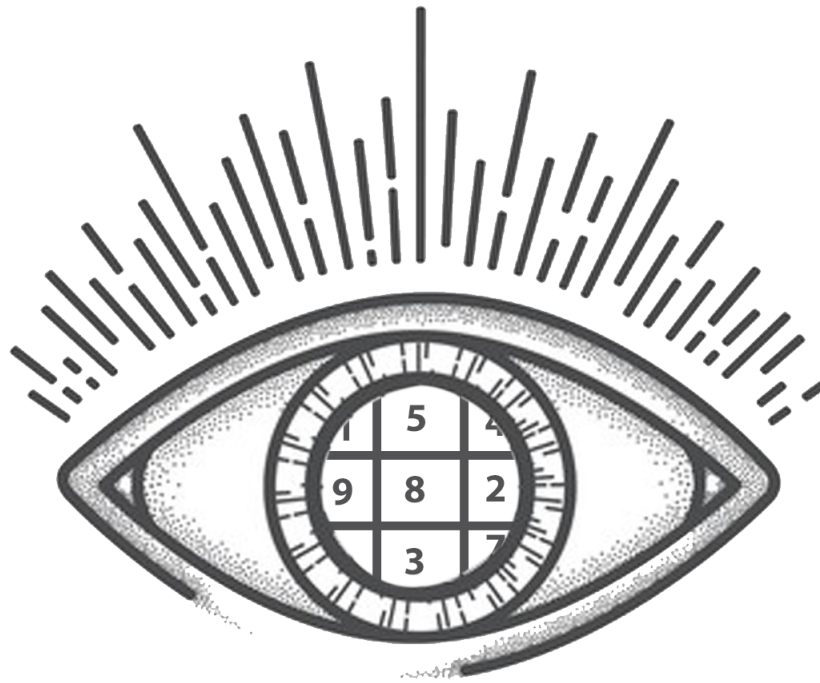


Rapport première soutenance OCR

SUDO.C.R

Giovanni Le Bail | Kristen Quesnel | Tudual Lefevre | Aloïs Colléaux-Le Chêne

Octobre 2021



SUDO.C.R

Table des matières

1	Introduction	3
1.1	Qu'est ce qu'un OCR ?	3
1.2	Qu'est ce que SUDO.C.R ?	3
2	Pré-traitement de l'image	3
2.1	Règles générales du traitement d'image	3
2.2	Mise en niveau de gris	4
2.3	Réduction de bruit	4
2.4	Flou linéaire	4
2.4.1	Flou gaussien	5
2.4.2	Filtre de Kuwahara	5
2.5	Binarisation	6
2.6	Détection des lignes	7
2.7	Rotation d'image	8
3	Interface Graphique	10
3.1	Identité graphique	10
3.2	Interface utilisateur - Utilisation	11
3.3	Interface utilisateur - Artistique	12
4	Le solver de Sudoku	13
4.1	Le principe de fonctionnement	13
5	Réseau de neurone	14
5.1	Le principe de fonctionnement	15
5.2	Le principe d'entraînement du réseaux de neurone	15
6	Conclusion	17
6.1	Bilan de la première soutenance	17
6.2	Objectifs seconde soutenance	17

1 Introduction

Sécu Studio, se lance dans un nouveau projet, SUDO.C.R (SUDO pour Sudoku et sudo pour Linux et OCR). Un nouveau projet pour le Sécu Studio toujours composé de ses 4 membres (Aloïs Colleaux-Le Chene, Tudual Lefeuvre, Kristen Quesnel et Giovanni Le Bail)

Pour ce projet de début de 2 ème année à l'EPITA, nous devons concevoir un logiciel capable de reconnaître des chiffres dans une grille de sudoku (partie "OCR") ainsi que de la résoudre.

Ce projet se déroule sur 4 mois durant lesquels nous devons repartir la grande charge de travail afin de finir ce projet dans les temps. Nous découperons ce projet en 4 parties :
- Interface graphique - Prétraitement d'image - Reconnaissance de caractères - Solveur de Sudoku

1.1 Qu'est ce qu'un OCR ?

OCR qui vient de "Optical Caractère Recognition", ou Reconnaissance optique de caractère, "désigne les procédés informatiques pour la traduction d'images de textes imprimés ou dactylographiés en fichiers de texte."¹, il s'agit d'un logiciel permettant de reconnaître du texte sur une photographie, dans notre cas une photographie de sudoku.

1.2 Qu'est ce que SUDO.C.R ?

Dans notre cas, notre logiciel sera capable de reconnaître puis résoudre une grille de sudoku.

Dans un premier temps, il conviendra de récupérer les différents chiffres contenus dans la grille, ainsi que leur position. Une fois ces informations récupérées, elles seront rentrées dans un solveur automatique de sudoku, qui nous retournera les couples chiffres, position dans la grille afin de pouvoir par la suite reconstituer la grille de sudoku initiale.

L'utilisateur disposera d'une interface graphique ergonomique, afin de permettre à tous d'utiliser simplement notre logiciel.

Pour ce faire, il nous faut mettre en place un réseau de neurones capable d'apprendre le XOR, une partie de pré-traitement des images afin de transmettre des images de qualité au réseau de neurones.

2 Pré-traitement de l'image

2.1 Règles générales du traitement d'image

De manière générale, un algorithme de traitement d'image s'applique sur chaque pixel d'une image. Cela signifie que cet algorithme va passer de façon itérative sur tout les

1. https://fr.wikipedia.org/wiki/Reconnaissance_optique_de_caract%C3%A8res

pixels de l'image et appliquer le même algorithme, qui va modifier les valeurs Rouge, Vert et Bleu (aussi appelées valeur RGB) de ce pixel. Dans les prochaines sections, lorsque nous parlerons d'un pixel, nous ferons référence à un pixel générique, qui sera appliqué à l'algorithme de traitement.

Il faut d'ailleurs noter que les pixels modifiés ne doivent pas interférer avec les pixels qui n'ont pas encore été modifiés par l'algorithme, ainsi pour certains algorithme, il sera nécessaire de stocker les nouveaux pixels avant de pouvoir réellement modifier l'image. On utilisera ensuite la notation P pour noter le pixel d'entrée de l'algorithme, et P' le pixel de sortie de l'algorithme.

