

Rapport de soutenance finale

rock - Projet S3



Cléry Pelvillain
Thybalt Marshutz
Antoine Mayol
Mathis Person



Info-Sup EPITA
Promo 2026 - Rennes

Contents

1	Introduction	4
2	Notre Groupe	4
3	Repartition des charges	4
4	État D'avancement du projet	5
5	Pré-traitement	6
5.1	Structures	6
5.1.1	Structure Pixel	7
5.1.2	Structure Image	8
5.2	Transformation en nuances de gris	9
5.3	Filtres intermédiaires	10
5.3.1	Inversion	10
5.3.2	Normalisation de la Luminosité	10
5.3.3	Réduction des contrastes	10
5.4	Lissage	11
5.5	Binarisation	13
5.5.1	Otsu	14
5.5.2	Seuillage	14
5.5.3	Déttection des lignes	15
5.6	Transmission des données	17
6	Detection de grille	17
6.0.1	Rotation de l'image	19
6.0.2	Decoupage de la grille	20
7	Réseau de neurones	21
7.1	Fonctionnement	21
7.2	Le XOR	23
7.3	Compréhension de chiffres	24
7.3.1	Vue d'ensemle des modifications	24
7.3.2	Type de données sauvegarde du réseau	25

7.3.3	Base de donnée	26
7.3.4	Utilisation du réseau	26
8	Resolution d'un sudoku	27
9	interface	28
9.1	Glade	28
9.2	GTK3+	29
9.3	Utilisation	30
9.3.1	SDL2	32
9.4	Interaction de l'utilisateur	34
9.5	Base de donnée	34
9.6	Déroulement final par étapes	35
10	Bibliographie	36
10.1	Outils	36
10.2	Sources	36

1 Introduction

Pour notre projet de S3 à l'EPITA nous devons créer un OCR (Optical Character Recognition) en 5 mois. Le but de cet OCR est de pouvoir analyser une grille de sudoku à partir d'une image. L'image doit passer par plusieurs étapes de traitements afin d'en récupérer les éléments importants et de résoudre le sudoku. Nous détaillerons ces différentes étapes et les aspects technique du projet à travers cette première soutenance.

2 Notre Groupe

Nous sommes Rock, un groupe de 4 étudiants de seconde année à l'EPITA. Etant tous intéressé par différents domaines de l'informatique, ce projet nous réuni afin d'aguiseur notre curiosité sur des facette de l'informatique auquelle nous ne portions pas d'intention.

Ce groupe est composé de Cléry Pelvillain, Antoine Mayol, Thybalt Marschutz et Mathis Person.

3 Repartition des charges

Afin de mener à bien ce projet nous avons décidé de travailler à deux sur les étapes qui nous paraissaient les plus complexes pour nous permettre d'être plus efficace sur certaines parties. Enfin nous avons assigner une personne pour les plus petites étapes.

Membres Tâches	Cléry	Antoine	Thybalt	Mathis
Traitement de l'image				
Détection de grille				
Réseau de neurones				
Résolution de sudoku				
Base de donnée				
Interface				

4 État D'avancement du projet

Nous avons remplis toutes les tâches demandées pour la première soutenance et nous sommes confiant quant au travail qu'il nous reste à réaliser. Nous avons un bon rythme de travail et nous avons eu une avancé constante sur le projet. Ainsi nous comptons continuer sur cette lancée pour pouvoir terminer le projet à temps.

Tâches	État	État d'avancement des taches
Traitemet de l'image		90%
Détection de grille		100%
Réseau de neurones		70%
Résolution de sudoku		100%
Base de donnée		100%
Interface		90%

5 Pré-traitement

Le traitement d'image est primordial pour un OCR. Cette étape permet de conserver un maximum d'informations, tout en optimisant le temps d'exécution et la taille occupée par les données.

La précision du pré-traitement est alors déterminante pour la suite du projet puisque chaque bruit ou impureté peut empêcher la récupération correcte des lignes et des cases du sudoku, rendant ainsi impossible les étapes à venir.

Ainsi, plus nombreux seront les filtres appliqués, plus haute sera la précision des résultats futurs. Le pré-traitement de l'image est composé de trois phases principales, présentées chacunes dans l'ordre d'exécution dans la section suivante.

5.1 Structures

Pour manipuler une image en C, il faut utiliser la librairie SDL. Cependant, cette structure est très complète mais trop complexe pour notre utilisation.

Il fallut alors créer notre propre structure d'Image, dans le but de simplifier la manipulation, tout en améliorant la vitesse de calcul, l'utilisation de la mémoire et le stockage.

C'est pourquoi nous utilisons des structures de données dynamique, en allouant avec précaution chaque adresse mémoire.

Cette structure est également accompagnée d'une dizaine de fonctions, utiles d'une part pour la conversion avec SDL, mais aussi pour en faciliter la manipulation dans les différentes méthodes de traitement.

5.1.1 Structure Pixel

Cette structure permet de re-créer un pixel mais de manière épurée, en ne gardant uniquement les paramètres essentiels pour notre projet.
Nous passons effectivement d'une dizaine de paramètres pour à SDL à uniquement 3 paramètres:

r Code rouge

g Code vert

b Code bleu

Ces trois valeur sont des entiers non signés compris entre 0 et 255. A noter que la valeur de ces trois parametres sont identiques à partir du premier pré traitement(Nuances de Gris), mais la dissociatioon de ces trois valeurs nous est particulièrement utile pour la conversion avec une Surface de la librairie SDL.

5.1.2 Structure Image

De manière identique, la structure d'image est conçue pour utilisation simple et optimisée. Cette structure est composée de trois paramètres:

width Largeur de l'image

height Hauteur de l'image

pixels Un tableau de Pixel, de taille width x height

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

Image initiale

5.2 Transformation en nuances de gris

Cette première phase de traitement est destinée à transformer l'image initiale de couleur en nuances de gris. Cela permet de réduire la taille de la donnée de chaque pixel.

