chaque liaison est constituée d'un poids($W_{i,i}$) et chaque (B_i) est un biais.

Ainsi pour le neurone 1 de la couche cachée, la propagation se fait via la fonction :

$$f(x) = x_2 * W_{2,1} + x_1 * W_{1,1} + B_1$$

Et ce pour chacun des neurones du réseau. Grâce à la backpropagation qui permet de calculer le taux d'erreur du réseau ainsi que la fonction mettant à jour les poids et biais pour les ajuster. Nous avions réussi à obtenir des résultats convaincants.

```

iteration : 0
m_dimensions: 4 rows - 1 cols
[ 0.676848 ]
[ 0.681624 ]
[ 0.707081 ]
[ 0.643640 ]

```

Figure 4.10: Première itération

```

iteration : 1000000
m_dimensions: 4 rows - 1 cols
[ 0.998292 ]
[ 0.998153 ]
[ 0.001981 ]
[ 0.001641 ]

```

Figure 4.11: Dernière itération

Les 2 premières lignes correspondent aux inputs [1, 0] et [0, 1] qui obtiennent ainsi une probabilité moyenne de 99.8% de sorties égales à 1.

Les 2 dernières lignes correspondent aux inputs [0, 0] et [1, 1] qui obtiennent ainsi une probabilité moyenne de 0.001% de sorties égales à 1 soit près de 99.9% de sorties égales à 0.

Ce sont des résultats tout à fait acceptables pour un réseau de neurones à 2 neurones cachés.

Sauvegarde et chargement des poids

Après que le réseau de neurones soit entrainé, il nous faut sauvegarder tous les poids et biais du réseau afin de faire une prédiction en fonction des entrées que nous voulons via les paramètres entraînés au préalable.

Pour ce faire, nous utilisons un simple fichier texte dans lequel seront représentées les matrices des poids et biais de chaque couche. Dans le cas du réseau de neurones pour le XOR, nous obtenons ainsi 4 parties, les 2 premières correspondent à la première couche (hidden layer) et les 2 suivantes respectivement aux poids et biais de la deuxième couche (output layer).

Du fait de l'architecture simpliste des fichiers de sauvegardes des poids et biais, il nous est facile de les importer pour des prédictions futures.

```
m_dimensions: 16 rows - 1 cols
[ 0.161432 ]
[ 0.657850 ]
[ 0.641609 ]
[ -0.219941 ]
[ 0.178667 ]
[ 0.268243 ]
[ -0.283402 ]
[ 0.392328 ]
[ -0.043898 ]
[ 0.229975 ]
[ 0.127252 ]
[ -0.334991 ]
[ 0.683396 ]
[ -0.010784 ]
[ 0.310754 ]
[ 0.104214 ]

m_dimensions: 16 rows - 16 cols
[ -0.156105  0.246165  0.383707  -0.464095  -0.004069  -0.021687  -0.484409  -0.105119  -0.328379  0.018186  -0.037513  -0.689080  -0.036456  -0.310773  -0.026124  -0.181324 ]
[ -0.076250  -0.392636  -0.532552  -0.615115  0.391510  0.524111  -0.500286  0.766928  0.064086  -0.384877  0.331852  -0.805473  0.062808  0.659600  -0.771101  0.271440 ]
[ 0.431756  -0.609468  0.343764  -0.769056  -0.037123  0.657428  0.268705  -1.047144  -0.370922  -0.359424  -0.497416  -0.106465  0.588844  0.719936  0.458406  -0.424042 ]
[ 1.263103  -0.519933  -0.034661  0.204853  0.366664  -0.340477  -0.036507  -0.932455  -0.120816  0.131343  0.033210  -0.787841  -0.888500  -0.042827  0.583485  0.881001 ]
[ -0.865310  0.275423  -0.289741  -0.144323  0.562306  -0.517185  0.602673  0.320107  0.521818  0.253051  0.153140  0.943059  -0.283328  -0.911054  -0.515765  0.747189 ]
[ 0.556517  1.296763  0.657895  0.252273  -0.089885  0.643640  -0.697639  0.046540  1.027043  0.244716  -0.195671  -0.638112  -0.472371  -0.547513  0.486911  -0.637816 ]
[ 0.441057  -0.219674  0.482179  -0.070064  -0.406795  0.347004  -0.075063  0.099833  -0.286137  -0.182199  0.329390  0.818619  -0.090397  -0.197341  -0.585720  -0.364057 ]
[ -0.607600  -0.351514  -0.252084  1.182568  0.146830  -0.251875  -0.273751  -0.240598  -1.164033  -0.428887  -0.044861  -0.336008  -0.378489  0.300370  0.556639  1.396594 ]
[ 0.237117  0.053019  -0.064723  -0.697809  -0.316317  -0.934702  -0.496481  0.638625  0.398377  0.810857  0.342317  -0.282971  0.527892  -0.410594  0.601668  0.187788 ]
[ -0.578599  0.056515  0.383922  0.591039  -0.409594  -0.376524  0.368991  -0.147570  0.576182  0.459327  0.403212  0.336408  -1.219075  0.448952  1.061886  -0.215509 ]
```

Figure 4.12: weights file

4.4.2 Réseau de neurones principal

Base d'images pour l'entraînement du réseau

Pour entraîner notre réseau de neurones qui a pour objectif de repérer quel chiffre est écrit sur une case du sudoku, nous avons préparé un dataset(base de données) de plusieurs images de chacun des chiffres allant de 0 à 9 . Il nous sera possible dans le futur d'importer les lettres A, B, C, D, E et F dans le cas où nous travaillerons sur des sudoku hexadécimaux.