2.2 Mise en niveau de gris

La mise en niveau de gris d'une image est l'une des premières étapes à appliquer lors d'un traitement d'image. Elle consiste à "supprimer" les couleurs d'une image en faisant la moyenne des valeurs RGB pour chaque composante du pixel.

Ainsi, pour chaque pixel P , si l'on pose R, G, B (resp. R', G', B') les composantes RGB de P (resp. P'), on a :

$$R' = G' = B' = \frac{1}{3}R + \frac{1}{3}G + \frac{1}{3}B$$

Il existe aussi une formule alternative représentant une vision plus réaliste pour l'œil humain où les coefficients ne sont pas tous égaux, mais le choix de l'une ou l'autre technique n'est pas important pour notre utilisation étant donné qu'il y a aura une étape de binarisation plus tard dans le processus.

L'étape de mise en niveau de gris est importante, car elle permet pour les algorithmes suivant de ne prendre en compte qu'une seule composante de couleur du pixel plutôt que les 3, simplifiant le développement des algorithmes.

2.3 Réduction de bruit

Le bruit d'une image signifie une fluctuation parasite et aléatoire de lumière ou de couleurs dans une image. Ce bruit est souvent présent lors de prise de photos, et peut causer de nombreux problèmes pour plus tard. Cependant, il est très difficile de réduire le bruit entièrement, mais il est possible de le réduire un peu, au moins de façon à ce qu'il soit moins impactant. Il existe plusieurs façons de réduire le bruit, et nous avons considéré trois d'entre elles pour notre programme.

2.4 Flou linéaire

Pour appliquer le flou linéaire, nous avons utilisé une technique répandue dans le traitement d'image : les matrices de convolution.

Une matrice de convolution est une matrice carrée, de taille impaire et souvent symétrique, avec des poids. Cette matrice sera centrée en un pixel, et la nouvelle valeur de ce pixel sera la moyenne des pixels autour, multipliée par leur poids respectif. Cette nouvelle valeur est donc la nouvelle valeur du pixel. En suivant ce raisonnement, une approche naïve serait de faire la moyenne des pixels aux alentours. Cette approche se nomme le flou linéaire, elle a cependant un inconvénient non négligeable, les pixels éloignés du centre, sont considérées

comme aussi important que les pixels plus centraux.
Ce problème est donc réglé par la technique suivante, le flou gaussien.

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

FIGURE 1 – Matrice linéaire de paramètre $n = 3$

2.4.1 Flou gaussien

Le flou gaussien répartit les poids en fonction de leurs distance par rapport au pixel. Ainsi les poids plus aux centres de la matrice seront plus importants, tandis que ceux plus proches des bords et des coins seront bien moindres. L'avantage principal est que ce filtre préserve les bords des images.

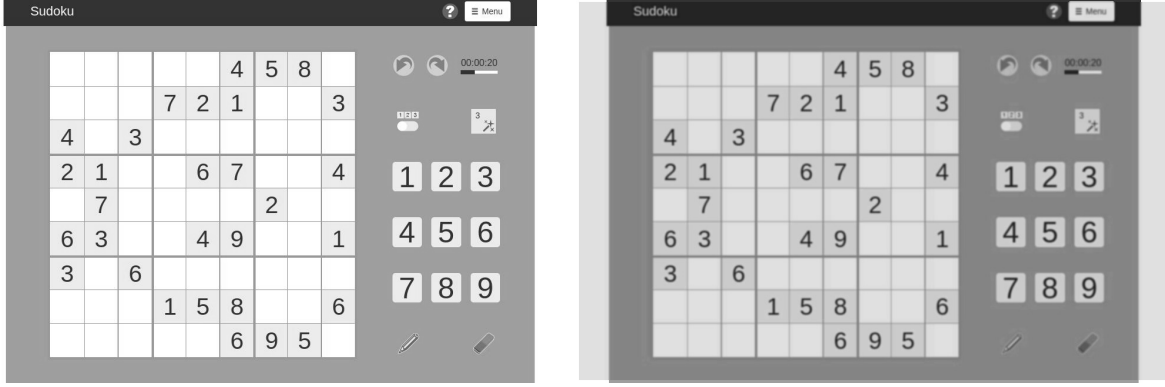


FIGURE 2 – Avant/Après application du flou gaussien

2.4.2 Filtre de Kuwahara

Bien que le flou gaussien est relativement efficace pour préserver les bords, le filtre de Kuwahara permet de lisser l'image, en préservant encore plus les bords. Supposons une fenêtre carrée de taille $2a + 1$ centrée autour d'un point (x, y) de l'image, et $I(x, y)$. Ce carré peut être divisé en quatre zones carrées plus petites $Q_{i=1..4}$ tel que

$$Q_i(x, y) = \begin{cases} [x, x + a] \times [y, y + a] & \text{si } i = 1 \\ [x - a, x] \times [y, y + a] & \text{si } i = 2 \\ [x - a, x] \times [y - a, y] & \text{si } i = 3 \\ [x, x + a] \times [y - a, y] & \text{si } i = 4 \end{cases}$$

où \times est le produit cartésien entre deux ensembles.

Il est notable de remarquer qu'il y aura des pixels présents dans plusieurs zones à la fois. La moyenne $m_i(x, y)$ et l'écart-type $\sigma_i(x, y)$ des quatre régions sont calculés et sont utilisés pour déterminer la valeur du pixel central. La valeur du filtre de Kuwahara $\Phi(x, y)$ pour chaque point (x, y) est donnée par $\Phi(x, y) = m_i(x, y)$ où $i = \arg \min_j \sigma_j(x, y)$

En d'autres termes, le pixel prendra la valeur de la moyenne de la zone la plus homogène.