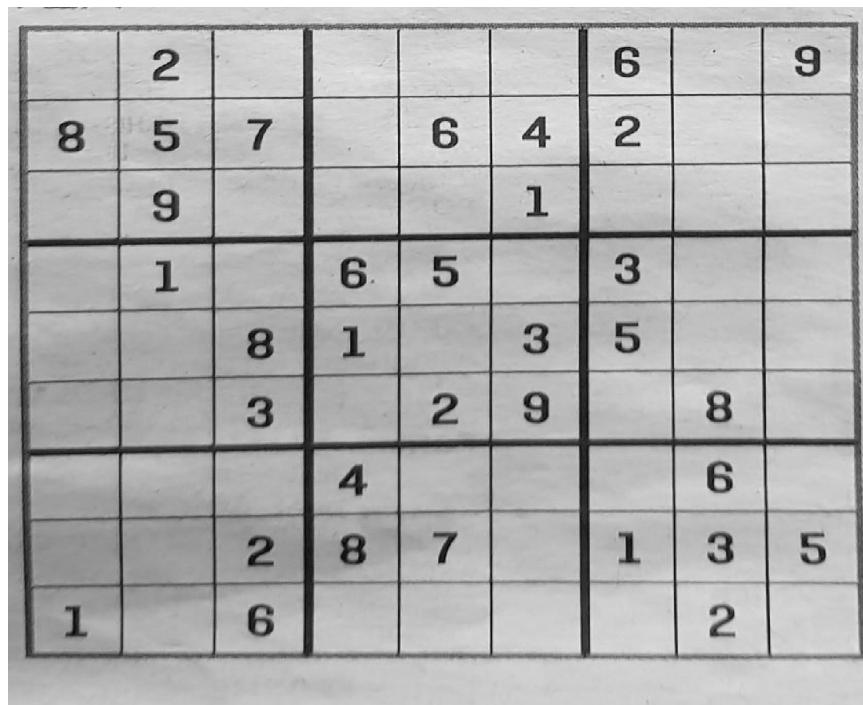
En effet, l'image passe d'une structure à trois variables r, g et b, chacunes contenant une valeur de 0 à 255, à une seule valeur comprise entre 0 et 255. Elles sont ses valeurs respectives du rouge, vert et bleu de chaque pixels de l'image.

Il est maintenant plus simple d'effectuer des traitements sur l'image étant donné que les trois variables ont désormais la même valeur.

Les valeurs respectives du rouge, vert et bleu du pixel seront à présent identiques puisqu'elles seront actualisées à la nouvelle valeur de gris suivante:

$$p(x, y) = r \times 0.3 + g \times 0.59 + b \times 0.11$$

Ainsi, dans les calculs futurs, la valeur de la couleur du Pixel aux coordonnées (x,y) sera résumée à une seule valeur représentée par p(x,y).



Nuances de gris

5.3 Filtres intermédiaires

5.3.1 Inversion

Les chiffres du sudoku ainsi que les grilles sont de couleur noire par défaut. Or, il est plus simple pour le réseau de neuronne ainsi que pour la détection des cases d'avoir des chiffres et des lignes blanches.

Cela est du au fait que la couleur noire est proche de zéro et le blanc située proche de 255. Il y aura ainsi plus de précision dans les calculs à venir étant donné que la plage des données est plus large.

Ce traitement est appliqué lors de la normalisation de la luminosité de manière pour des raisons d'optimisation.

5.3.2 Normalisation de la Luminosité

Ce traitement fut rapide à implementer, étant donné qu'il d'agit d'un simple parcours des pixels en actualisant chaque pixels de la manière suivante:

$$p(x, y) = 255 - p(x, y) \times (255/m)$$

avec m: valeur maximale d'un pixel dans l'image

5.3.3 Réduction des contrastes

Au fur et à mesures des étapes, nous avons constaté une perte massive des lignes du sudoku. La milleure solution est alors de réduire le contraste de l'image. Ce filtre peut etre calculé par la fonction suivante:

$$p(x, y) = \begin{cases} (k + 1) \times (255/f) & si \quad p(x, y) \in [k \times (255/f), (k + 1) \times \frac{255}{f}] \\ p(x, y) & sinon \end{cases}$$

avec f: facteur flottant de contraste et $k \in [0, f]$.

Cependant, ce filtre n'est pas automatique et nécessite de déterminer manuellement un facteur adapté.

La valeur la plus optimale pour une image de sudoku est 0.7.

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

Filtres intermédiaires

5.4 Lissage

Par la suite, il est question d'appliquer différents algorithmes pour affiner l'image et réduire au maximum les différents parasites et impuretées.

Cette étape peut contenir plusieurs méthodes additionnées. Le plus efficace reste principalement le Flou Gaussien.

Ce traitement consiste tout d'abord à générer une matrice carrée, plus communément appelé noyau ou “kernel” en anglais, générée par une fonction mathématique déviante de la fonction Gaussienne adaptée pour un espace à 2 dimensions.

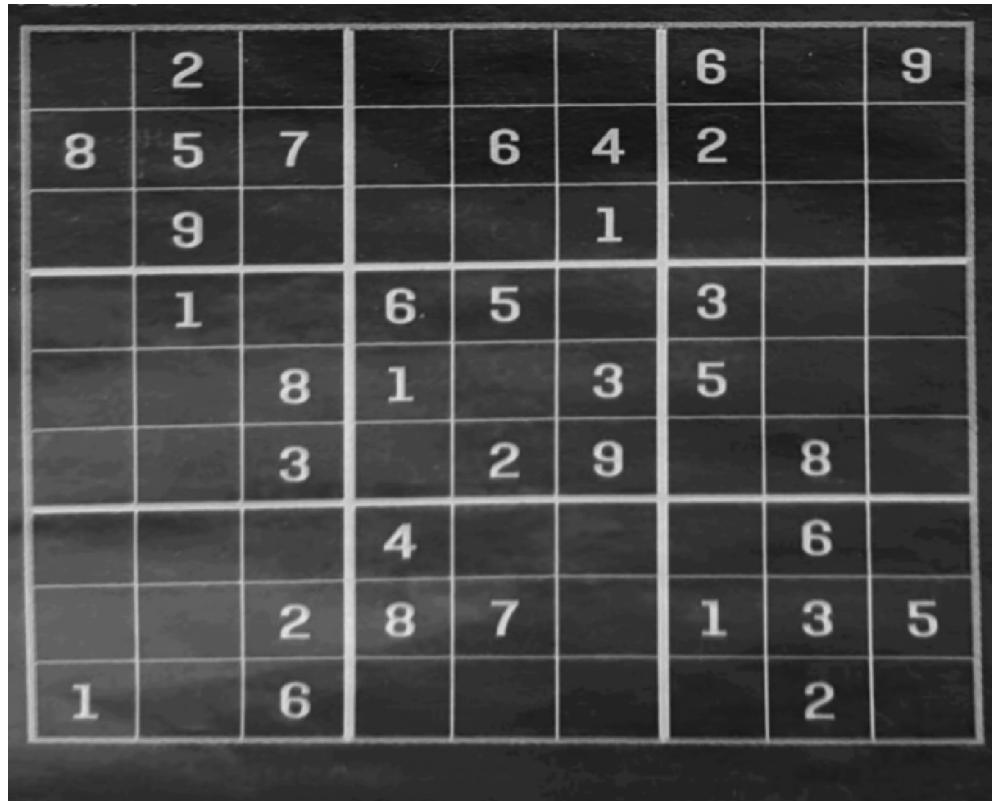
Il utilise globalement le principe d'une matrice de poids, distribué à une certaines valeur définie selon sa taille.

Le noyau alors généré, nous pouvons maintenant l'appliquer avec le principe de convolution. Cela consiste à positionner le noyau autour d'un pixel central, multipliant la valeur chacun de ses voisins aux valeurs au même indice du noyau.

