

Rapport de Projet

OCR Sudoku

20 Decembre 2022

Polverelli Thomas - Boismartel Léo - Nunes-De-Barros Leïla - Pinto Mathias



1 Introduction

Ce rapport a pour but de présenter l'avancer de notre travail sur le projet OCR Sudoku Solver. Dès le début nous nous sommes reparti les taches nécessaires pour la première soutenance et nous avons établi un plan sur l'organisation du projet. Nous nous sommes avant tout concentrés sur les taches obliagatoires mais nous avons aussi réfléchis sur celle facultative. Vous trouverez donc un résumé taches par taches de ce que chacun de nous a fait.

Sommaire

1	Introduction	2
2	Présentations personnelles	4
3	Avancement du projet	6
3.1	Répartition des charges	6
3.2	Avancement général	7
4	Aspects techniques	8
4.1	Traitement de l'image	8
4.1.1	Suppression des couleurs	8
4.1.2	Flou d'image	9
4.1.3	Méthode d'Otsu	9
4.1.4	Inversion des couleurs	10
4.1.5	Filtre de Sobel	11
4.2	Detection de la grille et de la position des cases	12
4.2.1	Enregistrement sous format image de la grille et de la position des cases	13
4.3	Rotation de l'image	15
4.3.1	Rotation manuelle de l'image	15
4.3.2	Detection de l'angle	17
4.4	Implémentation de l'algorithme de résolution d'un sudoku	18
4.5	Réseaux de Neurones OU EXCLUSIF	20
4.6	Réseaux de Neurones	23
4.7	Reconstruction de la grille	27
4.8	Interface Graphique	29
5	Annexe	35
6	Conclusion	36

2 Présentations personnelles

PINTO Mathias

Pour présenter mon parcours, je viens d'un baccalauréat avec des spécialités scientifiques : les mathématiques ainsi que la physique chimie. Mes passions sont le sport comme le football ainsi que les jeux vidéos. Je me suis tournée vers l'Epita avec une ambition de vouloir créer des jeux vidéo. Cependant ayant fait une première année en tant que SUP a l'epita et ayant effectué un projet de création de jeu vidéo, cet intérêt pour la création des jeux vidéo s'est estompé. En outre, elle a relevé de tout autre nouvel aspect sur l'informatique que je souhaiterai découvrir d'avantages.

NUNES DE BARROS Leïla

Je m'appelle Leïla, j'ai 18 ans et j'ai obtenu un baccalauréat général avec spécialités Mathématiques, Physique-Chimie. J'ai intégré l'EPITA évidemment car je m'intéresse à la programmation et plus particulièrement aux jeux vidéos, mais aussi car j'ai apprécié la méthode d'apprentissage et l'aspect international et professionnel que pouvait offrir l'école. J'ai toujours été curieuse à propos de comment fonctionnaient les machines, les appareils ou les programmes informatiques c'est pourquoi je suis sûre que ce projet sera enrichissant pour moi.

POLVERELLI Thomas

J'ai toujours aimé l'informatique et je savais depuis petit que je voudrais travailler dans ce domaine. Ce projet est totalement différent de celui de l'année dernière en un sens ou il requiert de nombreuses connaissances dans le domaine du Machine Learning pour la gestion des réseaux de neurone. Je suis certain que ce projet va m'apporter beaucoup en termes de connaissance mais aussi en termes d'expérience pour les projets futurs.

BOISMARTEL Léo

Projet intéressant qui nous pousse vers l'apprentissage autodidacte et la lecture de manuel pouvant paraître long et difficile à comprendre mais qui finalement se laisse apprécier. Ce projet permet de comprendre en quelques mesures un exemple de réseau de neurones.

L'année dernière nous avons fait un TP de programmation sur un sudoku donc je trouve ça intéressant d'avoir une continuation dans ce que nous avons fait afin d'aller plus loin.

3 Avancement du projet

3.1 Répartition des charges

Tâche	Mathias	Leila	Thomas	Léo
Suppression des couleurs	X			X
Prétraitement	X			
Détection des case et de la grille	X			
Enregistrement sous format image des traitements	X	X		
Récupération des chiffres présents dans les cases	X			
Detection d'angle et rotation		X		
La reconstruction de la grille	X			
Implémentation de l'algorithme de résolution d'un sudoku			X	X
Réseaux de Neurones			X	X
Sauvegarde du resultat	X			
Affichage de la grille		X		
Interface graphique		X		

3.2 Avancement général

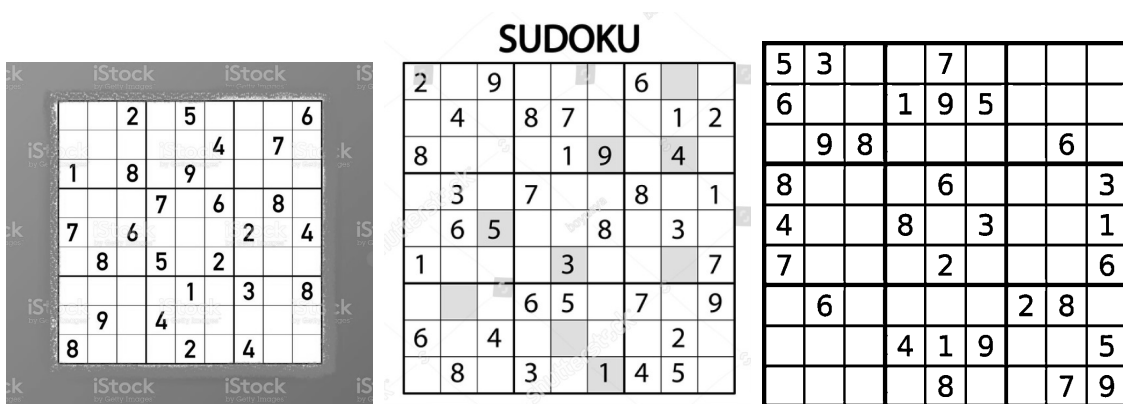
Tâche	Avancement Prévu	Avancement Réel
Suppression des couleurs	100%	100%
Prétraitement	100%	100%
Détection des case et de la grille	100%	100%
Enregistrement sous format image des traitements	100%	100%
Récupération des chiffres présents dans les cases	100%	100%
Detection d'angle et rotation	100%	100%
Implémentation de l'algorithme de résolution d'un sudoku	100%	100%
Reconstruction de la grille	100%	100%
Réseaux de Neurones	95%	86%
Affichage de la grille résolue	100%	100%
Sauvegarde de la grille résolue	100%	100%
Interface graphique	100%	90%

4 Aspects techniques

4.1 Traitement de l'image

4.1.1 Suppression des couleurs

Tout d'abord Mathias a effectué un GrayScale qui permet de retirer les couleurs qui diffèrent du gris et du blanc d'une image. Ainsi l'image aura uniquement des pixels de couleurs blancs et gris, cette méthode est importante car elle permet de faciliter l'étape suivante car en effet, par la suite il a mis en place une méthode de conversion d'image "Black and White" qui permet de remplacer l'image grise et blanche en une image composé que de pixel noir ou blanc.