a	a	a/b	b	b
a	a	a/b	b	b
a/c	a/c	a/b/ c/d	b/d	b/d
c	c	c/d	d	d
c	c	c/d	d	d

FIGURE 3 – Fenêtre utilisée par un filtre de Kuwahara

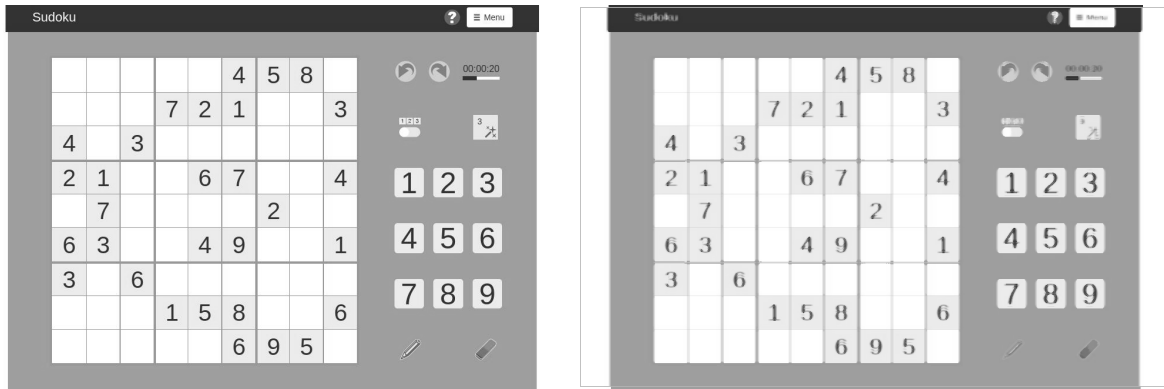


FIGURE 4 – Avant/Après application du filtre de Kuwahara

2.5 Binarisation

L'étape d'après la réduction de bruit est la binarisation. Cela signifie passer d'une image en nuance de gris à une image en noir et blanc. Cette étape est très utile car elle permet de réduire encore plus l'information, et simplifie grandement les algorithmes d'après. Pour ce faire, nous avons utilisé une méthode de seuillage. Un seuillage, comme son nom l'indique, consiste à définir une certaine valeur s en tant que seuil. Tout pixel dépassant ce seuil deviendra un pixel blanc, tandis que tout les autres pixels deviendront noir. La méthode la plus basique consiste à définir un s constant pour toute l'image, il s'agit du *seuillage global* mais cette technique basique est peu efficace pour des images ayant des variances de luminosités. C'est pour cette raison que nous avons choisi la technique du *seuillage adaptatif*.

Le seuillage adaptatif fait varier le seuil s tel que s est égal à la moyenne des valeurs des pixels aux alentours, moins une constante C . Ainsi, les pixels plus sombres que leurs voisins seront mis en arrière-plan tandis que les pixels plus clairs ou de même intensité seront au premier plan.

Nous pouvons cependant remarquer que les résultats ne sont pas idéaux. Il est ainsi prévu de changer l'algorithme pour une autre méthode censée apporté des résultats plus convaincants.

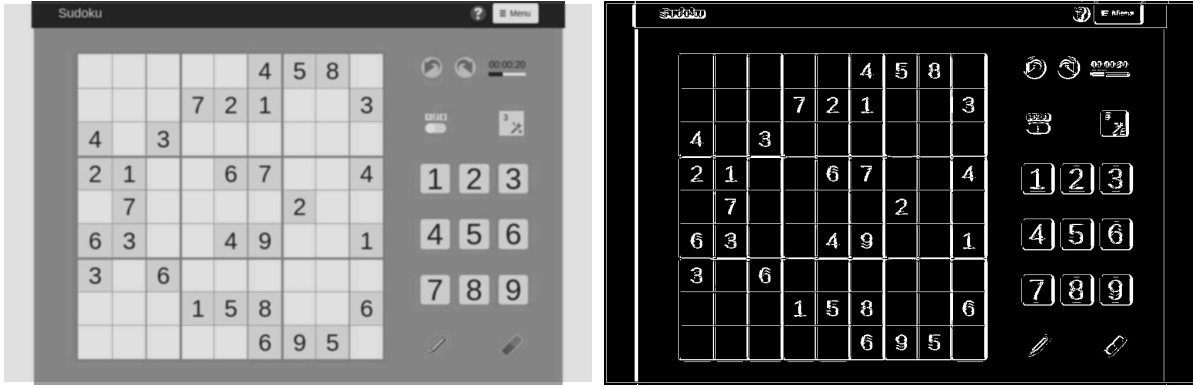


FIGURE 5 – Avant/Après application du seuillage adaptatif

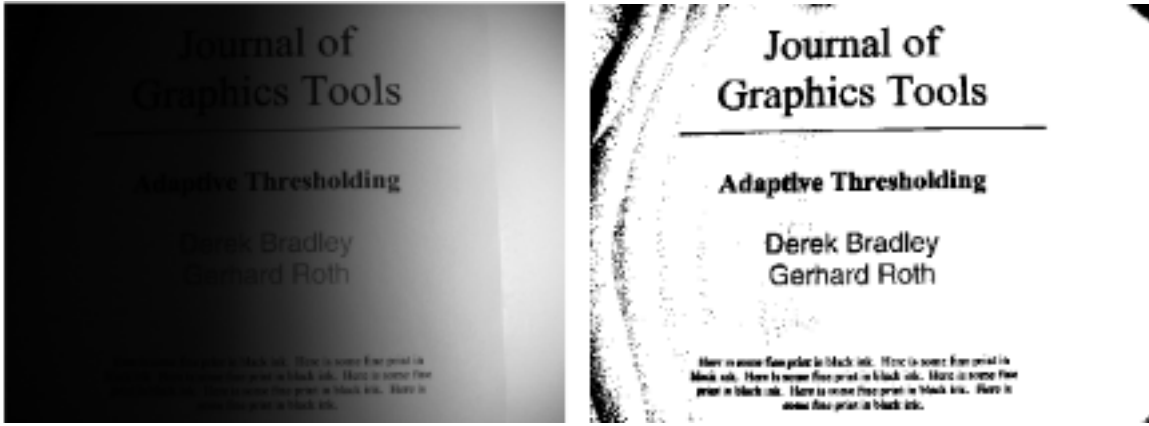


FIGURE 6 – Résultats attendus d'un algorithme plus performants

2.6 Détection des lignes

Bien que non nécessaire dans l'immédiat, la détection des lignes est une étape cruciale pour la suite du traitement d'image. Elle permettra de détecter le plus gros polygone à quatre côtés, qui sera notre grille de sudoku (à noter qu'il ne s'agira pas forcément d'un carré bien orienté, mais peut-être d'une grille penchée donc non-carrée). Pour détecter les lignes, nous utilisons la technique de la *transformée de Hough*.