Pour ce faire, nous utilisons des polices googles et un programme nous permettant d'exporter des fichiers incluant les différentes images de 28x28 pixels distribués dans 10 dossiers correspondants à chaque chiffre en fonction de leur label respectif.

Mise en place du réseau principal de neurones

Nous avons décidé d'insérer comme entrée à notre réseau de neurones des images de 28 pixels par 28 pixels soit

$$28 * 28 = 784$$

neurones pour la couche 1.

Ensuite, nous n'utilisons qu'une seule couche cachée avec un nombre de neurones inférieur à 784 (16 neurones finalement décidé de manière arbitraire) puis 10 sorties indiquant chacune d'elles la probabilité qu'il s'agissent du nombre dont elle réfère.

Ainsi, la décision est prise en fonction du neurone de sortie qui délivrera la probabilité la plus forte (proche de 1).

Contrairement au premier réseau de neurones, nous avons préféré changer de fonction d'activation pour ce réseau plus grand et complet car les résultats que nous avons obtenus avec la fonction sigmoid ont été très en dessous de nos attentes.

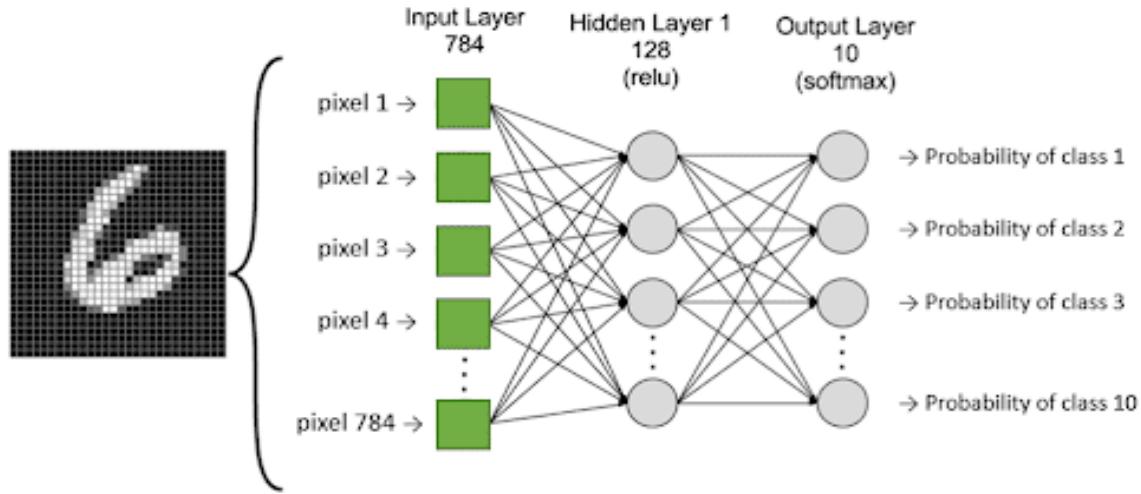


Figure 4.13: Exemple d'architecture viable

Fonction d'activation : Relu

La fonction Rectified Linear Unit (ReLU) est la fonction d'activation la plus simple et la plus utilisée. Elle se définit comme suit :

$$Relu(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

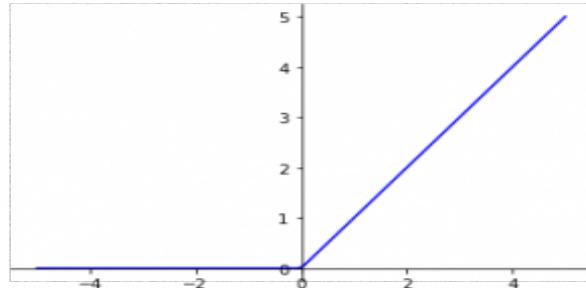


Figure 4.14: Tableau GD Courbe de la fonction Relu

Cette fonction permet d'effectuer un filtre sur nos données. Elle laisse passer les valeurs positives ($x \geq 0$) dans les couches suivantes du réseau de neurones. Elle est utilisée presque partout mais surtout pas dans la couche final

Dérivé de la fonction d'activation : dRelu

Pour la back propagation, nous avons besoin de la dérivée de cette fonction d'activation. Elle s'exprime tout simplement comme suit:

$$dRelu(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Fonction d'activation : Softmax

Défini comme suit :

$$y_i = \frac{\exp(x_i)}{\sum_{p \geq 0} \exp(x_p)}$$

La fonction d'activation **Softmax** est utilisée lors de la dernière couche afin de forcer le réseau de neurones à renvoyer des vecteurs de probabilité, ce qui semblait plus adéquat pour notre problème de classification à multiples réseaux de sorties. En effet elle permet d'extrapoler le meilleur résultat parmi plusieurs neurones et donc de prendre une décision parmi les 10 neurones de sorties.

Résultats obtenus pour des chiffres numériques

Il est crucial de se rendre compte le nombre important de configurations différentes que nous avons testées dans le but d'avoir des résultats convaincants.

Au départ, nous avions voulu avoir un dataset incluant les images blanches (vides) à la place des cases comportant un 0 car nul besoin de repérer ce chiffre au sein d'une grille de sudoku. Ce faisant, nous obtenions des résultats parfait pour la détection des cases vides mais un résultat très incertain pour la reconnaissance des bons chiffres.