4.1.2 Flou d'image

De plus, Mathias à l'aide d'une approximation des couleurs effectuée sur l'image en noir et blanc un flou d'image. Cette méthode est nécessaire car elle permet d'augmenter la qualité sur la représentation de la grille en noir et blanc sur différentes image très utile pour la méthode suivante.

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

4.1.3 Méthode d'Otsu

Mathias a implémenté la méthode d'Otsu qu'il a fini par rendre optionnelle car cette dernière utilise un seuillage global. Ainsi, elle n'est pas adaptée pour les images contenant du bruit. Des images que nous avons réussi à régler avec l'inversion de couleur ou l'on retrouve les même resultat . En effet car la méthode d'Otsu relativement intuitive définie la couleur noir ou blanche d'un pixel à partir d'un seuil obtenu par l'algorithme d'Otsu si la luminance est inférieur au seuil, le pixel sera blanc sinon il sera noir. Ensuite, on parcourt tous les seuils possibles et on effectue des calculs afin de déterminer une variance pour chaque classe d'intensité.

SUDOKU

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

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

4.1.4 Inversion des couleurs

Mathias effectue ensuite pour tous les pixels de l'image une inversion des couleurs en effet à l'aide d'une fonction `GetValueOfPixel` qui nous permet d'obtenir la valeur du Pixel traité une fonction inverse s'occupe d'inverser la couleur de ce dernier ainsi s'il possède une couleur noir sa couleur deviendra blanche vis versa. Cet algorithme aurait pu être négligé dans le processus au vu de la méthode qui en suit "Le Filtre de Sobel". Cependant, nous avons vu une amélioration conséquente sur la représentation des images en ajoutant cette méthode.

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

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

(voir Annexe pour une meilleur représentation de la 2eme image)

4.1.5 Filtre de Sobel

Mathias a mis en place l'algorithme du filtre de Sobel qui est un algorithme permettant de détecter les contours dans une image. Ce filtre de Sobel calcule le gradient de l'intensité pour chaque pixel définie par cette formule :

$$G_{x,y} = \sqrt{G_x^2 + G_y^2}$$

Si le gradient est ensuite réinjecté dans le pixel. Pour calculer ce gradient on utilise deux matrices pour obtenir les valeurs de G_x et G_y sur une image A. (voir Annexe pour une meilleur représentation de la 1er image)

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

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

4.2 Detection de la grille et de la position des cases

Afin de détecter la grille et les différentes cases Mathias a mis en place un Hough Transform qui est un algorithme qui s'effectue sur une image précédemment traité par le Filtre de Sobel et qui détecte les différentes grilles en prenant en compte que la plus grande sera la grille général du Sudoku. Une gestion d'angle ainsi que d'intersection paramétrique est mise en place dans cet algorithme afin de trouver l'ensemble du sudoku. De plus, elle trace et sauvegarde sur une image les intersections ainsi que le contour de chaque.

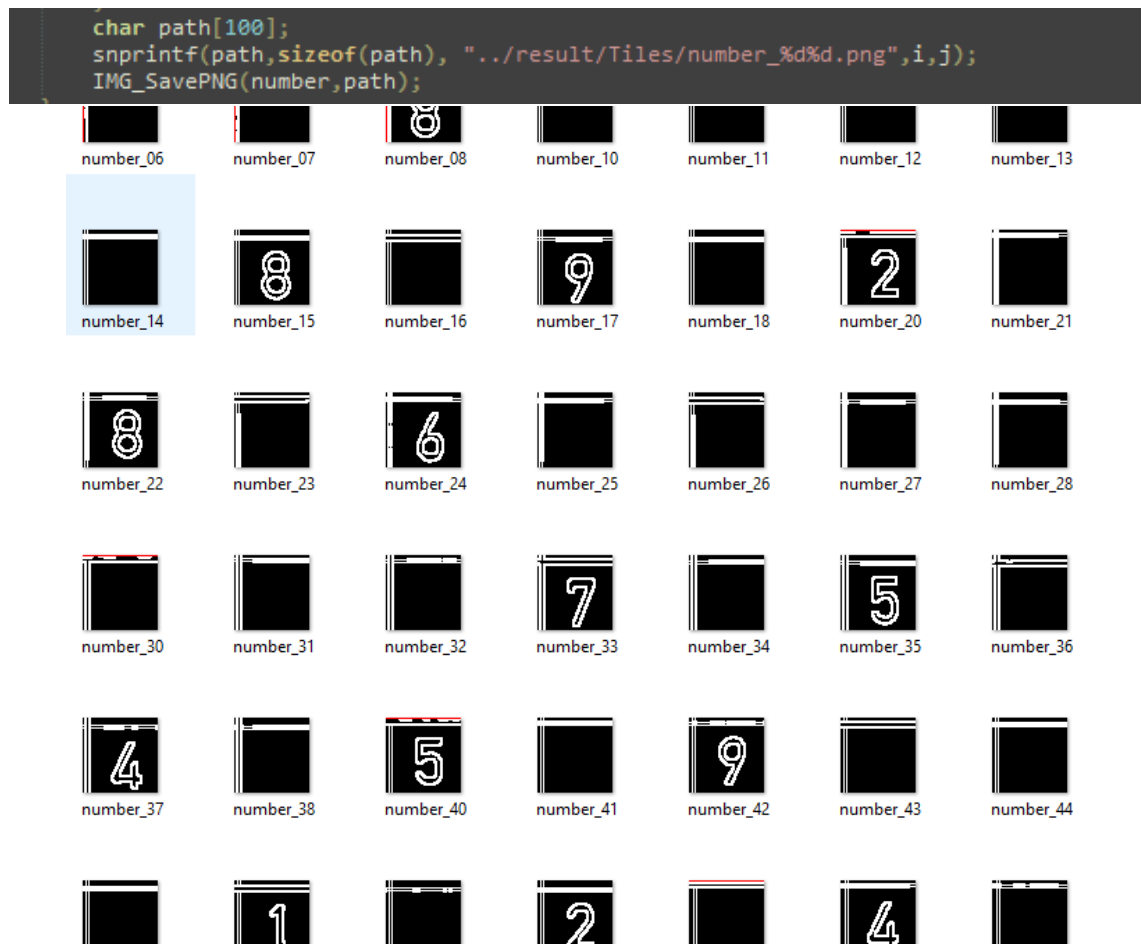
5	3			7			
6			1	9	5		
	9	8					6
8				6			3
4			8		3		1
7				2			6
	6					2	8
			4	1	9		5
				8			7