Une droite peut être écrite sous forme paramétrique : $y = ax + b$, et chaque point de coordonnées blanc a des coordonnées (x, y) . Le principe de la transformée est sous forme de vote, chaque point blanc a donc certains a et b qui valident l'équation $y = ax + b$, ce point-là va donc voter pour chacun des ses points. Et ainsi de suite pour chaque points blancs. Il est ensuite possible de récupérer les couples (a, b) ayant récupéré le plus de votes. Ce seront les lignes de notre image. La seule différence avec l'application pratique de l'algorithme de Hough est qu'il ne s'agit pas de couple (a, b) , tel que $y = ax + b$ mais de couple (ρ, θ) , tel que $\rho = x \cos(\theta) + y \sin(\theta)$. En effet, ce changement est nécessaire car il est impossible de représenter des droites horizontales avec le couple (a, b) , cela aurait demandé $a = +\infty$ et c'est impossible.

Ci-dessous, un exemple de l'espace des couples (ρ, θ) . Plus un point est blanc, plus il a reçu de vote.

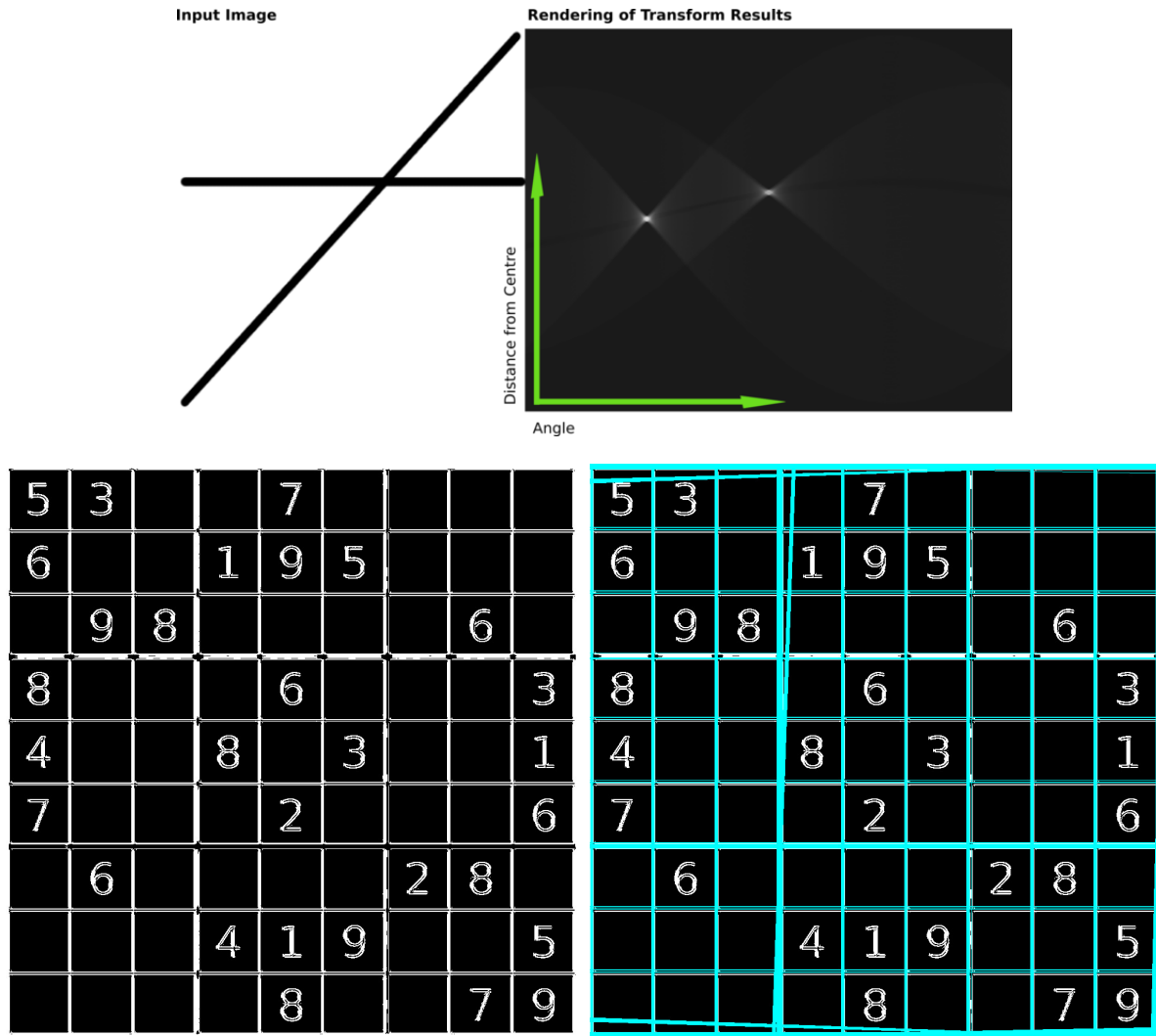


FIGURE 7 – Avant/Après application de la transformée de Hough

2.7 Rotation d'image

Il existe plusieurs méthodes de rotation d'image. Nous avons opté pour la plus simple et efficace. Nous avons tout d'abord implémenté une méthode simple qui consiste à faire la rotation avec une seule matrice :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Le problème étant que nous avons des pertes de qualités de l'image, car certains pixels n'existaient plus. Ils étaient hors de l'image ce qui est dû à une simplification et cela donnais donc une imprécision de la formule.