Ainsi, nous obtenions 30 à 50 % de précision sachant que 10 à 15 % provenait des images vides bien repérées et le reste, d'un très faible apprentissage sur les chiffres restants.

Afin d'essayer de pallier à ce problème, nous nous sommes dit que nous allions gérer les cases vides en dehors du réseau, c'est à dire avant même de rentrer dans le RN. Pour ce faire, nous repérons le taux de cases noir et si ce dernier est trop faible et est inférieur à un certain seuil que nous avons fixé arbitrairement, la case est considérée comme vide et ne passe pas par la forward propagation.

De ce fait, nous avons entraîné un réseau avec des chiffres de 0 à 9 pensant que le 0 n'allait pas nous déranger mais sans réelle surprise, le taux de précision de notre réseau était seulement de 20 %.

Ainsi nous avons changé toutes nos structures, nos méthodes d'entraînement avec d'autres fonctions d'activations mais c'est seulement lorsque nous avons essayé d'entraîner nos images les unes après les autres que le réseau nous a instantanément permis d'obtenir un résultat convaincant de 84% de précision sur un échantillon de 200 nouvelles images tests.

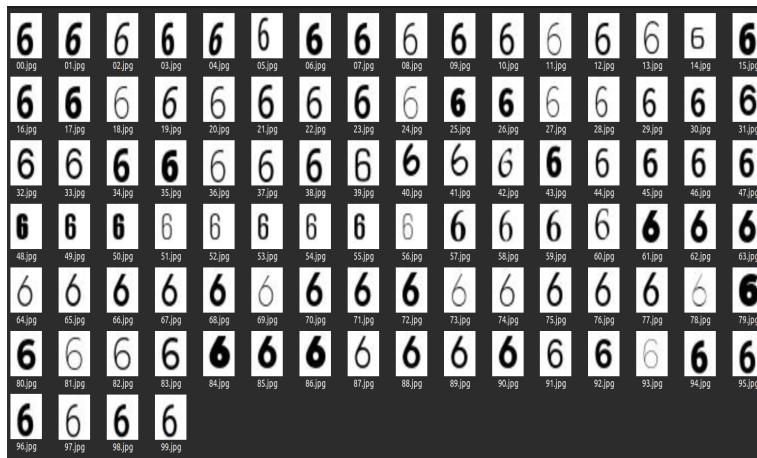


Figure 4.15: Base de données de caractères (dossier 6)

```

got 1 - expected 1 - Correct
got 9 - expected 1 - Incorrect
got 9 - expected 9 - Correct
got 6 - expected 6 - Correct
got 9 - expected 9 - Correct
got 4 - expected 4 - Correct
got 8 - expected 8 - Correct
got 9 - expected 9 - Correct
got 3 - expected 3 - Correct
got 3 - expected 3 - Correct
got 9 - expected 9 - Correct
got 9 - expected 9 - Correct
got 6 - expected 6 - Correct
got 2 - expected 2 - Correct
got 8 - expected 8 - Correct
got 6 - expected 8 - Incorrect
got 4 - expected 4 - Correct
Number of correct predictions: 165 over 200 tests
accuracy: 0.825000

```

Figure 4.16: Résultat des prédictions

Malgré ce bon fonctionnement du réseau, les images séparés par nos soins et issues des grilles de sudokus ne sont pas bien reconnus car elles diffèrent de façon conséquente avec les images du dataset d'entraînement.

BONUS : Résultats sur des chiffres manuscrits

Dans le but d'entraîner notre RN pour détecter des chiffres manuscrits, nous avons utilisé la même méthode mais à partir d'une base de données en ligne : **MNIST**.



Figure 4.17: Base de données MNIST

Ainsi divisé en 10 fichiers distincts, il nous a été possible d'entraîner notre réseau jusqu'à obtenir des résultats corrects de 70% de précision.

```

got 6 - expected 6 - Correct
got 1 - expected 1 - Correct
got 4 - expected 4 - Correct
got 4 - expected 5 - Incorrect
got 9 - expected 9 - Correct
got 5 - expected 5 - Correct
got 5 - expected 6 - Incorrect
got 8 - expected 5 - Incorrect
got 3 - expected 3 - Correct
got 8 - expected 8 - Correct
got 4 - expected 4 - Correct
got 3 - expected 5 - Incorrect
got 9 - expected 9 - Correct
got 9 - expected 9 - Correct
Number of correct predictions: 145 over 200 tests
accuracy with MNIST dataset: 0.725000

```

Figure 4.18: Résultat des prédictions (avec MNIST)

4.5 Interface graphique

Pour l'interface graphique, nous voulions une interface facile et agréable à utiliser. Pour cela, nous avons utilisé Glade et CSS pour créer et personnaliser les différents éléments de l'interface ainsi que GTK pour mettre en place l'ensemble du projet.

Interface utilisateur

Pour commencer, nous avons créé la fenêtre de Interface utilisateur (UI) et y avons ajouté cinq boutons, deux labels et un scale.