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

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

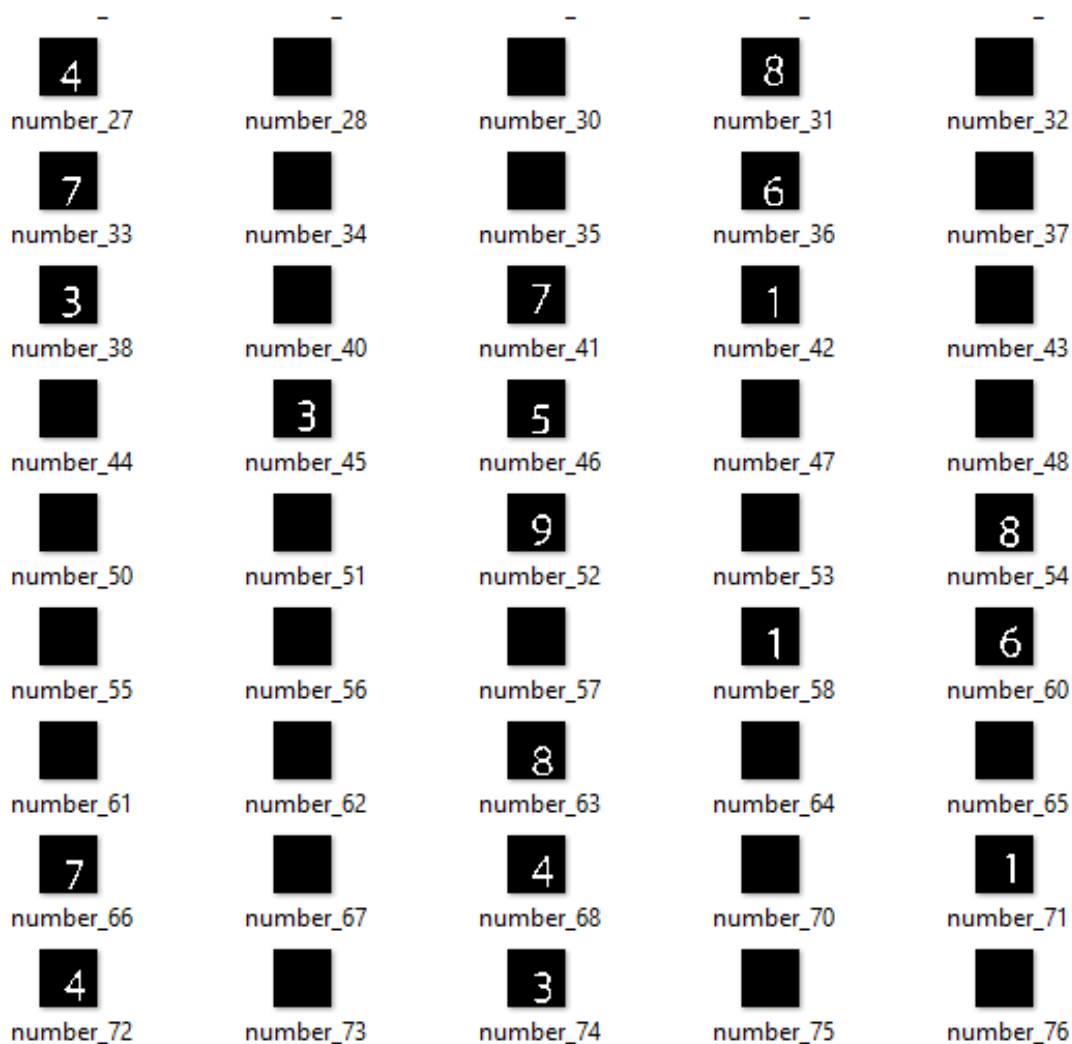
4.2.1 Enregistrement sous format image de la grille et de la position des cases

Dans l'algorithme Hough Transform, Dès qu'une case est détectée une image se crée ayant pour dimension l'angle supérieur gauche jusqu'à l'angle inférieur droit de se faire à l'aide de deux boucles, de la méthode "snprintf" ainsi que de la méthode IMG_SavePNG(number,path); Ainsi un dossier contient l'image de chaque case ayant de plus les bonnes coordonnées sur une matrice.



Par la suite Mathias a fait une fonction "resize" ainsi que "zoom".

Ces fonctions enregistrent uniquement le chiffre de chacune des cases sous un nouveau format d'image du 28x28 afin de faciliter la compréhension du réseau de neurone. De plus elle enlève les contours blanc de la grille qui pouvait porté a confusion lors de la détection du réseau de Neurone.



4.3 Rotation de l'image

4.3.1 Rotation manuelle de l'image

Pour effectuer la rotation j'ai tout d'abord commencé par faire mes recherches sur internet afin de savoir comment procéder et par quoi commencer. J'ai cherché plusieurs manières et écrit différents programmes qui n'ont pas fonctionné jusqu'à ce que j'apprenne que l'on pouvait faire la rotation grâce à la librairie SDL que nous avions déjà manipulée. J'ai donc repris mon TP6 et modifié quelques fonctions.

J'étais partie dans l'optique de faire une fonction qui permet de faire la rotation manuellement: lorsqu'on appuie sur la flèche droite, l'image fait une rotation à droite et inversement à gauche. Cependant, lorsque j'ai relu le cahier des charges, je me suis rendue compte que ce n'était pas ce qui était demandé. J'ai donc refais mes fonctions et modifier le main afin de faire la rotation à partir d'un angle donné par l'utilisateur.