Nous avons utilisé un algorithme de Shearing qui consiste à prendre chaque pixel de l'image puis appliquer 3 multiplications successives, par des matrices avec θ notre angle en radian :

$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \begin{pmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

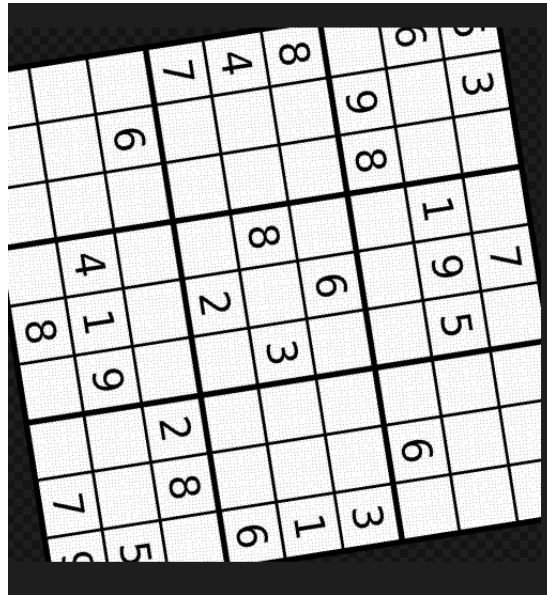
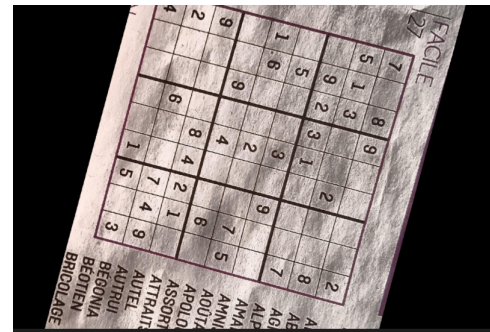
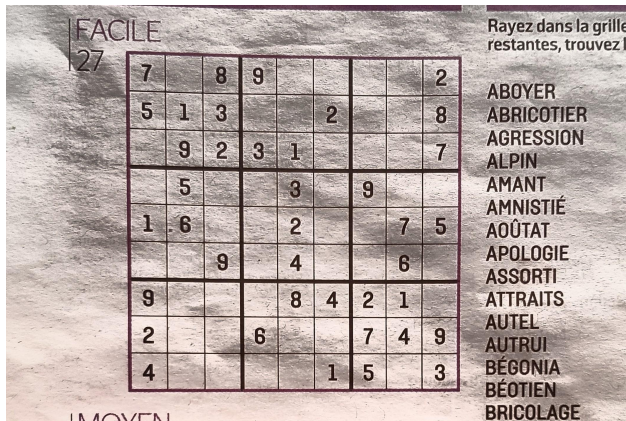


FIGURE 8 – Rotation de 45°

Après avoir calculé nos nouvelles coordonnées, ils nous faut leurs ajouter un décalage pour pouvoir replacer bien l'image en son centre.



La perte de qualité après rotation est minime cependant, il y a un peu de crênelage sur l'image. Mais cela n'empêche en rien la détection de ligne. Ce qui est bien mieux que la première fonction de rotation ou nous avions de la perte qui pouvait occasionner une gêne dans la détection des lignes.

3 Interface Graphique

L'interface graphique n'était pas demandée pour la première soutenance, néanmoins nous avons décidé de la commencer dès le départ pour permettre une évolution de l'interface graphique entre les deux soutenances.

Nous avons choisi d'utiliser la librairie GTK3+, qui fait partie des librairies disponibles pour ce projet. Cette librairie permet de manipuler plus facilement les différents éléments graphiques, que de concevoir l'interface graphique via la "SDL", une librairie utilisée, entre autres, pour le pré-traitement des images.

Afin de concevoir une interface graphique plus ergonomique et plus simplement, nous avons utilisé Glade, un outil interactif de conception d'interfaces graphiques qui utilise la librairie GTK+

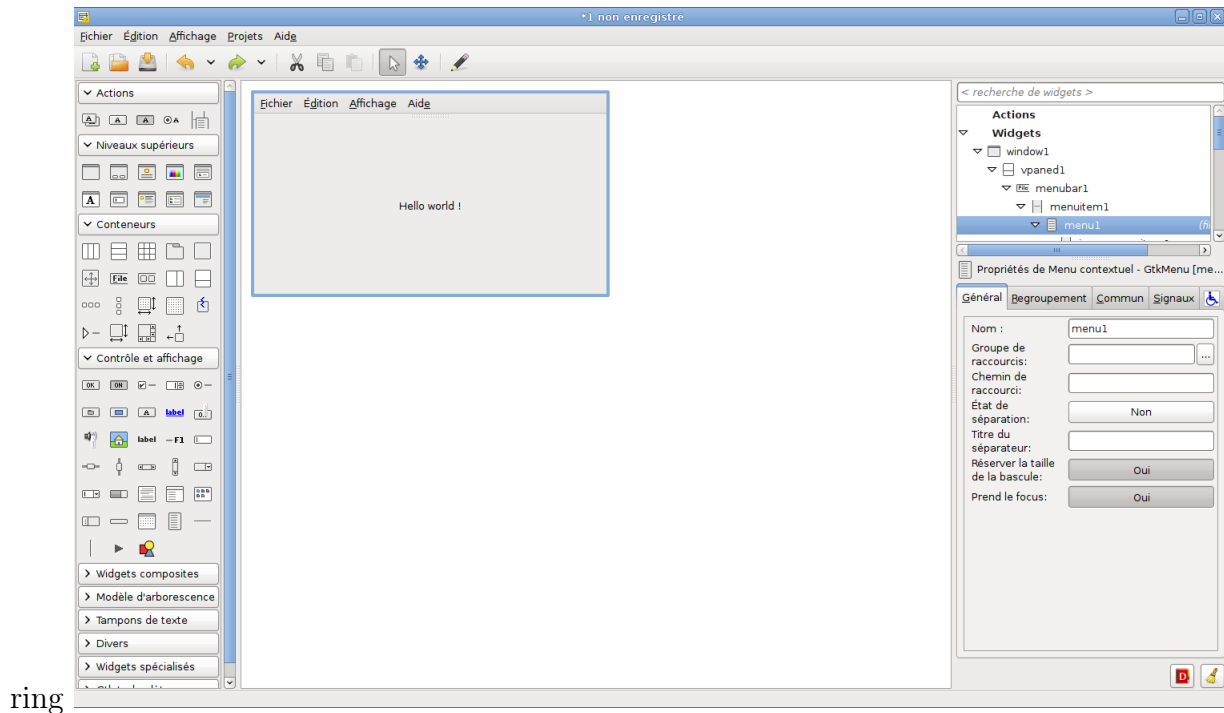
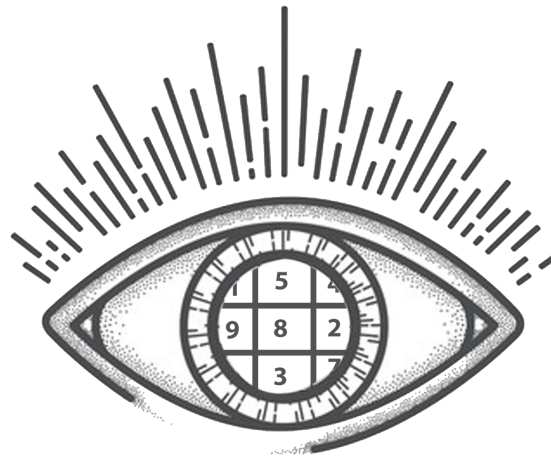