Chaque produit est additionné et la somme finale est alors appliquée au pixel central.

En répétant cette dernière étape pour chaque pixel de l'image, nous obtenons un flou étendu sur toute l'image, plus ou moins intense selon la taille du noyau généré. L'idéal pour des images de Sudoku est un noyau de 5×5 .

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



Lissage par flou Gaussien

5.5 Binarisation

Cette étape est la phase finale consiste à transformer l'image en noir et blanc. Il existe des dizaines d'algorithmes différents pour cette étape.

Tout d'abord, nous avons implémenté un seuillage adaptatif, mais cette méthode était trop aléatoire car il fallait changer la valeur manuellement du seuil et de la taille de la zone à traiter pour chaque image.

Nous avons ensuite implémenté la méthode du seuillage basique, mais il fallait renseigner cette valeur manuellement pour chaque image.

Nous avons alors essayé d'améliorer cette méthode en l'adaptant par zone de pixels, une première fois en faisant la moyenne des pixels l'entourant, puis en y appliquant la convolution avec un noyau de taille 12 x 12.

Encore une fois, le résultat obtenu n'était pas convainquant. Finalement après de multiples tentatives, la méthode la plus adaptée pour des images de sudoku est la méthode d'Otsu.

5.5.1 Otsu

Cet algorithme est utilisée pour déterminer cette valeur de seuil en fonction des différentes propriétés de l'image. Cet algorithme permet d'effectuer un seuillage automatique à l'aide de l'histogramme de l'image.

Il consiste à trouver le plus haut seuil en calculant la variance intra-classe maximale de la manière suivante:

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_0(t)\omega_1(t)[\mu_0 - \mu_T])$$

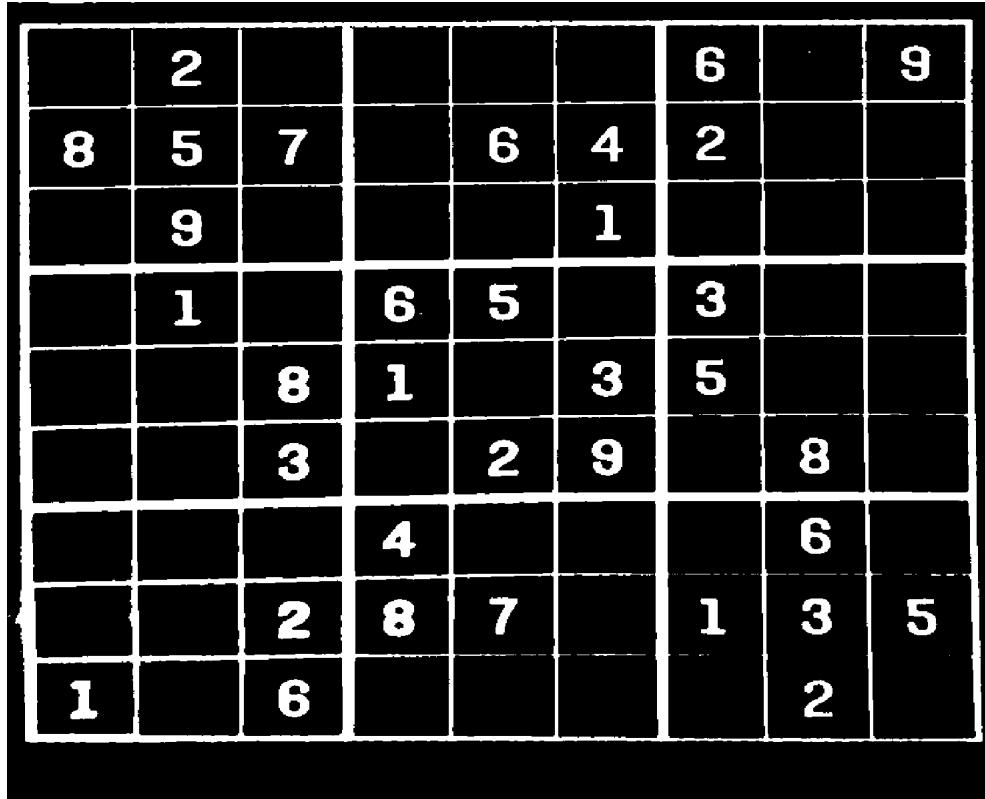
5.5.2 Seuillage

Après avoir appliqué l'algorithme d'Otsu, nous allons pouvoir utiliser cette valeur comme seuil. En effet, cette dernière étape consiste à parcourir chaque pixel de l'image tout en appliquant une unique condition:

- Si la valeur du pixel est strictement inférieure au seuil, on la change pour 0 (noir).
- Sinon on la met à jour à 255 (blanc):

$$p(x, y) = \begin{cases} 255 & si \quad p(x, y) \geq t \\ 0 & sinon \end{cases}$$

avec t: valeur du seuil d'Otsu



Binarisation par Otsu

5.5.3 Détection des lignes

Ce dernier filtre post-binarisé est essentiel pour le traitement futur de l'image. En effet, le filtre de Sobel permet le détournage des différentes lignes rencontrées dans l'image.

Cette étape facilite de manière conséquente la détection des contours, étant donné que ce filtre détouche les arrêtes et les cases du Sudoku.

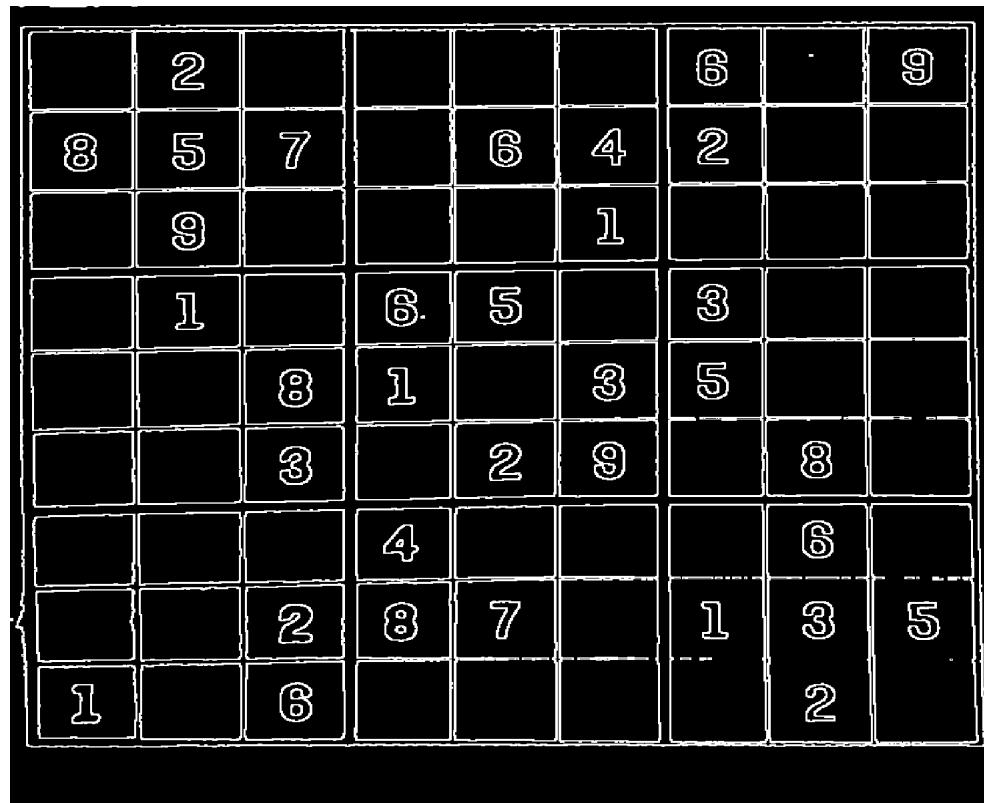
Cet algorithme utilise également la convolution, mais cette fois avec deux noyaux spécifiques de taille 3x3.