Pour cela j'ai commencé par créer une fonction `ReadPixel()` qui renvoie la valeur d'un pixel à l'aide d'une surface, d'un x et d'un y. Ensuite j'ai écrit une fonction `WritePixel()` qui permet de modifier la valeur d'un pixel. Enfin j'ai fait ma fonction principale `SDLRotation()` qui effectue la rotation de l'image à partir d'un angle et qui ajuste la taille de la surface. Pour cela j'ai commencé par déterminer l'angle donné en radian afin d'utiliser `cos` et `sin` pour calculer la taille de la surface de la nouvelle image. Je crée ensuite une nouvelle surface qui contiendra l'image tournée. Je parcours donc chaque pixel de ma nouvelle surface afin de récolter leur donnée (`ReadPixel`) et de créer deux nouveaux x y qui correspondent au pixel tourné puis les remplacer dans le pixel de la surface (`WritePixel`).

Le main reprend le schéma de grayscale du TP6 avec quelques modifications. En effet, il y a plus deux surfaces et l'ajout de `printf`

et scanf afin que l'utilisateur puisse entrer ses valeurs (angle, rotation droite ou gauche...).

Le programme se lance avec une image en png. Lorsque le programme est lancé, un angle est demandé à l'utilisateur:

```
chose the angle of rotation
35
```

Puis la direction de la rotation:

```
Write 1 for left
Write 2 for right
1
```

Et enfin l'utilisateur à le choix de refaire une rotation ou non:

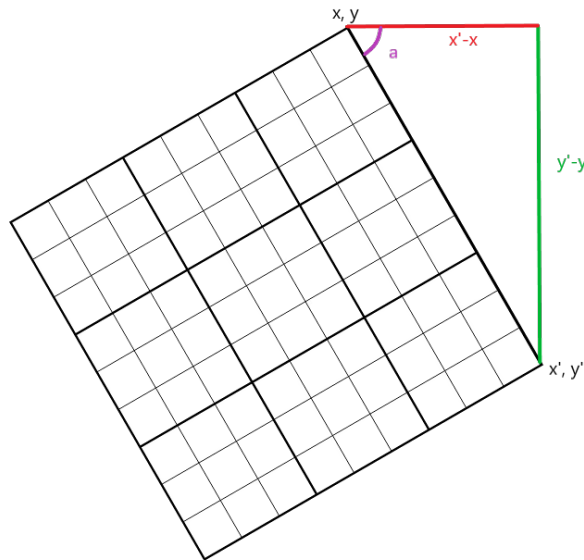
```
Do you want to rotate again?
Write 1 for yes
Write 2 for no
2
```

L'image est ensuite sauvegardé sous le nom de rotated.png.

4.3.2 Detection de l'angle

Afin de transformer cette rotation manuelle en rotation automatique il était nécessaire de détecter l'angle de rotation à appliquer. Pour cela, il suffisait de parcourir l'image afin de trouver le coin haut de la grille et stocker ses valeurs (x,y) pour ensuite trouver le coin gauche ou droit et de même stocker ses valeurs (x',y') .

Une fois ces deux coordonnées trouvées, il suffisait de calculer les longueurs $x'-x$ et $y'-y$ avec d'avoir les côtés d'un triangle et enfin trouver l'angle a à l'aide de la formule: $\tan(a)=(y'-y)/(x'-x)$ (Voir schéma ci-dessous). On obtient donc $a=\arctan((y'-y)/(x'-x))$



Cet angle est directement appliqué à la rotation manuelle: au lieu de demander à l'utilisateur la rotation se fait directement avec la valeur de l'angle trouvé.

4.4 Implémentation de l'algorithme de résolution d'un sudoku

Pour la partie de résolution du sudoku je suis parti sur une implémentation sous forme de liste à deux dimensions, donc une qui contient les neuf lignes et dont chaque ligne contient 9 éléments. Puisqu'il nous fallait résoudre un sudoku à partir d'un fichier texte j'ai d'abord fait une fonction permettant d'afficher un sudoku dans la console pour tester les résultats. Pour ce faire je parcours le sudoku tout en affichant les éléments parcourus et je rajoute des espaces tous les 3 éléments pour la visibilité et un retour à la ligne toutes les 3 lignes.

Ensuite j'ai créé la fonction pour lire les sudokus dans un fichier texte. J'ai utilisé pour ça la commande `fopen(path, "r")`, "r" pour read. Le parcours que je fais s'arrête à "EOF" autrement dit End Of File. Pendant le parcours je place les caractères trouvés à leur place dans le sudoku passé en paramètre. A noter que les caractères sont convertis en entier dans mon programme.

J'ai donc un programme qui lit un sudoku et l'autre qui l'affiche. Maintenant il me faut le résoudre. Pour cela j'ai besoin de 3 fonctions supplémentaire une première qui trouve la première place vide dans le sudoku en lisant de gauche à droite et de haut en bas. La seconde fonction retourne un booléen qui correspond à si le sudoku est résolu ou non. La troisième retourne un booléen qui correspond à si le sudoku est valide ou non.

Trouver la place vide est une fonction simple à envisagé. Pour savoir si un sudoku est résolu il faut tester que toutes les lignes et toutes les colonnes ne contiennent qu'une fois un seul élément, idem pour les éléments tous les carrés du sudoku. Elle s'arrête automatiquement si elle trouve un point (le point désigne une case vide) car le sudoku ne peut donc pas être fini.

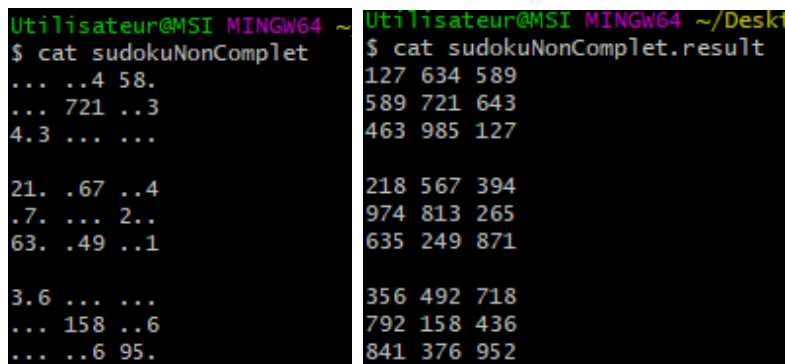
La fonction qui vérifie si le sudoku est valide utilise le même principe mais ne s'arrête pas si elle trouve un point.

J'ai implémenté une fonction qui m'aide pour savoir si les éléments sont unique grâce à un parcours dans une liste qui vérifie si l'élément est présent.