FIGURE 9 – Interface de Glade

3.1 Identité graphique

Pour l'identité graphique de ce projet, contrairement au projet précédent, nous avons décidé de choisir des couleurs sombres.

Notre logo combine à la fois un œil, qui représente l'OCR, ainsi qu'une grille de sudoku, rappelant le but de ce logiciel à l'utilisateur.



SUDO.C.R

FIGURE 10 – Logo de SUDO.C.R

3.2 Interface utilisateur - Utilisation

Aujourd'hui notre interface graphique permet à l'utilisateur de charger une image depuis son ordinateur au format PNG. Une fois l'image chargée, l'utilisateur a la possibilité d'appliquer plusieurs filtres à l'image, les filtres disponibles pour le moment sont : Le niveau de gris, le flou gaussien, la binarisation ainsi que le filtre de Kuwahara.

Notre interface graphique se présente actuellement sous cette forme.

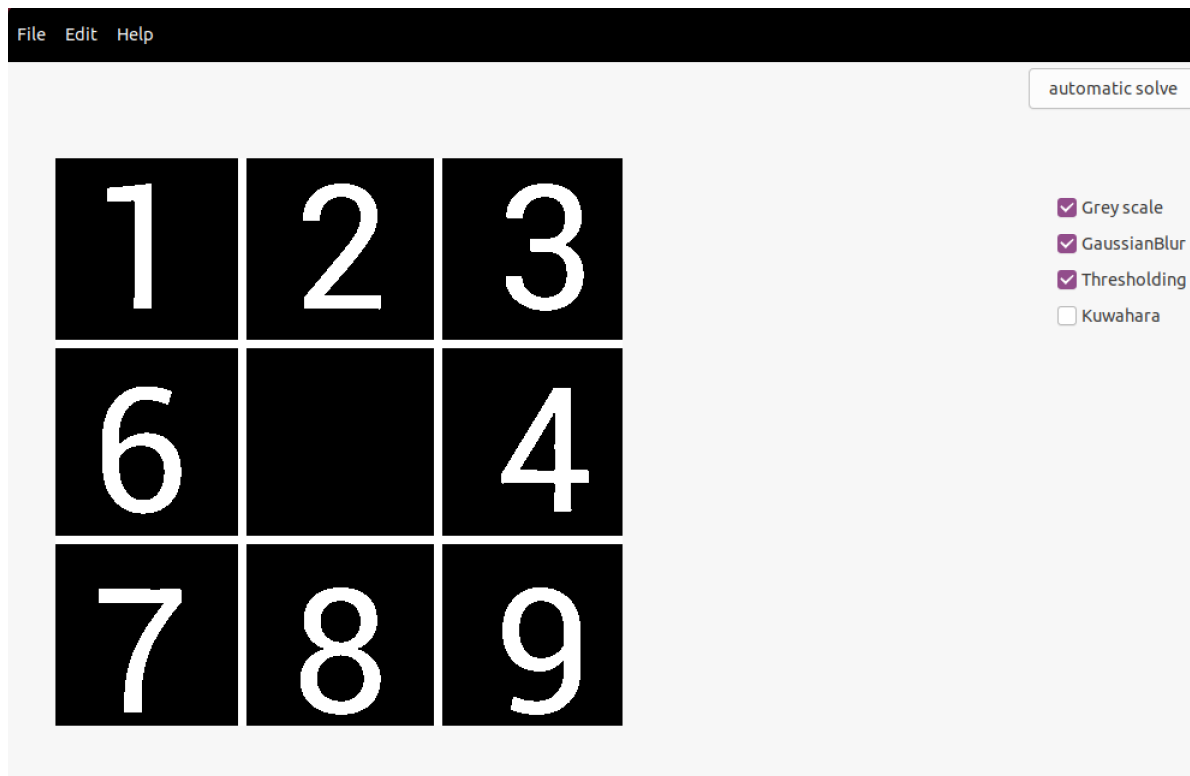


FIGURE 11 – Interface SUDO.C.R

3.3 Interface utilisateur - Artistique

Notre interface graphique à terme ressemblera à cela : Le thème sombre permettra à