Noyau pour les lignes horizontales : $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

noyau pour les lignes verticales : $G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

La valeur finale chaque pixel est alors déterminé par le calcul suivant:

$$G = \sqrt{G_x + G_y}$$



Sobel

5.6 Transmission des données

Pour transmettre notre image finale à la prochaine partie du traitement, il est nécessaire d'effectuer une conversion.

Les données contenues dans l'image sont encore des Pixels et la prochiane étape requiert des entiers binaires.

Pour cela un seul parcours est nécessaire, il suffit de convertir chaque pixel en appliquant leur condition suivante:

$$p(x, y) = \begin{cases} 1 & \text{si } p(x, y) = 255 \\ 0 & \text{sinon} \end{cases}$$

Pour conclure, le traitement d'image nous perme de transformer une image de couleur en une simple matrice binaire de taille identique à l'image.

6 Detection de grille

Une fois notre image obtenue sous forme d'une matrice "binaire", nous devons extraire la grille de sudoku de l'image.

Pour se faire nous avons essayé d'utiliser un alggorithme de composants connexes.

Celui-ci nous permet de récupérer le plus grand composant, ou bien sous-matrice de notre matrice.

0	0	0	0	0	0	0	0	0
0	1	1	0	0	3	3	3	0
0	0	1	0	0	0	3	3	0
0	1	0	0	0	3	0	3	0
0	0	0	0	0	0	0	0	0
0	0	0	2	2	0	0	0	0
0	0	0	0	2	0	0	0	0
0	0	0	2	0	0	0	0	0
0	0	0	0	2	0	0	0	0

Une fois notre plus grand composant connexe repéré il suffit d'extraire la sous-matrice correspondant à ce composant. Nous avons maintenant la grille du sudoku isolé.

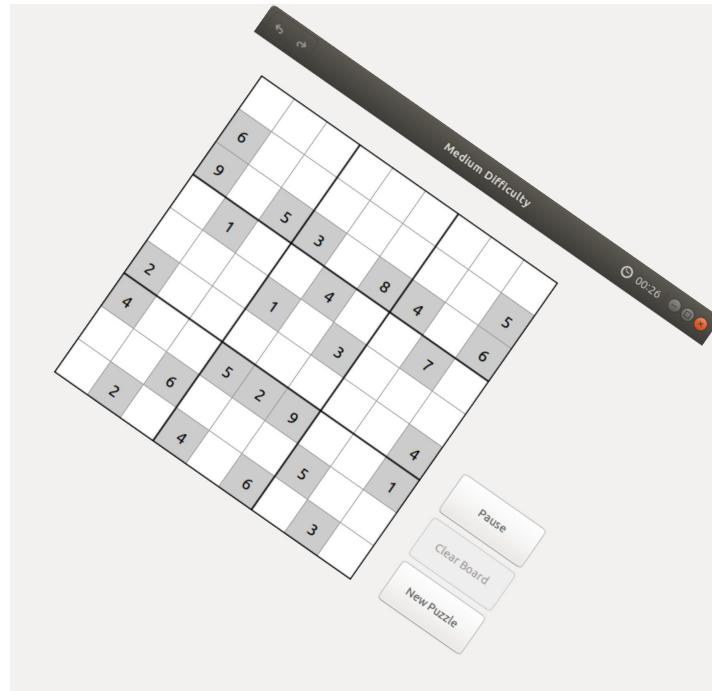
De ce fait il est bien plus simple de détecter les lignes de cette grille. En effet tout les bruits présents en dehors de la grille ne nous gèneront plus.

Cependant cet algorithme ne s'adapte pas au bruit qui pourrait se créer suite au flou gaussien. Ainsi nous avons décidé d'écarté cet algorithme qui aurait été un plus dans notre projet.

Pour le remplacer nous utilisons l'algorithme "flood fill" qui détermine les différentes régions de l'image par expansion d'une couleur sur cette dernière. Grâce à cet algorithme il nous est bien plus aisés de récupérer la grille du sudoku le plus précisément possible.

6.0.1 Rotation de l'image

Pour cette première soutenance nous devons implémenter une fonction qui réajuste manuellement la rotation d'une image.



Grille de sudoku penchée

Après un traitement, nous pourrons remettre cette grille de sudoku droite pour pouvoir la traiter plus facilement.

Bien sûr ceci n'est pas automatique pour l'instant, donc nous devons connaître l'angle de rotation avant de pourvoir l'appliquer. D'ici à la prochaine soutenance, la rotation sera automatique

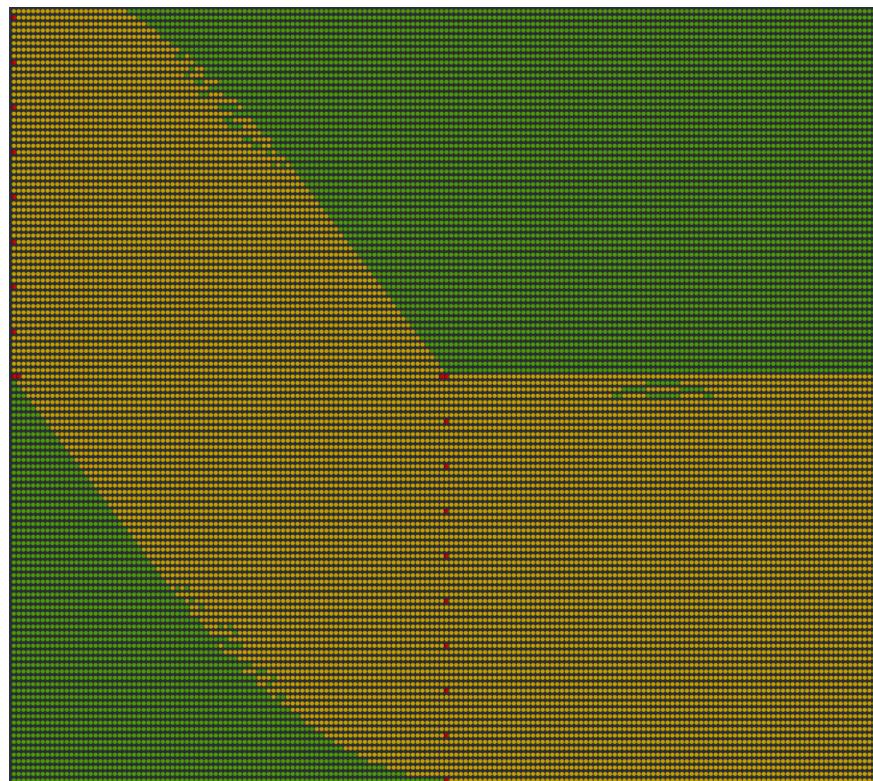
Le principe de rotation est de récupérer tous les bits d'une image pour pourvoir les décaler. Pour cela nous utilisons une formule mathématiques, celle de rotation de matrices inverses.