Maintenant la fonction qui résous le sudoku peut être écrite tout d'abord on vérifie si le sudoku est résolu si oui on s'arrête sinon on continue. Ensuite on trouve la place libre et on met la valeur 1 dedans. Si le sudoku est valide comme ça on appelle la fonction dans laquelle on est pour tester les prochaines cases. Si en suivant les prochaines cases le sudoku n'est plus valide alors on revient en arrière petit à petit en remplaçant les éléments que l'on a posé par des cases vide.

Une fois la fonction de résolution appelée on se retrouve avec un sudoku résolu que l'on peut afficher mais pas sauvegarder. Alors j'ai rajouté une fonction qui crée un fichier texte avec l'extension ".result" à la fin et le nom est le même que le fichier texte appelé au départ. Cette fonction lit élément par élément tout en ajoutant les éléments dans le fichier et en rajoutant des espaces tous les 3 élément et des retours à la ligne toutes les 3 lignes.

Ainsi le fichier solver.c est appelé avec un paramètre correspondant à un fichier texte contenant un sudoku et renvoie un sudoku résolu.



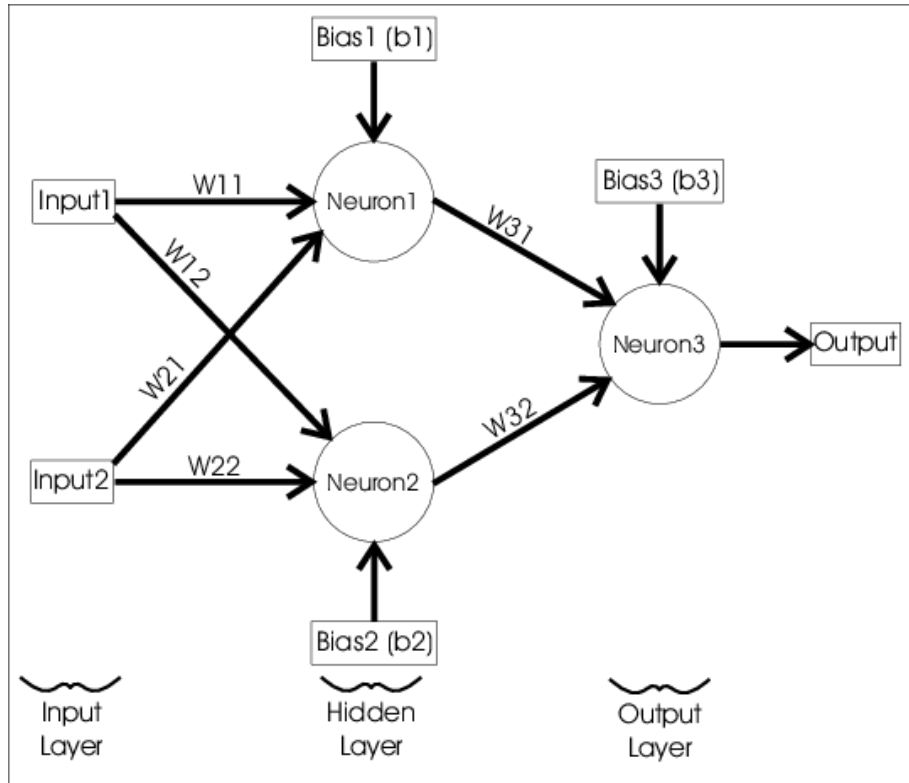
The image shows two side-by-side terminal windows. The left window displays the contents of a file named 'sudokuNonComplet', which contains a 9x9 grid of numbers and dots representing a partially solved Sudoku puzzle. The right window displays the contents of a file named 'sudokuNonComplet.result', which shows the same 9x9 grid but with all the dots replaced by numbers, representing the fully solved Sudoku puzzle.

```
Utilisateur@MSI MINGW64 ~  
$ cat sudokuNonComplet  
... ..4 58.  
... 721 ..3  
4.3 ... ..  
  
21. .67 ..4  
.7. ... 2..  
63. .49 ..1  
  
3.6 ... ..  
... 158 ..6  
... ..6 95.
```

```
Utilisateur@MSI MINGW64 ~/Desk  
$ cat sudokuNonComplet.result  
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
```

4.5 Réseaux de Neurones OU EXCLUSIF

Le réseau de neurones tient sur le principe d'apprentissage grâce à des poids. Dans le programme de porte XOR on utilisera un certain nombre de variable, d'abord les paramètres d'entrée donc 00, 01, 10, 11 et avec eux la sortie souhaitée 0, 1, 1, 0. Ensuite il faudra stocker les sorties récupérées sur la deuxième couche et neurones et la troisième également. En plus de leur sortie il faut retenir leurs poids retenus dans des listes. On voit donc que les poids $W1$ contiennent deux éléments, $W2$ et $W3$ également mais pour notre programme nous utiliserons une



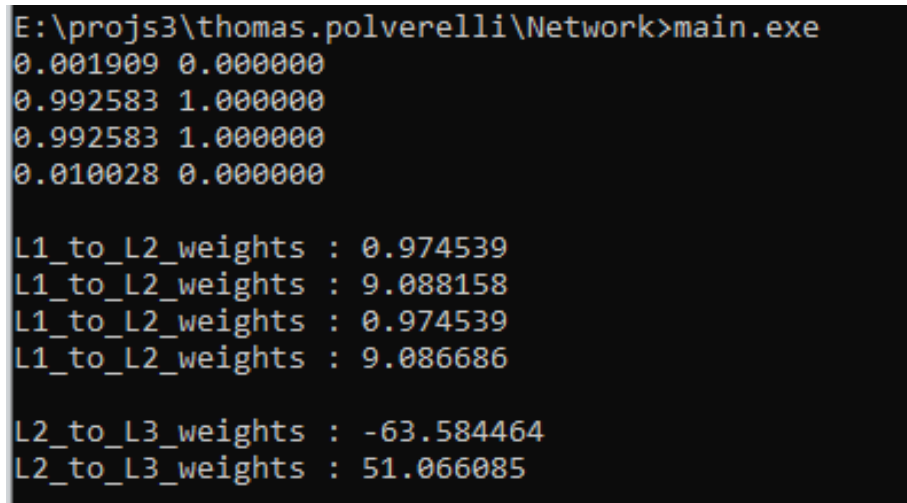
seule liste pour les poids de la première couche à la deuxième.