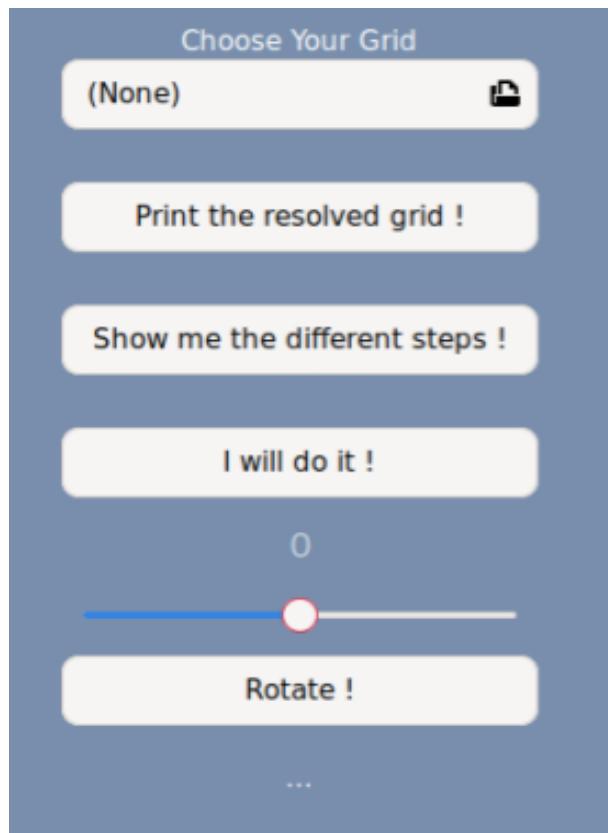


Figure 4.19: Différents éléments de l'interface

Les deux labels changent lorsque l'utilisateur interagit avec l'interface afin de le guider. Nous avons également ajouté un scale à l'interface qui permet à l'utilisateur de choisir l'angle qu'il désire pour la rotation manuelle entre -180 et 180 degrés. Ce scale est associé au bouton "Rotate !" qui affichera la rotation manuelle sous l'angle choisi.

Nous avons également ajouté un File Chooser qui permet à l'utilisateur de choisir l'image qu'il souhaite utiliser sur l'application. Cependant, l'utilisateur se doit de choisir un fichier avec une extension .png ou .jpg, sinon le label lui indiquera de changer de fichier s'il souhaite exécuter le programme.

Pour finir, les deux boutons restants sont les différents modes d'impression du résultat que nous avons implémenté sur l'interface.

- Choisir l'option "Print the resolved grid!" permet à l'utilisateur d'avoir la grille directement résolue.

- Choisir l'option "I will do it !" permet à l'utilisateur de voir chaque étape du programme jusqu'à la reconstitution de la grille. Cependant l'utilisateur est libre de choisir la durée pendant laquelle il souhaite voir chaque étape. Une nouvelle étape est affichée pour chaque clic sur le bouton.

Par la suite, nous avons ajouté l'espace destiné à l'affichage de l'image ainsi qu'un logo OCR. Cependant Glade ne permettant pas une interface très personnalisée, nous avons ajouté du CSS afin de modifier les couleurs des différents éléments de l'interface mais également la taille et la police des labels de l'interface. Le résultat obtenu nous plaît beaucoup puisque nous pensons avoir réalisé une interface simple d'utilisation et visuellement agréable pour l'utilisateur.

Voici le résultat obtenu pour l'interface graphique de notre projet OCR :