Ce qui nous permet de pouvoir récupérer toutes les nouvelles coordonnées de chaque bits pour recréer une nouvelle image.
Cette nouvelle image sera ensuite envoyé au traitement.

6.0.2 Découpage de la grille

Une fois notre image tourner dans le bon angle, notre image est prête à être découper pour ensuite en extraire les cases. L'algorithme le plus efficace pour détecter les lignes sur une image est celui de Hough.

Ce dernier va ajouter dans un tableau toutes les droites possible à partir de tout les points présent sur notre image.



Transformation de hough d'une matrice

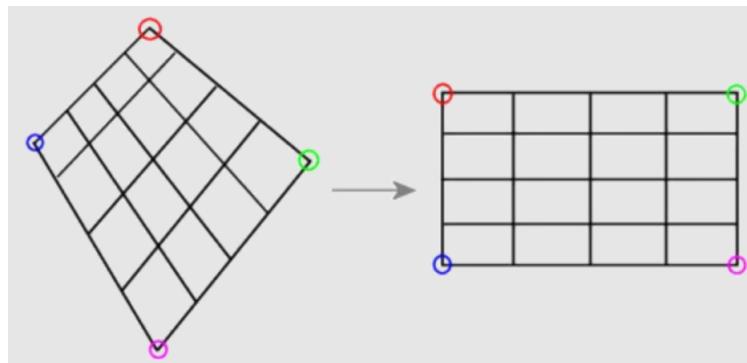
Ainsi les droites qui seront les plus fréquentes seront celle qui nous intéressent le plus. Cependant pour ne pas récupérer des lignes qui ne font pas partient du sudoku il est préférable de filtrer ces dernières.

Ensuite plus une ligne est grande plus elle a de chance de être sélectionnée. C'est pourquoi il faut tout de même vérifier si chaque ligne est à une bonne distance des autres en faisant une moyenne des distances entre chaque lignes.

Une fois nos vingt lignes trouvées nous pouvons extraire les intersections entre celle-ci. Nous avons maintenant les coordonnées de nos cases.

En appliquant quelque valeur d'ajustement et en redimensionnant ces cases nous obtenons la forme finale des cases qui sont maintenant prêtes à être transmises au réseau de neurones.

Grâce à cette algorithme nous pouvons également récupérer l'angle d'inclinaisons de notre matrice afin d'appliquer un algorithme d'homographie dans le cas où l'image serait inclinée.



Transformation par homographie

7 Réseau de neurones

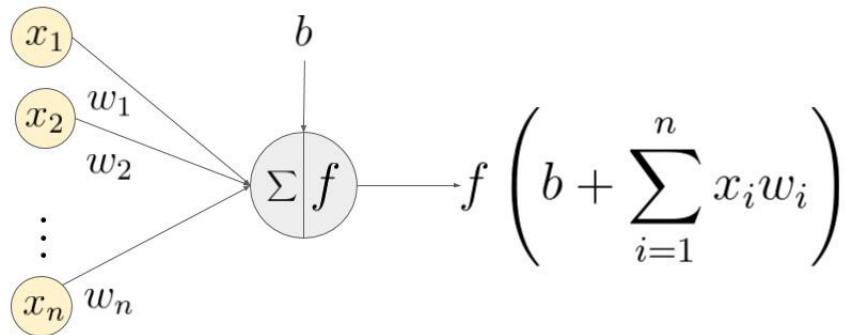
7.1 Fonctionnement

Un réseau de neurones est une structure de données implémentant un algorithme dit "*probabiliste*". En d'autres termes, il permet de déterminer la probabilité d'un événement de notre choix.

Les possibilités sont énormes mais sont parfois restreintes par les données disponibles. En effet, pour qu'un tel réseau soit efficace, il faut au préalable l'entrainer avec beaucoup de données différentes afin qu'il "apprenne" à différencier les cas et faire ses propres "choix".

Il est composé de différentes couches (généralement 3) dont une dite d'"entrée", une "cachée" et une de "sortie".

Les différentes couche communiquent entre elle grâce à la formule suivante :



An example of a neuron showing the input ($x_1 - x_n$), their corresponding weights ($w_1 - w_n$), a bias (b) and the activation function f applied to the weighted sum of the inputs.

f étant une fonction d'activation (sigmoid, ReLu, ...)

Chaque neurone d'une couche N sera lié à chacun de neurones de la couche $N-1$ (sauf la couche d'entrée) et sa valeur sera modifiée grâce à la formule précédente.

Enfin pour s'améliorer, il faut apprendre de ses erreurs. Pour cela, il utilise la formule de *descente de gradient* qui lui permet de modifier la valeur de ses noeuds par rapport à taux erreur.

$$w_{k+1} = w_k - \eta \frac{1}{m} \sum_{i=1}^m \nabla f_{w_k}(x^i)$$

La descente de gradient

7.2 Le XOR

Pour cette première soutenance, nous devions implémenter un réseau de neurone capable d'apprendre la fonction "OU EXCLUSIF" (aussi appelé "XOR").

Voici une représentation de ce à quoi ressemble le réseau que nous avons créé :

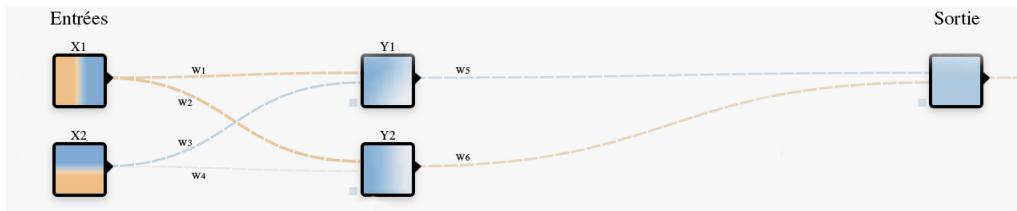


Schéma d'un réseau de neurones simple, le XOR.

Les "Entrées" X1 et X2 sont les données que nous entrons dans le réseau, ici des 0 et des 1. Les neurones Y1 et Y2 font parties de la "Couche cachée" du réseau.