Pour avoir des résultats fiables nous utiliserons des poids aléatoires au départ afin de s'assurer qu'il fonctionne tout le temps. La fonction permettant de générer des nombres à virgule aléatoires est assez simple tout d'abord on initialise une "seed" en fonction de l'heure actuelle afin de changer tout le temps. Les autres fonctions utilisées sont la fonction sigmoïde qui permet d'avoir un résultat graduel et non seulement 1 ou 0. Ensuite nous utilisons des fonctions qui actualise les poids au fur et à mesure du programme. Ce changement correspond à faire la différence entre le poids recherché et la dérivé du poids multiplie par le taux d'apprentissage. Ainsi dans la fonction qui renvoie

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

un résultat pour chaque neurone l'on a affaire avec une multiplication de matrice. On multiplie le poids obtenu entre un neurones N1 et N2 par la valeur N1 et on obtient la nouvelle valeur N2. Pour tester les valeurs différentes j'utilise une fonction pour remettre à 0 les valeurs des couches 2 et 3 de neurones. L'algorithme de back-propagation est au cœur de l'apprentissage en profondeur dans les réseaux de neurones. L'objectif principal est de réduire la fonction de coût en trouvant des minima locaux basés sur les poids et les biais calculés. Pour trouver les minima locaux, on effectue des dérivations successives pour modifier les poids. L'algorithme de feed-forward est utilisé pour propagée les valeurs des neurones aux suivants. Et s'effectue par combinaison linéaire entre la valeur des neurones et leur poids associés vers les neurones de la couche suivante de la manière d'un produit de matrice.

Pour sauvegarder les poids j'utilise une fonction qui crée un fichier "XorWeights.txt" avec les poids de chaque neurone vers le prochain neurone. La fonction pour réutiliser des poids déjà existant recherche un fichier "XorWeights.txt" et utilise les poids dedans s'ils existent, grâce à la fonction fscanf de la librairie "stdio.h", sinon ils les génèrent aléatoirement.



```
E:\projs3\thomas.polverelli\Network>main.exe
0.001909 0.000000
0.992583 1.000000
0.992583 1.000000
0.010028 0.000000

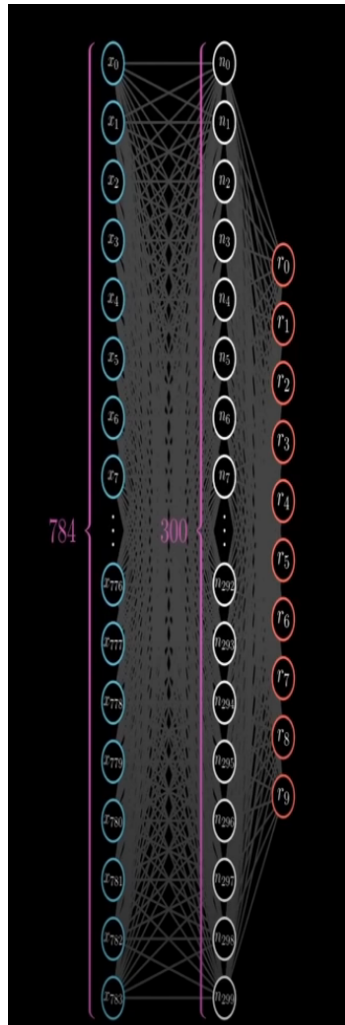
L1_to_L2_weights : 0.974539
L1_to_L2_weights : 9.088158
L1_to_L2_weights : 0.974539
L1_to_L2_weights : 9.086686

L2_to_L3_weights : -63.584464
L2_to_L3_weights : 51.066085
```

Figure 1: Résultats du réseaux de Neurone et valeurs des poids

4.6 Réseaux de Neurones

Pour la partie du réseau de neurones, nous avons commencé par réfléchir sur la structure en elle-même. En effet le OU EXCLUSIF ne possédant que 2 entrées il était difficile de faire rentrer une image entière. Notre réseau de neurones sera défini par une couche d'entrée, une couche "cachée" et une couche de sortie. De plus nous entraînerons le réseau grâce à des poids qui relie entré-cachées, caché-sortie.



Pour cela nous allons donc étendre la layer des entrées a 784 pour utiliser des images de 28x28 pixels qui seront préalablement découpé sur l'image d'origine. Nous avons donc créé notre propre structure de Matrice pour pouvoir représenter les layers et pouvoir faire des opérations entre elles tel que les multiplications entre matrice qui sont primordiales. Nous avons décidé de représenter les images sous la forme de fichier csv. Chaque ligne représente une image, la première donnée représente la valeur attendu (label) et les cases qui suivent sont les couleurs des pixels d'une image triée en Row-Majors :

label , 1x1 , 1x2 , 1x3 , 1x4 , 1x5 , ... , 28x28

Pour utiliser le format csv correctement nous avons donc implémenter plusieurs fonctions pour passer du csv aux SDL-surface. Le processus pour utiliser une image est tout d'abord de la convertir en csv puis à partir du csv on construit une liste d'entier représentant les couleurs des pixels, pour créer un objet de type struct Img lié à une matrice de pixels qui convient à notre utilisation dans ce réseau de neurone.

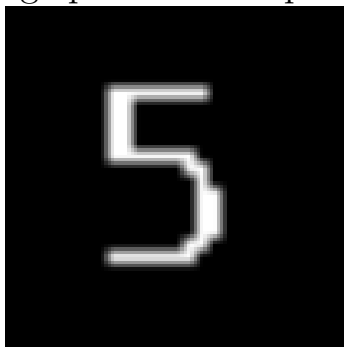
On utilisera ce format d'image pour entrainer nos neurones grâce aux images proposée par le mnist(Modified ou Mixed National Institute of Standards and Technology) qui contient 60000 images stockées sur 256 nuances de gris(rgb). Nous avons donc téléchargé au préalable les images d'entraînement. Pour ce qui est des images de test nous passerons les images récupérer par le découpage de la grille.

Pour maintenant entrainer notre réseau on calcule d'abord le passage à la layer suivante grâce aux poids actuels, en les multipliant avec les données des neurones, puis on applique la fonction sigmoïde pour obtenir des valeurs qui varie en fonction de la fonction exponentiel inverse. Donc plus la valeur est basse plus on observe du changement. Ensuite nous calculons l'erreur de sortie en soustrayant les matrices de sortie et des résultats finaux.