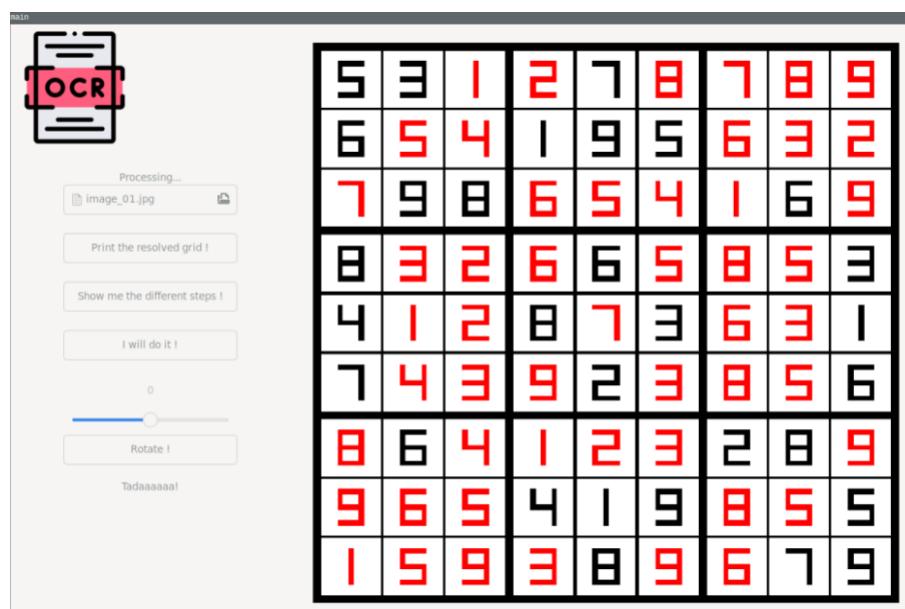


Figure 4.20: Interface réalisée sur Glade

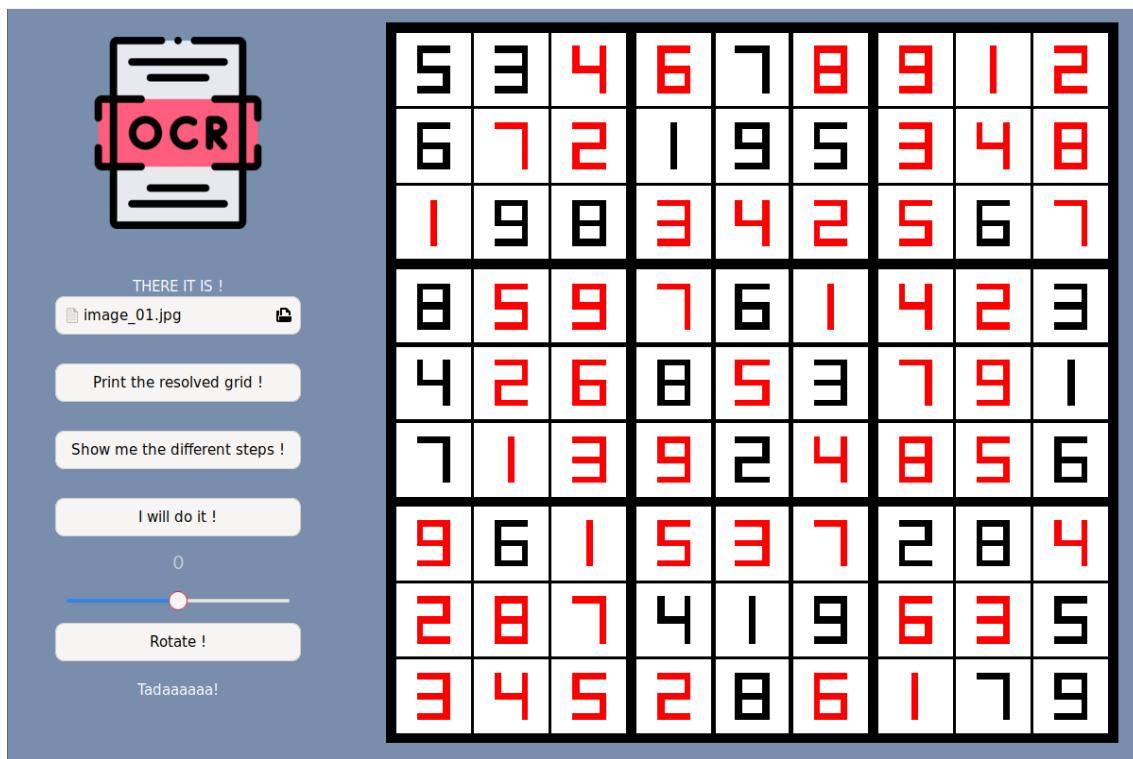


Figure 4.21: Interface finale

Modification de la grille détectée

Le réseau de neurones présente des résultats satisfaisant néanmoins il est à même de faire quelques erreurs et donc engendrer une résolution impossible de la grille.

Ainsi l'utilisateur peut entrer les coordonnées de la grille là où il désire modifier un chiffre s'il a mal été repéré.

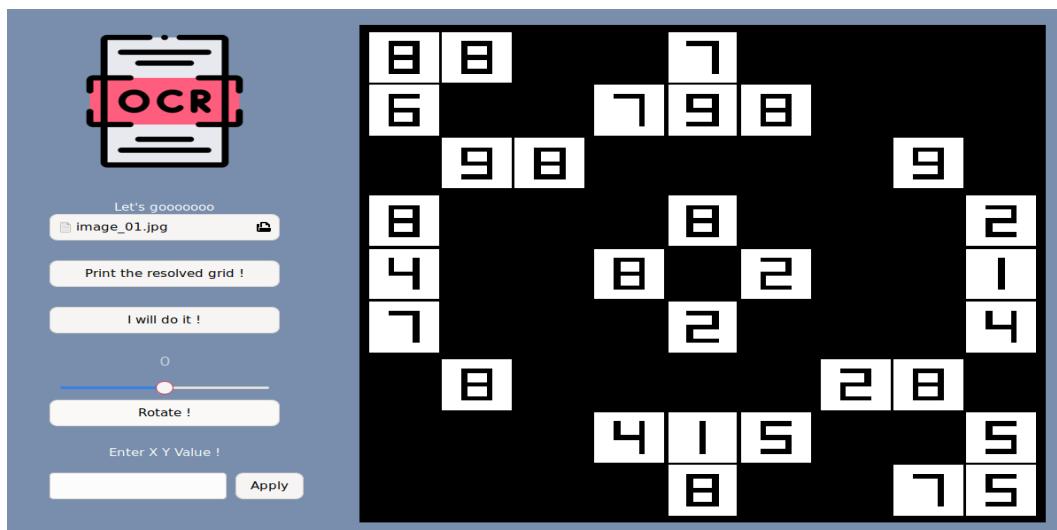


Figure 4.22: Interface de modification

4.6 Reconstruction de la grille

En ce qui concerne la reconstitution de la grille, nous avons décidé de prendre une nouvelle grille entièrement vierge pour plus de lisibilité. Nous utilisons ensuite deux fichiers organisés de la même manière, le premier provient de l'IA, le second provient du Solver et nous sauvegardons les différents caractères correspondant aux différentes cases du sudoku.

Par la suite nous venons comparer les caractères des deux fichiers, si ceux-ci s'avèrent être les mêmes, le caractère correspondant est copié de couleur noire sur la grille à la case correspondante. Dans le cas où les caractères ne sont pas les mêmes, le caractère provenant du fichier Solver est copié de couleur rouge sur la grille. Afin de copier les différents chiffres sur la grille, nous avons créer une banque de dix-huit images de taille 28x28 comportant les chiffres de 1 à 9 en noir et en rouge.