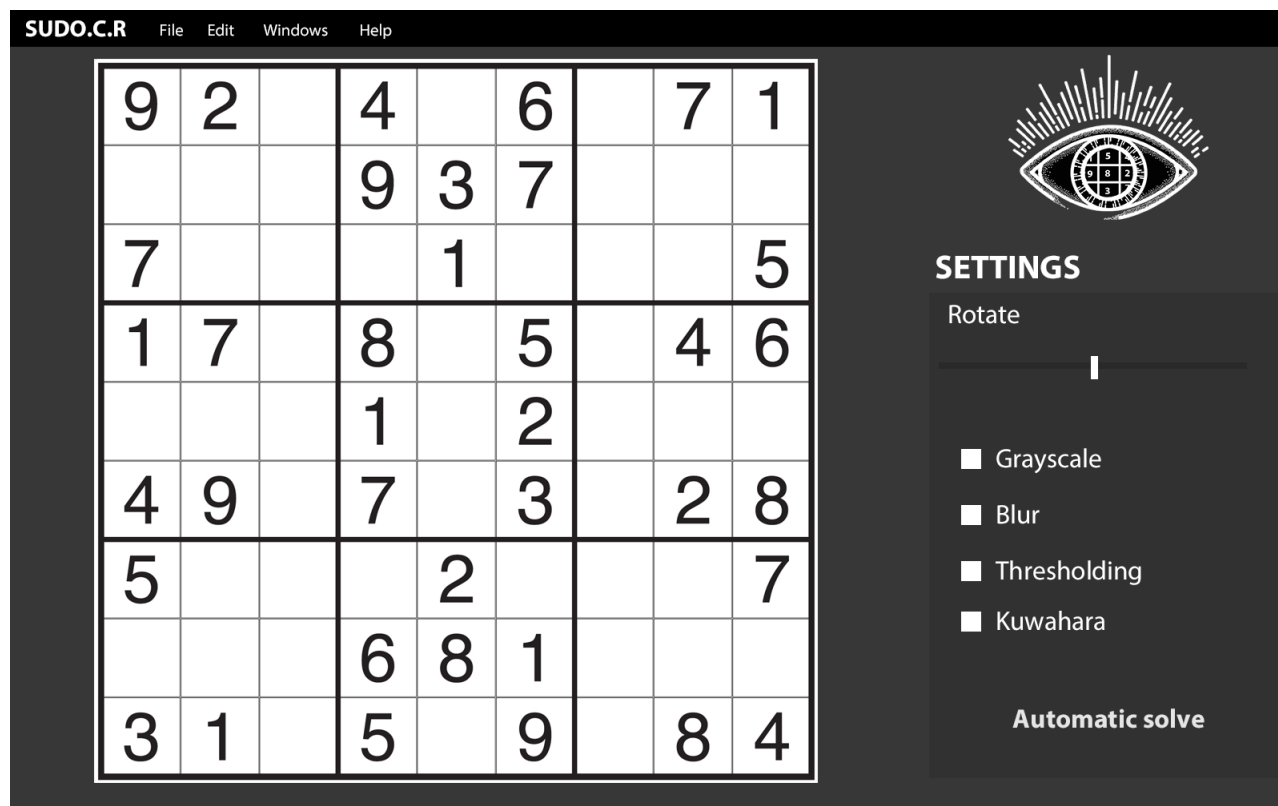


FIGURE 12 – Maquette SUDO.C.R

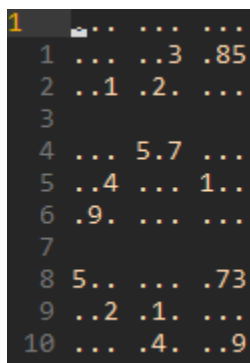
l'utilisateur de pouvoir utiliser le logiciel quand il le souhaite sans que les couleurs "ne l'agressent".

De plus, l'interface graphique se devra d'être ergonomique, afin de permettre une utilisation facile, à tout utilisateur, quelles que soient ses compétences en informatique.

4 Le solver de Sudoku

4.1 Le principe de fonctionnement

Le but du solver est de résoudre le Sudoku, ce dernier est un exécutable qui va prendre 1 argument le chemin d'un fichier. Dans ce fichier, on va y retrouver un sudoku divisé en plusieurs parties :



```
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
2 1 1 1 1 1 1 1 1 1
3 1 1 1 1 1 1 1 1 1
4 1 1 1 1 1 1 1 1 1
5 1 1 1 1 1 1 1 1 1
6 1 1 1 1 1 1 1 1 1
7 1 1 1 1 1 1 1 1 1
8 1 1 1 1 1 1 1 1 1
9 1 1 1 1 1 1 1 1 1
10 1 1 1 1 1 1 1 1 1
```

FIGURE 13 – Fichier contenant le sudoku

Dans ce fichier, les points représentent les 0, chaque bloc représentent les différents blocs de 3 colonnes.

Le programme va lire le fichier en ignorant les espaces et les retours à la ligne en recopiant chaque caractère dans un tableau.

Une fois la chaîne de caractère entièrement recopié, nous allons vérifier qu'il s'agit bien d'un sudoku valide, s'il n'est pas valide alors le programme renverra une exception, et seulement là nous alors pouvoir le résoudre.

Pour ce faire, nous avons utilisé un algorithme de "backtracking". Notre algorithme va tester une possibilité pour une case puis passer à la suivante si jamais cette possibilité était fausse alors il reviendra sur cette case puis testera une autre possibilité jusqu'à la résolution complète du sudoku.

Une fois résolu, il va l'enregistrer suivant le même format que le fichier d'origine.

5 Réseau de neurone

Nous avons développé un réseau de neurones artificiels. Nous lui avons fait apprendre la fonction ou exclusif aussi appelé XOR, afin de démontrer le bon fonctionnement de notre réseau.

Entrée 1	Entrée 2	Sortie
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 14 – Table de vérité du xor

Le modèle que nous avons utilisé est le perceptron multi-couches, c'est un système qui apprend par l'expérience. Son objectif est d'apprendre à classier les données que l'on lui donne en entrée. Pour cela chaque neurone va créer une courbe pour séparer les données.