Pour les erreurs des neurones cachés on transpose nos poids de sortie, c'est à dire que l'on inverse sa matrice, et nous la multiplions à ses erreurs de sortie. Finalement on utilise donc la fonction de back-propagation pour calculer les poids cachés entre les différentes layers. Nous faisons la même chose pour la matrice de sortie pour obtenir une sortie. Pour tester notre réseau de neurone on a mis en place un test sur 300 images et avons fait la moyenne de ceux qui étaient correcte. Nous avons atteint 90% de réussite sur les images du MNIST.

Au final nous avons crée la fonctions NNpredict qui prends en paramètre un pointer de char qui représente le chemin et qui calcule, en premier lieu, la somme des couleurs des pixels pour vérifier que l'image n'est pas toute noire ici si la somme est inferieur a 2000 on estime que l'image est noire. Ensuite créer donc un fichier csv avec l'image et on charge le network avec les poids ayant été sauvegardé lors de l'entraînement. D'ici, on aplati l'image en un vecteurs colonne et on le multiplie avec les poids du network. Pour finir on recupère le nombre le plus probable dans le vecteur résultat.

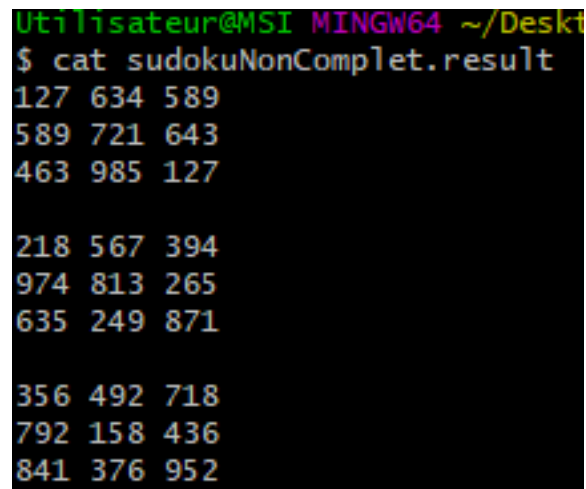
Figure 2: image prédite : 5 à partir de l'image



26

4.7 Reconstruction de la grille

Mathias s'est occupé de la reconstruction de la grille. Il a créé une fonction qui permet de lire un fichier que le solveur générerait en résolvant le Sudoku. Car en effet, le Solveur enregistre le Sudoku sous un format très précis

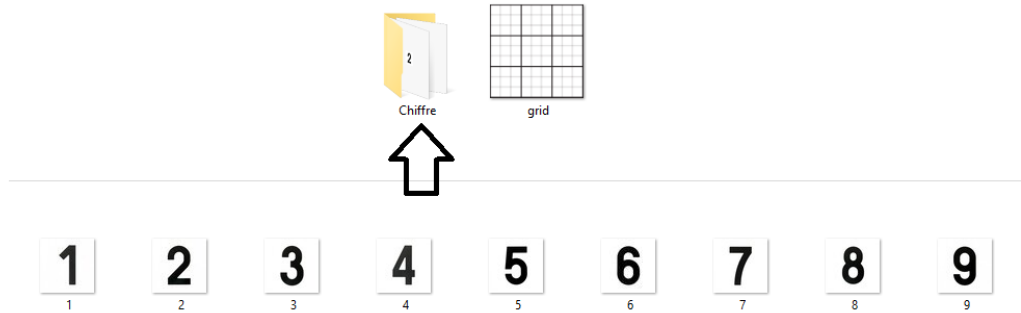


```
Utilisateur@MSI MINGW64 ~/Deskt
$ cat sudokuNonCompleet.result
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

356 492 718
792 158 436
841 376 952
```

Ce fichier est ensuite lu grâce à une fonction "read Sudoku" créée pour cet effet. Par la suite Mathias a implémenter une image d'une grille de sudoku en format png qui sera propre à chaque sudoku. De plus, il a implémenter un format de chaque chiffre qui le sudoku peut avoir (de 1 à 9)



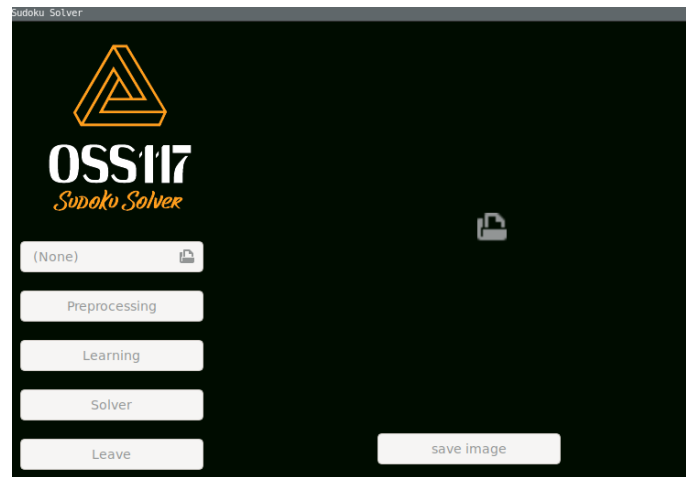
Ainsi, grâce à la fonction `Read sudoku`. Mathias a démarré une double boucle afin de copier grâce à une fonction `"getpixel"` et collé grâce à `"putpixel"` le bon chiffre sur la grille de sudoku. De plus, chaque image de chiffre ont la même taille 52x52 et correspond parfaitement aux dimensions de la grille.

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

4.8 Interface Graphique

Pour l'interface nous avons utilisé glade et gtk étant donné que nous avons déjà une certaine connaissance avec ces outils.

Voici à quoi elle ressemble:



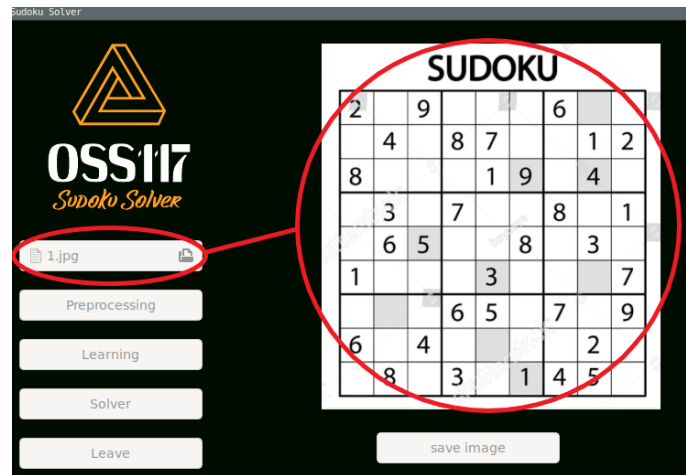
Pour cette interface j'ai commencé par créer une window directement sur l'interface dans laquelle j'ai inclus un Gtk Fixed afin de pouvoir introduire des boutons et des images.