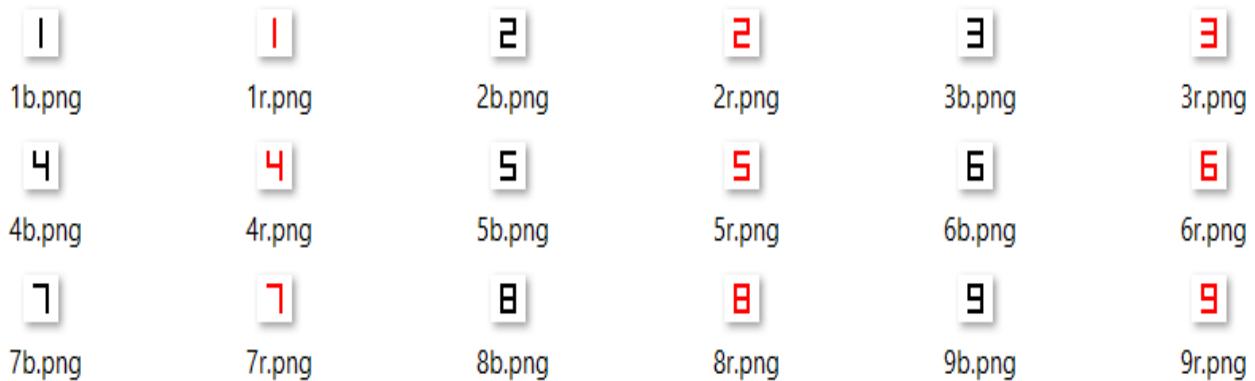


Figure 4.23: Banque d'images

Une fois la bonne image choisie, nous la copions pixel par pixel sur la nouvelle grille afin de reconstituer le sudoku résolu.

Sauvegarde du résultat

Une fois la grille traitée, les caractères reconnus par l'IA, le sudoku résolu et reconstitué, nous affichons l'image finale sur l'interface et sauvegardons l'image en .png afin qu'elle puisse être affichée dans une invite de commande ou utilisée d'autres façons.

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

Figure 4.24: Reconstitution du Sudoku 01 résolu

4.7 Site internet

Afin de mieux expliquer ce que notre projet fait, nous avons décidé de faire un site internet ! Pour cela, nous avons utilisé des langages basiques en web, le HyperText Markup Language (HTML), le Cascading Style Sheets (CSS) et le JavaScript.

La page principale est plutôt simple étant donné que nous n'avons pas eu beaucoup de temps pour le faire (nous avons eu l'idée d'en faire un au dernier moment). Nous sommes donc partis sur des couleurs simples avec un design plutôt sobre. La page principale se présente comme ceci:

Chaque rubrique présente dans la catégorie dans "comment fonctionne notre OCR" représente ce que nous allons vous expliquer pendant la soutenance.

Enfin pour afficher du code directement sur le site, nous avons utilisé Highlightjs qui permet de faire des rendus de code informatique plus lisible sur les sites internet.



Qu'est ce qu'un OCR ?

Un ocr correspond à la reconnaissance automatisée de textes imprimés ainsi qu'à la retranscription de ces textes écrits en fichiers numériques. De fait, un ocr permet à un appareil numérique de "lire" le contenu d'un document papier et de le restituer informatiquement. Dans notre cas, l'objectif de notre ocr est de scanner un sudoku imprimé, de numériser la grille ainsi que les chiffres déjà inscrits. Par la suite, notre logiciel devra être capable de résoudre ce sudoku pré-rempli pour enfin le restituer dans sa forme complétée via une interface utilisateur ergonomique et intuitive.

Comment fonctionne notre OCR ?



Qui sommes-nous ?

Nous sommes une petite équipe de 4 actuellement en SPE Info à EPITA. Nous sommes tous très intéressés par l'informatique sous tous ces aspects. L'équipe est composée de:



Alexandra
Wolter

Joueuse de Wow et seule fille du groupe, j'ai réalisé l'intégralité de l'interface graphique du projet. C'était une grande première pour moi et j'ai beaucoup aimé réaliser cette tâche. J'ai également implémenté la reconstitution de la grille mais aussi d'autres tâches qui m'ont beaucoup appris.



Guillaume
Jolivalt

Curieux du groupe, je me suis attelé à des tâches dans différents domaines telles que la résolution du sudoku, la réduction de bruits ou encore la recherche d'information sur le réseau de neurones. J'ai beaucoup apprécié travailler sur ces diverses parties avec ce groupe.

```
struct Squares find_square(struct Lines lines, int max_size){  
    int a = 0;  
}
```

CHAPTER 5

LIMITES ET AMÉLIORATIONS POSSIBLES

Les limites

Lors de l'aboutissement de ce projet, nous avons manqué de temps pourachever tous nos objectifs ce qui fait que notre Sudoku Ocr ne répond pas à toutes les demandes et présente des limites.

Notre traitement d'image n'est pas en capacité de préparer toutes les images. Celles présentant un fort taux de bruits, ou des perspectives difficiles nous sont impossible à traiter.

En terme de réduction de bruit, certains filtres que nous avions implémentés ne font pas l'effet voulu. En effet, ils font l'inverse, c'est à dire qu'ils éliminent certaines lignes ou colonnes réduisant ainsi notre faculté à détecter la grille et les cases, c'est le cas du filtre médian pour certaines images. De plus, ils viennent parfois couper les chiffres les rendant notamment impossible à être traité correctement par le réseau de neurones.

De ce fait, le réseau de neurones n'est pas capable d'avoir un fort taux de réussite à la découpe de certaines grilles complexes car ces images de 28x28 diffèrent de façon conséquente aux images utilisés lors de l'entraînement.