La "Sortie" est composée d'un neurones qui nous donneront notre résultat, ici la probabilité d'obtenir un 1.

Les liens entre les neurones sont représentés par les pointillés W1, W2, ... Ces liens peuvent modifier une donnée qui les traversent grâce à leur poids (un nombre décimal).

Le réseau de neurones pour apprendre un XOR est assez petit, notre réseau final sera composé de plusieurs couches cachées et de beaucoup plus de neurones par couche.

Il ne nous reste plus qu'à généraliser cette solution pour qu'elle s'adapte au programme final : la détection d'un chiffre sur une image.

7.3 Compréhension de chiffres

7.3.1 Vue d'ensemble des modifications

Afin de relever ce défi, nous avons garder le même principe que pour résoudre le XOR, mais avons modifier certains aspects.

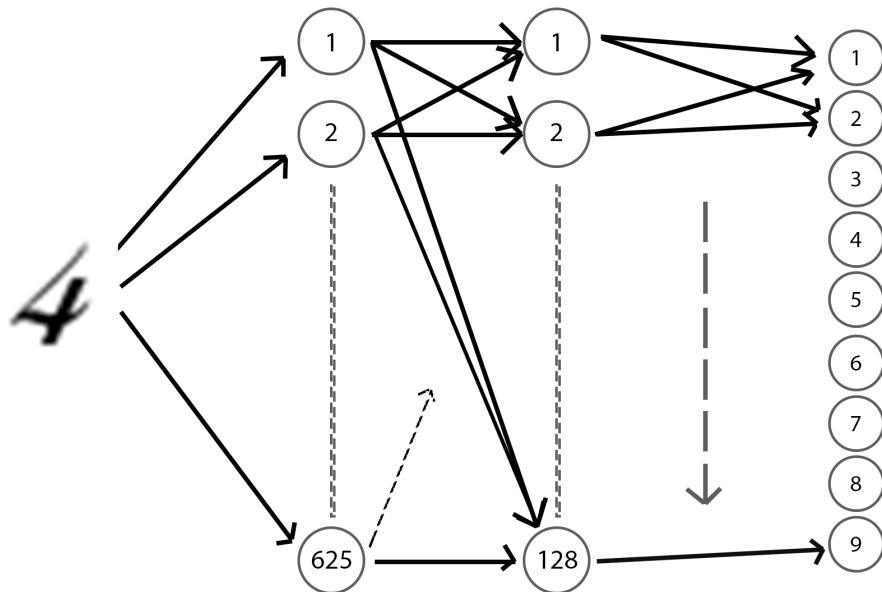


Schéma du réseau de neurones finale.

Premièrement, nos entrées. Cette première couche sera composée de 728 neurones (un pour chaque pixel d'une image de 28 * 28 pixels), cette résolution nous permet de rester précis tout en limitant la quantité de données que nous sauvegardons.

Ensuite, notre couche de sortie ne comporte plus qu'un seul neurone, mais dix. Effectivement, chaque neurones correspond à un chiffre (de 0 à 9) et nous permet de déterminer la nature de l'entrée.

En effet, il nous suffit de retourner le neurones ayant la plus haute probabilité d'être correct (le plus proche de 1 possible)

Enfin la couche cachée s'est épaissit et comporte maintenant 128 neurones, ce nombre à été determiner après plusieur tests et semble approprié pour ce problème.

Tout réseau de neurones possèdent une fonction d'activation, nous avons choisis *Sigmoid* disponible dans la librairie standard du C.

7.3.2 Type de données sauvegarde du réseau

Afin de gérer le réseau et ses différentes couches, nous avons créer plusieurs liste de *double* (nombre à virgule) afin d'y insérer les différents paramètre des neurones. Il y en a 3 pour la couche caché, et 3 pour la couche de sortie.

Cette structure de données nous permet aisément de sauvegarder la totalité du réseau pour continuer de l'entrainer plus tard ou pour l'utiliser pour déterminer un chiffre.

Les différentes liste sont pour cela stocké dans des fichiers texte dans le dossier *neurones/* sous le nom de "*nerons[i].txt*" avec [i] l'ordre dans lequelle la liste en utilisé.

Afin de charger les fichiers pour réutiliser le réseau, nous avons utilisé la fonction "get_lines" de la librairie standard "stdio.h" qui nous as permit de parcourir nos fichier ligne par ligne.

Pour atteindre les différents fichier de nos dossier, nous avons utilisé la librairie "dirent.h" incluse dans les standard du C.

7.3.3 Base de donnée

Qui dit réseau de neurones dit apprentissage, et pour apprendre, il faut de quoi s'entraîner. C'est pour quoi nous avons créé notre propre banque d'image afin de préparer le réseau à reconnaître des chiffres écrits de manière différentes.

3 6 7 1 5 9

Extrait de la banque de données

Elle comporte une vingtaine de sous-dossiers, tous utilisant des polices d'écriture différentes, composés d'image représentant des chiffres de 1 à 9.

Nous avions dans un premier temps regroupé les images par chiffre (les différents 1 ensemble, les différents 2 ensemble, etc ...) mais cette méthode avait l'air de troubler le réseau et l'incitait à retourner 9 (le dernier testé).

7.3.4 Utilisation du réseau

Une fois créé et entraîné, son utilisation est assez basique.

Il reçoit les différentes cases précédemment détectées dans l'ordre, les passe une par une dans le réseau, et stocke les résultats dans une liste représentant le Sudoku afin qu'il soit interpréter et résolu.

L'important dans cette partie était de bien entraîner notre réseau et le préparer à reconnaître tout type de chiffre.

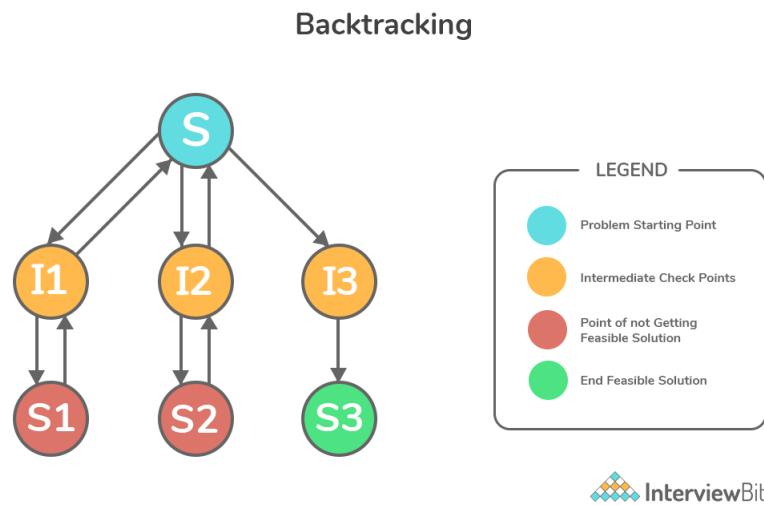
8 Resolution d'un sudoku

Une étape tout autant évidente qu'importante de ce projet est la résolution du sudoku.