Pour les boutons j'ai utilisé un gtkbox qui permet de couper une section de la window en plusieurs espaces afin d'insérer différents boutons dans chaque partie. J'ai récupéré chaque bouton dans mon fichier code afin de les connecter aux fonctions correspondantes.

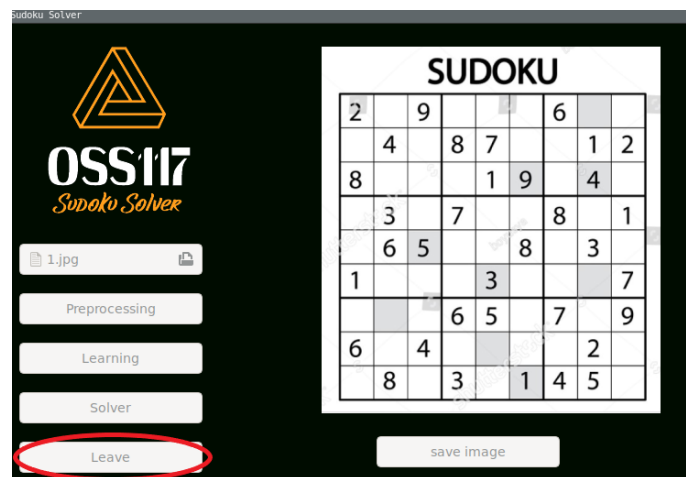
Trois boutons reliés aux fonctions ont été créés: un bouton "pre-processing" qui effectue le prétraitement, un bouton "solve" qui soulève la grille et le bouton learning qui permet de reconnaître des chiffres.



Un bouton qui permet de choisir un fichier (une grille de sudoku à résoudre) à été créé afin que l'utilisateur puisse choisir le fichier qu'il souhaite. Enfin, un bouton "Leave" pour quitter l'interface est

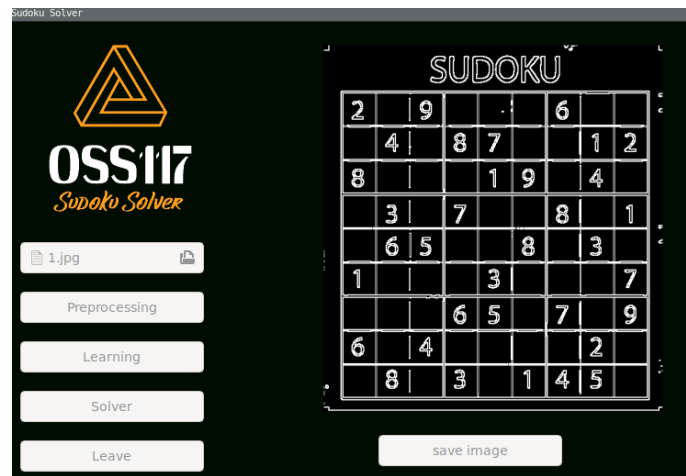


également présent.

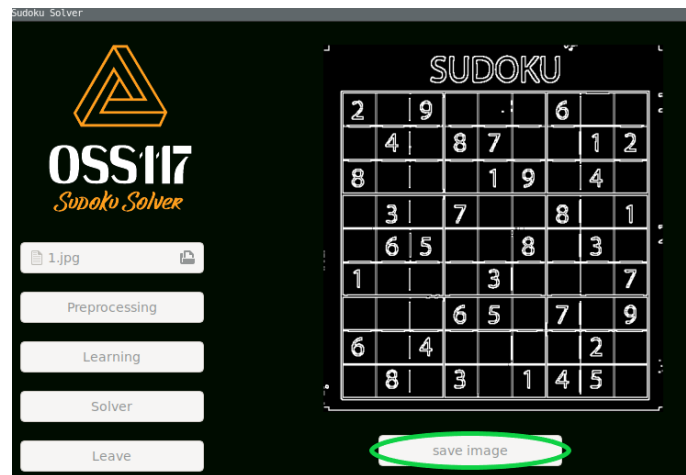


Afin de relier toutes fonctions codées jusqu'à maintenant dans l'interface il était primordial de réunir tous les Makefile en un seul, afin qu'il puisse compiler tous les programmes, mais aussi d'appeler les fonctions des autres dossiers dans l'interface.

J'ai ensuite affiché une image à l'aide d'un gtkimage que j'ai connecté à au bouton mentionné précédemment qui permet de choisir un fichier. Pour cela j'ai créé un pixbuf, à l'aide d'un Gtk FileChooser, dont j'ai récupéré le filename afin de l'inclure dans l'image à afficher sur l'interface. Cette image est modifiée après chaque étape afin que l'utilisateur puisse visualiser chaque étape nécessaire pour la résolution du sudoku.



Enfin, il a la possibilité de sauvegarder l'image à l'aide du bouton "save image" situé en dessous de l'image



Afin que l'interface ne reste pas neutre car le design d'une interface est la première chose que l'on remarque, j'ai donc essayé de faire quelque chose de simple sans pour autant qu'il soit banal.

Pour cela j'ai tout d'abord ajouté un logo créé sur paint à l'effigie de notre groupe "OSS117", cependant sur un fond blanc les couleurs du logo crée ne rendaient vraiment pas bien et l'interface semblait toujours trop neutre. C'est pour cette raison que j'ai changé la couleur du fond de la fenetre en noir ce qui a permit de faire ressortir la couleur orange du logo et de donner un coté original au design de l'interface.

5 Annexe

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

6 Conclusion

Pour conclure, nous avons fini tous les éléments obligatoires pour cette soutenance Finale et avons terminer complètement le projet. Le traitement a été fait et renvoi une image sans contour et découpe les images en ne laissant aucun bruit. La rotation automatique fonctionne bien avec les images tournées à moins de 90° . La résolution du sudoku fonctionne sans problèmes. Le XOR nous a permis de comprendre l'architecture à utiliser et comment entrainer et tester ses images. Le réseau de neurones marche sur la plupart des images données mais il reste des prédictions fausses. La reconstruction de la grille utilise des images prédéfinies à partir du fichier texte retourner une fois la fonction solve finie. L'interface graphique nous permet d'orchestrer le tout par sa facilité d'utilisation des différentes fonctions.