Dans tous les cas, le réseau de neurones ne permet pas d'assurer la détection de tous les chiffres correspondant. De ce fait, nous avons implémenté une interface de correction avec laquelle l'utilisateur peut corriger les maigres erreurs effectuées par le réseau.

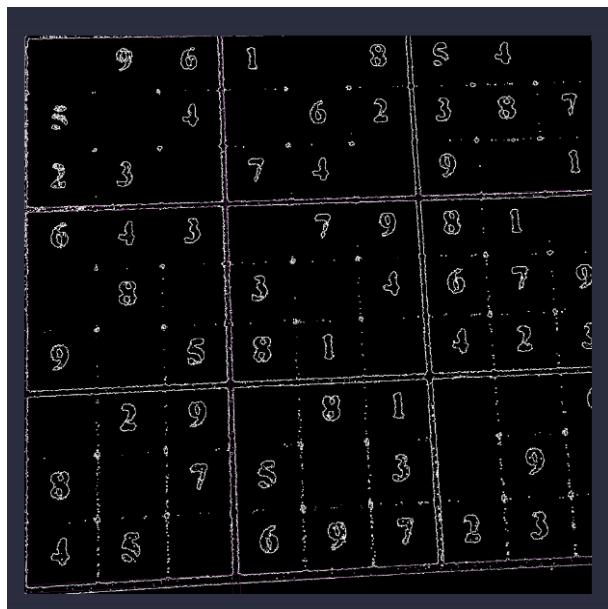
Les améliorations possibles

Bien que nous soyons très fiers du produit fini de notre projet, il est indispensable d'admettre que différentes améliorations pourraient encore y être apportées afin de le perfectionner.

En terme de bonus et de continuité de développement, Guillaume et Erwann ont préparé l'héxadoku solver permettant de résoudre des sudokus de 16 par 16. Si nous avions eu plus de temps, nous avions pensé à inclure cela dans le projet et donc retravailler la totalité des parties pour permettre d'inclure la possibilité de travailler sur des héxadokus.

Pour ce qui est du prétraitement de l'image, la partie mériterait encore d'être améliorée. En effet, nous pourrions ajouter de nouveaux filtres afin d'éliminer plus de bruits sur les images. Nous devrions aussi refaire tout notre système de séparation car si l'image est inclinée en profondeur (comme l'image 6), le système ne marche pas (preview en dessous)

L'idée serait donc de travailler sur des parallélogrammes au lieu de travailler sur des carrés.



Faire cela impliquerait donc de changer tout le système de séparation des cases et de la détection de la grille.

CHAPTER 6

RESSENTIS DU PROJET

Erwann Lesech

Comme je l'avais auparavant écrit dans le rapport de soutenance intermédiaire, j'étais et je suis toujours transandé par ce projet. Cela aura été la première occasion pour moi de me lancer dans le machine learning et le deep learning à travers l'implémentation de mon premier réseau de neurones sur la porte XOR avant novembre. Entre temps, la reconnaissance complète de caractère ne m'a pas parru aisé, bien au contraire. Il m'a fallu de nombreuses recherches et une bonne quantité de pérsverance pour réussir cette partie du projet. Durant une longue période je me suis senti un peu laisser à l'abandon que ce soit l'absence de TP sur le sujet lors de l'année ou encore l'absence de cours magistral sur ce sujet si complèxe qu'est le machine learning. Néanmoins, ce super groupe a été très compréhensif concernant mes difficultés et m'a soutenue jusqu'à la fin, ce qui m'a permis de pouvoir présenter un résultat satisfaisant lors du rendue de fin de projet.

Dans le but d'élargir ma pensé à la totalité du sujet, je suis convaincu que cela nous a à tous apporté de nombreuses nouvelles compétences et surtout une nouvelle manière d'aborder le travail en équipe. Contrairement au projet S2 de l'année dernière, le spectre de domaine de travail est beaucoup plus limité nous obligeant à travailler ensemble au sein d'une même partie et de fait, cela nous demande d'être davantage rigoureux quand à la gestion du git, des commits, et de notre planning d'avancement afin de ne pénaliser personne à cause d'un retard d'un des membre du groupe.

Guillaume Jolivalt

Dans cette seconde partie de projet je me suis occupé d'une tâche totalement différente de la première partie et cela a été très instructif pour moi. En effet, j'ai pu me pencher sur les différents filtres utilisés pour la réduction de bruits et la binarisation. La phase de recherche pour déterminer quel filtre était le plus intéressant à implémenter et sur le fonctionnement des différents filtres a été très constructive et m'a beaucoup plu. De plus, c'était un domaine que je ne connaissait absolument pas et j'ai pu en apprendre plus sur celui-ci. J'ai beaucoup aimé travailler sur ce projet dans la bonne ambiance avec ce groupe soudé.

Ce projet m'a permis de découvrir beaucoup de choses qui m'étaient dès lors totalement inconnues. Il a donc été très constructif et passionnant malgré certaines ambuches.

Alexandra Wolter

J'avais beaucoup aimé travailler sur la première partie du projet mais cette seconde partie m'a davantage plu. Les différentes tâches que j'ai effectuées lors de cette fin de projet m'ont beaucoup appris et m'ont permis de m'améliorer sur différents domaines. J'ai beaucoup aimé travailler seule, mes tâches avançaient à un très bon rythme durant tout le semestre pour permettre au groupe de rendre un projet de qualité et fini. J'ai également adoré aider mes camarades et les motiver tout au long du projet, nous avons su travailler tous ensemble et une bonne ambiance régnait au sein du groupe.