Nous devions alors analyser un fichier texte, en ressortir les données, résoudre le sudoku puis sauvegarder le résultat.

Pour analyser le fichier, nous avons créé un "parser" qui nous permet de nous y balader afin ressortir sous forme de tableau un tout les chiffres ainsi que leur position.

Suivant cela nous avons utilisé une méthode de résolution de sudoku vue en première année appelé "BackTracking".



Présentation du principe de "BackTracking"

Cet algorithme de résolution fonctionne suivant ce principe : On parcours un à un les éléments du tableau et quand on arrive sur une case vide, on y assigne le premier chiffre pouvant se trouver sur cette case.

On répète le principe pour les cases suivante jusqu'à ce qu'une d'entre elles soit vide. Alors nous revenons en arrière pour passé au chiffre suivant de la dernière case vide.

En effet, si la grille ne possède que très peu de chiffre au départ, la résolution sera plutôt lente. Ainsi des optimisations sont possibles et seront réalisées pour la prochaine soutenance.

9 interface

L'interface est un point tout aussi important dans notre projet. Celle-ci doit être lisible et facile d'utilisation pour l'utilisateur.

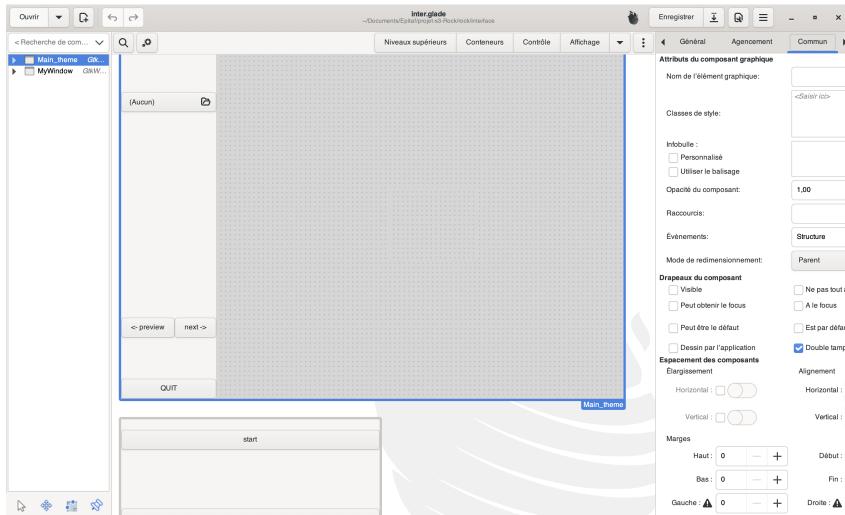
Pour créer cette interface nous avons utilisé plusieurs outils que nous allons détailler pour rendre l'expérience du logiciel la plus agréable possible.

9.1 Glade

Pour tout le style graphique, ainsi que la mise en place des différents boutons permettant une interaction avec l'utilisateur nous avons décidé d'utiliser glade.

Glade est un outil permettant le développement rapide et facile d'interfaces utilisateur pour la boîte à outils GTK et l'environnement de bureau GNOME.

Les interfaces utilisateur conçues dans Glade sont sauvegardées en XML.



Fenêtre d'utilisation de glade

Grâce à la sauvegarde en XML nous allons pouvoir l'utiliser pour créer une interaction avec les différents boutons proposés par le logiciel.

On peut noter d'ailleurs que l'utilisation de certains boutons sont très importants, notamment celui qui permet de chercher dans l'ordinateur une image choisie par l'utilisateur, mais nous en reparlerons plus tard.

9.2 GTK3+

Glade nous permet de créer une interface utilisateur pour que celui-ci puisse interagir avec notre programme de résolution de sudoku. Ainsi en utilisant GTK nous allons pouvoir programmer tous les différentes interactions de notre interface.

Pour pouvoir récupérer toutes les différentes informations sauvegarder en XML par Glade nous pouvons utiliser *GtkBuilder* qui nous permet d'utiliser les différentes formes et fonctions des différents boutons.

Grâce à GTK nous allons aussi pouvoir afficher des images ce qui donnera un aperçu à l'utilisateur du fonctionnement de la résolution de sudoku.

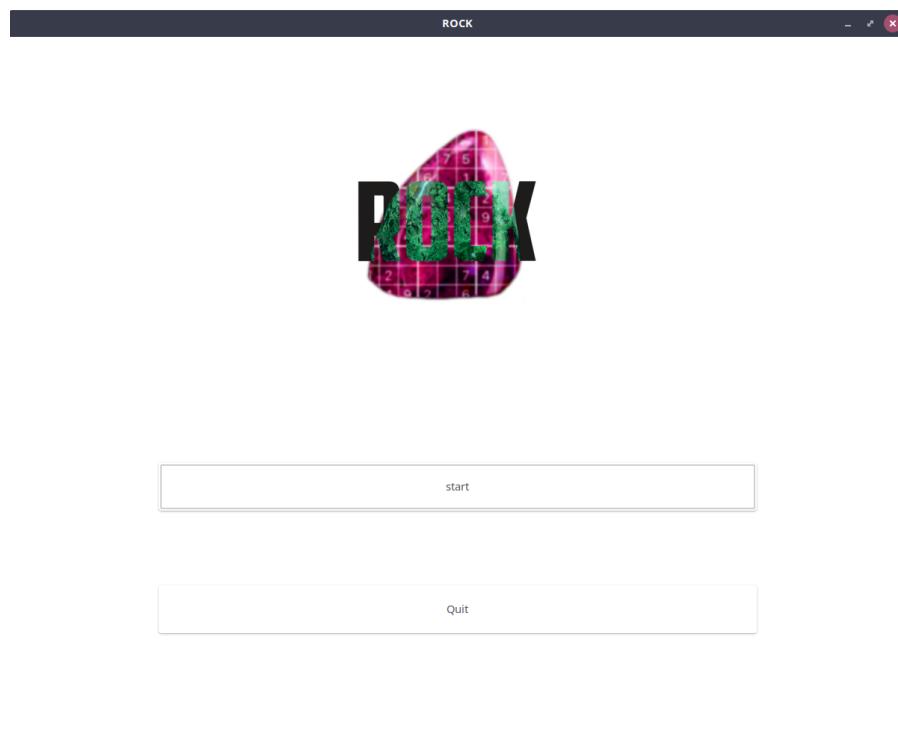
D'abord pour voir si il a choisis la bonne image, ainsi que la résolutions de son sudoku.

En effet, GTK est un très bon outils pour créer et gérer une interaction avec un utilisateur par le biais d'une interface graphique.