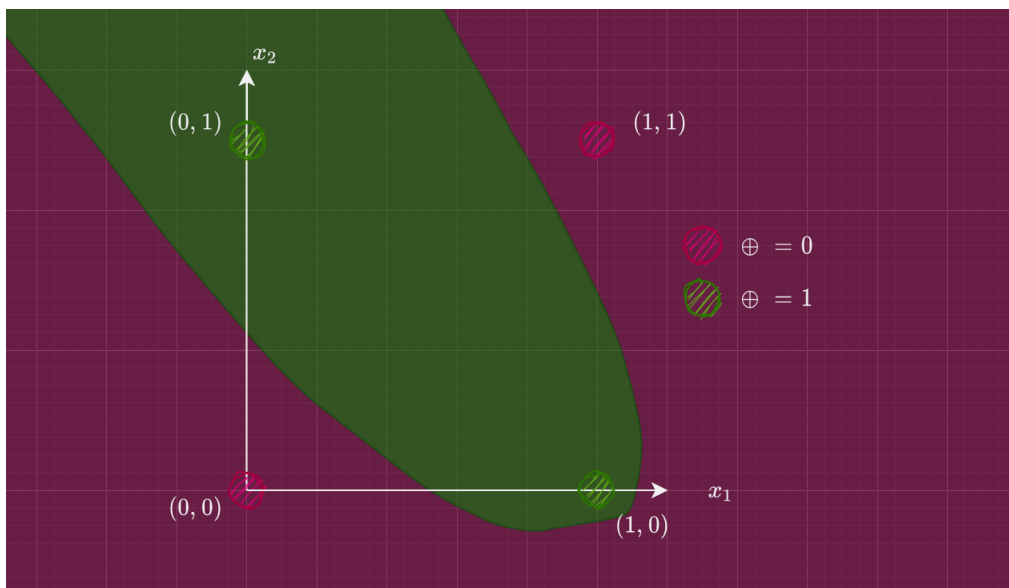


FIGURE 15 – Graphique du tri des données pour le réseau XOR

5.1 Le principe de fonctionnement

Un perceptron multi-couches est un ensemble de neurones organisé en plusieurs couches. On trouve une couche d'entrée, une couche cachée et une couche de sortie. Les neurones entre chaque couches sont reliés avec des liens pondérés, les valeurs des liens pondérés sont appelées les poids. On peut résumer le fonctionnement ainsi : on entre des valeurs puis on propage ces valeurs au travers de chacune des couches. Au final, dans la dernière couche on obtient la probabilité d'avoir un 1 pour le résultat de l'opération XOR par exemple. La couche d'entrée et la couche cachée possèdent des matrices de poids et de biais. Les biais sont des valeurs qui permettent de décaler vers la gauche ou la droite les courbes que dessine chaque neurone.

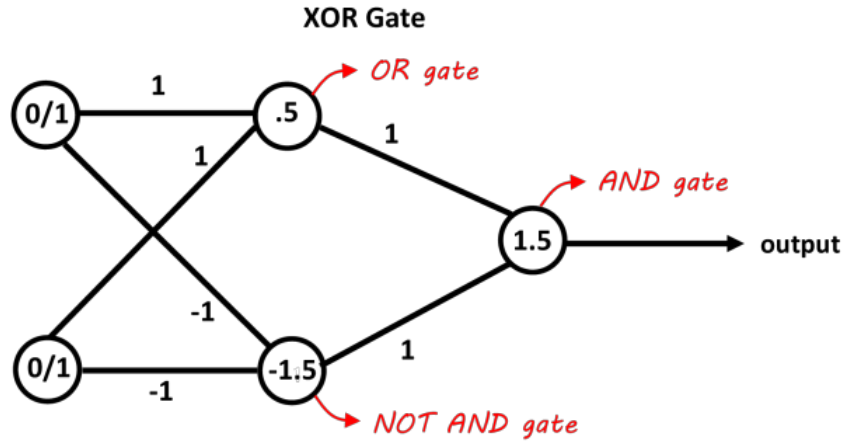


FIGURE 16 – Réseaux de neurones pour le xor

5.2 Le principe d'entraînement du réseaux de neurone

Nous allons détailler le fonctionnement de notre réseaux. Dans un premier temps, il va falloir entraîner le réseau de neurones. La procédure d'entraînement se divise en 4 étapes.

D'abord, on initialise le réseau de neurones avec la loi uniforme pour les poids et on initialise les biais à 0. La loi uniforme est la formule suivante, soit n le nombre de neurone de la couche à laquelle appartiennent les poids, tel que les poids appartiennent à l'intervalle suivant $[-y, y]$ avec $y = \frac{1}{\sqrt{n}}$.

Ensuite, on propage les valeurs de la couche d'entrée. Pour calculer les valeurs d'une couche à une autre on utilise la formule suivante :

$$Y^l = \sum_{i=1}^n \sigma(x_i^{l-1} * w_i^l - \theta^l)$$

Ici l représente l'indice de la couche, n représente le nombre de neurone de la couche précédente et x_i est la valeur du neurone i de la couche précédente. Le poids du lien est w_i , le biais est θ et enfin σ est la fonction d'activation.

La fonction d'activation que nous utilisons est la fonction sigmoïde, elle permet d'obtenir

une valeur entre 0 et 1, peu importe la valeur d'entrée. Il suffit de multiplier la matrice des valeurs des neurones de la couche précédentes par la matrice des poids et d'ajouter la matrice des biais. Enfin on active le résultat avec la fonction sigmoïde.

La troisième étape consiste à appliquer la *backpropagation*. Dans un premier temps, on calcule l'erreur de la couche de sortie, puis on la propage en arrière aux autres couches. La formule de l'erreur est la soustraction des valeurs attendues aux valeurs de la couche de sortie. Ensuite grâce à cette erreur, on peut calculer la correction des poids et des biais aussi appelées le gradient noté Δ .

Ainsi pour le calcul du gradient d'un poids on trouve la formule suivante : $\Delta w_i^l = x_i^l * error^l$. La formule du gradient des biais est : $\Delta \theta = \alpha \delta$

Enfin, on applique un algorithme appelé *gradient descent*. Grâce au gradient on peut mettre à jour les poids et les biais. Mais pour éviter que le résultat des neurones oscillent, en corrigeant les poids avec des trop grandes valeurs, on ajoute une valeur. Un pas noté α . On obtient la formule suivante : $w_i^{l+1} = \alpha * w_i^l + \Delta w_i^l$

Une fois le réseau de neurones entraînés il suffit d'entrer des valeurs, de les propager et de regarder la valeur des neurones de sorties.

6 Conclusion

6.1 Bilan de la première soutenance

En bilan de notre première soutenance, nous avons réalisé certaines des fonctionnalités de notre logiciel, comme :

- L'interface graphique de notre projet, il est actuellement capable de charger une image, appliquer des filtres et sauvegarder l'image.
- La partie traitement d'image (Binarisation et découpage de l'image)
- Un réseau de neurone capable de calculer un "XOR".

6.2 Objectifs seconde soutenance

Pour la seconde soutenance nous devons réaliser certains objectifs afin de finaliser notre projet, comme :

- Finir l'interface graphique afin de permettre une utilisation ergonomique et complète, pour l'utilisateur. Ainsi, que d'ajouter différentes options.
- Finir le réseau de neurone, pour qu'il soit capable de reconnaître les caractères.
- Rassembler toutes les parties de ce projet afin d'avoir un projet complet