Ce projet a été très enrichissant pour chacun d'entre nous et j'ai hâte de travailler sur un prochain projet en espérant qu'il m'apprenne autant que celui-ci.

Valentin Gibbe

Pour ma part je pensais avoir quasiment fini mes parties lors de la première soutenance. Je me suis rendu compte que les algorithmes que j'avais implementés étaient limités dès que nous passions sur des images plus complexes. Je trouve ce projet compliqué puisqu'il faut parfois essayer différentes façons de faire les choses dans le but de faire fonctionner le maximum d'images. Parfois nous arrivons à faire fonctionner une image en changeant quelques valeurs dans nos algorithmes mais une autre image qui fonctionnait précédemment ne fonctionne plus... Cependant, je suis très fier du résultat de notre groupe pour ce projet a été très formateur pour moi. J'ai pu découvrir plein de filtres que je ne connaissais pas. Ce projet fut très enrichissant pour chacun d'entre nous, je n'en doute pas!

Nous avons réussi ce qui relève d'une difficulté importante, coder un OCR fonctionnel en partant de 0 sur un nouveau langage sans librairie d'aide comme OpenCV. je suis très fier de notre groupe.

Avis général

C'est une page de notre scolarité à EPITA qui se tourne après avoir fini ce projet. C'est quasiment la fin de notre prépa et ce projet nous a donné de bons outils et de bonnes connaissances pour commencer l'année d'ING1. Tous les problèmes que nous avons sus ou non résoudre durant toute la période de l'OCR et la bonne cohésion de notre groupe nous a permis de progresser!

Le projet OCR s'est bien passé pour l'ensemble du groupe et nous nous sommes rendus compte que nous avons réussi à travailler tous ensemble. Nous sommes fiers de ce que nous avons réussi à accomplir et attendons avec impatience l'arrivée des futurs projets de notre scolarité !

CHAPTER 7

CONCLUSION

Pour conclure ce projet, nous sommes très fiers de ce que nous avons réussi à produire durant ce semestre. Le projet OCR est un énorme challenge et nous avons réussi à tenir le coup sans abandonner (même dans les moments difficiles).

Nous sommes satisfait de la façon avec laquelle notre groupe a collaboré durant ce semestre pour présenter ce produit fini et qualitatif. Notre gestion des outils tels que git s'en sont vu nettement amélioré et surtout cela nous a permis de perfectionner grandement notre maîtrise du C. En effet, nous avons utilisé des notions telles que les structures, les pointeurs et la mémoire allouée dynamiquement avant même que les TP traitant du sujet ne soient disponibles.

Nous avons réussi à finir le projet dans les temps et à rendre un logiciel fonctionnel malgré ces limites présentées au dessus.

Il serait intéressant que dans quelques mois/années nous nous repenchions sur ce projet pour aller plus loin encore. Parvenir à améliorer le traitement de l'image en réussissant à faire marcher le programme sur des images plus complexes, de moins bonnes qualités, obtenir un réseau de neurones avec un taux de réussite encore plus grand...

Comme disait Mark Twain, "L'amélioration continue vaut mieux qu'envisager la perfection"

LIST OF FIGURES

1.1	Sudoku imprimé	2
4.1	Principe du filtre médian	9
4.2	Exemples de découpes de l'image	14
4.3	Exemples de découpe après le nettoyage	14
4.4	Fichiers résultant de la découpe	15
4.5	Affichage et sauvegarde de la grille résolue	15
4.6	Résolution de l'hexadoku	16
4.7	Tableau GD Courbe de la fonction sigmoid	17
4.8	Tableau GD avec learning rate trop grand	18
4.9	Tableau XOR	18
4.10	Première itération	19
4.11	Dernière itération	19
4.12	weights file	20
4.13	Exemple d'architecture viable	21
4.14	Tableau GD Courbe de la fonction Relu	21
4.15	Base de données de caractères (dossier 6)	22
4.16	Résultat des prédictions	23
4.17	Base de données MNIST	23
4.18	Résultat des prédictions (avec MNIST)	23
4.19	Différents éléments de l'interface	24
4.20	Interface réalisée sur Glade	25
4.21	Interface finale	26
4.22	Interface de modification	26
4.23	Banque d'images	27
4.24	Reconstitution du Sudoku 01 résolu	28

LIST OF TABLES

2.1	Tableau de la répartition des tâches, R = Responsable & S = Suppléant . . .	4
3.1	Estimation des avancées des différentes tâches	5
3.2	Respect des prévisions de l'avancement du projet	6

ACRONYMS

CSS Cascading Style Sheets 28

GD Gradient Descent (descente de gradient 17, 18, 21, 35

HTML HyperText Markup Language 28

OCR Optical Character Recognition 2

RGB Red,Green,Blue 7, 9, 14

RN Réseau de neurones 16, 22, 23

ROC Reconnaissance Optique des Caractères 2

UI Interface utilisateur 24

XOR Porte ou-exclusive 18, 19, 35

SOURCES

Réseau de neurones

Machine learnia
SamsonZhang
3blue1brown
Deep learning

Hough Transform

Wikipedia
HIPR

Sobel filter

Wikipedia
Sobel filter en Python

Otsu filter

Binarisation
Otsu
Wikipédia

Rotation manuelle de l'image

Interpolation Bilinéaire
Interpolation Bilinéaire avec rotation

Découpage de l'image

WikiSDL

Dataset pour réseau de neurones

MNIST
numerical data

Site internet

HighlightJS
Doc HTML/CSS

Interface

GTK Lib