9.3 Utilisation

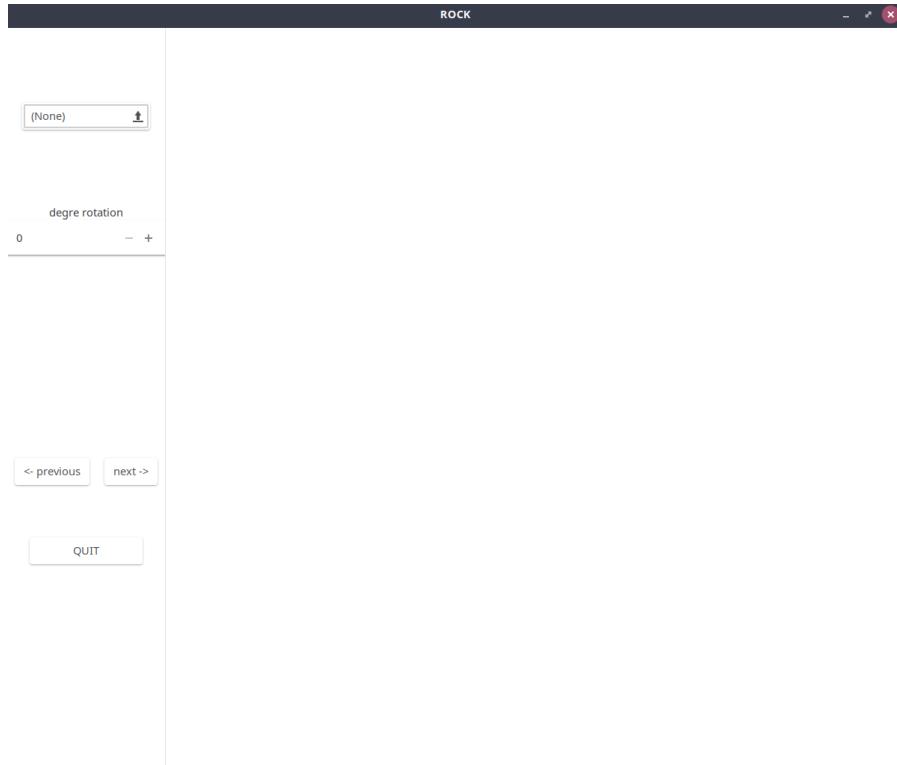
Grace à ce logiciel notre interface se devise en deux sections. La première, un menu d'accueil avec l'affichage de notre logo ainsi que deux bouton.

Ces deux boutons nous permettent, respectivement , soit commencer la procédure de résolutions de sudoku, soit de quitter le logiciel.



Ecran d'accueil du logiciel

Ensuite nous avons une seconde fenêtre qui elle va faire fonctionner notre projet. Cette fenêtre est la fenêtre principale qui permet à l'utilisateur de voir les différents stades d'avancement de l'algorithme.



Ecran principal du logiciel

Comme on peut le voir, quand on arrive sur cette fenêtre une grande partie de celle ci est vide.

Mais en appuyant sur le bouton en haut à gauche on peut choisir une image dans notre ordinateur qui sera directement affiché.

Grâce à ce bouton nous allons pouvoir récupérer le chemin de l'image et l'utiliser pour les différentes étapes de l'OCR.

Ainsi cette image sera envoyée au Pré-traitement, ensuite à la détection de grille puis au réseau de neurones pour enfin terminer par l'algorithme de résolution de sudoku.

Après l'affichage de l'image en brut, tout le traitement de résolution sera effectué, et deux images ressortiront pour les afficher à deux différents stade de la résolution.

9.3.1 SDL2

Ces deux images sont créées a partir d'un script SDL. Ce script va recevoir une matrices données après le traitement du réseau de neurones.

La première est l'image de la grille modifiée envoyée par l'utilisateur que vous pouvez voir ci dessous.

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

Grille de sudoku résolue

Cette image représente le sudoku avec seulement les cases initiales de la grille de sudoku affichée en noir

Ainsi la deuxième image générée par cet algorithme est la grille de sudoku résolue. Pour ce faire nous réutilisons la grille précédente que nous envoyons à l'algorithme de résolution de sudoku qui nous permet d'avoir une nouvelle grille résolue.

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

Grille de sudoku résolue

Mais le petit problème rencontré est (comment savoir quel sont les nouvelles valeurs rajoutées dans la grille pour pouvoir afficher les chiffres d'une couleur différente).

Pour cela il nous a suffit de soustraire la nouvelle grille par l'ancienne ce qui nous permet d'avoir tous les nouveaux chiffres et de pouvoir créer ces images.

Pour pouvoir créer c'est image on utilise SDL qui nous permet de superposer une image sur une autre on a donc eu la possibilité d'utiliser comme base la grille de sudoku et de superposer différentes images dessus pour avoir une grille résolue.

Pour pouvoir effectuer cette superposition nous pouvons remercier la fonction (SDL_BlitSurface).

9.4 Interaction de l'utilisateur

Ainsi dans cette deuxième fenêtre l'utilisation a la possibilité d'interagir avec trois boutons supplémentaires. Le premier bouton est le bouton qui permet de quitter l'interfaces.

Les deux autres boutons permettent a l'utilisateur de voir l'évolution de l'algorithme avec un bouton (Next) qui permet d'avancer dans les traitements de l'image.

Et un bouton(preview) qui lui permet de revenir en arrière pour voir les étapes précédentes du traitement.

Pour le style de l'interface nous avons choisi quelque chose de sobre notamment avec de la couleur clair (Blanche), nous prenons en compte une interface simple et épurer comme ceux le faire différente entreprise (apple), ce qui permet d'avoir une interface simple pour l'utilisateur qui lui permet de l'utiliser instinctivement

9.5 Base de donnée

Pour sauvegarder nos images, nous avons utilisé la fonction "SDL_SaveBMP".

Ensuite, nos données restant basique(entier, flottant, ...), la librairie "stdio.h" était suffisante.

9.6 Déroulement final par étapes

- Pré-traitement :
 - Nuances de gris
 - Normalisation de l'éclairage
 - Inversion
 - Modification du contraste
 - Lissage
 - Binarisation par seuil
 - Détection des lignes
- Détection de grille :
 - Recuperation du Sudoku
 - Rotation de l'image
 - Découpage de la grille
 - Collecte des cases
- Réseau de neurones
 - Prédiction du réseau pour chacune des cases
 - Reconstitution du Sudoku dans un tableau
- Résolution du Sudoku
 - Récupération des données du réseau
 - Calcul des possibilités
 - Stockage du résultat
- Interface
 - Affichage du résultat dans une matrice prédéfinie

10 Bibliographie

10.1 Outils

SDL2.0 : <https://www.libsdl.org>

GTK+3 : <https://www.gtk.org>

GitHub : <https://github.com>

10.2 Sources

Neron Playgrounds : <https://playground.tensorflow.org>

Kongakura : <https://www.kongakura.fr>

PCSI-Kleber: <http://pcsi.kleber.free.fr>

TowardsDataScience: <https://towardsdatascience.com>