Rapport de Soutenance 1 - Projet S3 EPITA
4D Generates

4D Generates
Gustave HERVÉ, Léo SAMBROOK, Arthur HAMARD et Briac GUELLEC

Novembre 2022

Table des matières

1 Introduction **3**
1.1 Notre Groupe 3

2 Répartition des Charges **3**

3 État d'Avancement du Projet **3**

4 Aspects Techniques **3**
4.1 Traitement d'image 3
4.1.1 Rotation d'image 3
4.1.2 Binarisation de l'image 5
4.2 Détection de la grille 9
4.2.1 Filtres morphologiques 9
4.2.2 Détection des lignes 10
4.2.3 Extraction des cases 11
4.3 Reseau de Neurones 11
4.3.1 XOr 11
4.3.2 Reconnaissance de Chiffres 14
4.4 Solveur de Sudoku 14
4.5 Interface Graphique 16

5 Conclusion **18**

1 Introduction

1.1 Notre Groupe

Notre groupe est constitué de : Gustave HERVÉ, Léo SAMBROOK, Arthur HAMARD et Briac GUELLEC. Nous sommes heureux, dans ce rapport, de vous présenter notre projet de S3 consistant en un OCR.

2 Répartition des Charges

Tâches	Responsable
Neural Network + Solver	Léo
Traitement image + Détection de grille	Gustave
Rotation	Briac
Interface + Sauvegarde résultat	Arthur

3 État d'Avancement du Projet

4 Aspects Techniques

Afin de concevoir cet OCR, nous avons dû nous intéresser à différents concepts qui nous ont permis d'avancer sur le projet.

4.1 Traitement d'image

Le traitement de l'image du sudoku à résoudre est un point crucial du projet. Ce dernier va clarifier l'image et délimiter les cases du sudoku, permettant par la suite de traiter les parties dans le réseau neuronal. Pour cette partie, nous avons utilisé la librairie SDL qui a pour avantage d'offrir des outils de manipulation d'image intégrés.

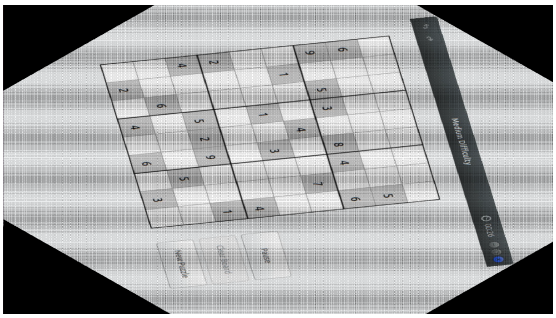
4.1.1 Rotation d'image

Pour réaliser la rotation de l'image la première technique qui a été implémentée est l'utilisation d'une matrice de rotation. Le premier script récupérait les coordonnées de chacun des pixels et en multipliant matrice de rotation et matrice des coordonnées du pixel on en obtenait une nouvelle matrice contenant les coordonnées du pixel dans la nouvelle image après rotation.

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

(a) Calcul des coordonnées d'un pixel après rotation

Cette méthode permet bel et bien d'effectuer une rotation de l'angle souhaité mais certains problèmes sont apparus lors des premières implémentations comme peut en témoigner l'image ci-dessous.



(a) image truffée d'aliasing

Le problème principal rencontré lors de la première implémentation de la rotation avec cette méthode est l'aliasing ou autrement dit les pixels noirs qui parsèment l'image et la rendent complètement inexploitable. Pour faire face au problème de l'aliasing nous avons implémenté la *Three shear rotation method* qui consiste à "découper" la matrice de rotation qui comporte des sinus et des cosinus en trois matrices différentes n'en comportant plus.

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

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

(a) Calcul des coordonnées en utilisant la méthode Three shear

Cela supprime bel et bien l'aliasing mais en faisant apparaître une fonction tangente la rotation d'images avec des angles multiples de 90 est compromise. En effet,

la tangente n'étant nul autre que le quotient de sinus par cosinus, calculer la tangente d'un angle de 90 degrés revient à divisier par cos(90) soit 0 ceci résultant en une rotation étrange. Finalement, les résultats n'étant pas satisfaisants pour les rotations proches de multiples de 90, nous nous sommes retournés vers la matrice de rotation vue précédemment.

En repartant de zéro et en supprimant un script qui faisait en sorte que l'image prenne l'entièreté de la fenêtre qu'importe la taille de cette dernière, la rotation fut enfin opérationnelle.

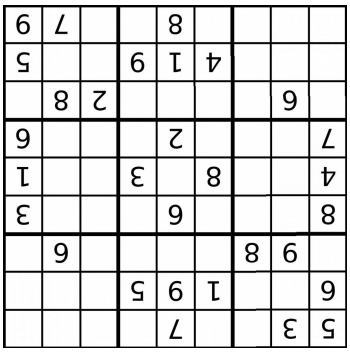


FIGURE 4 – Image après Rotation de 180°

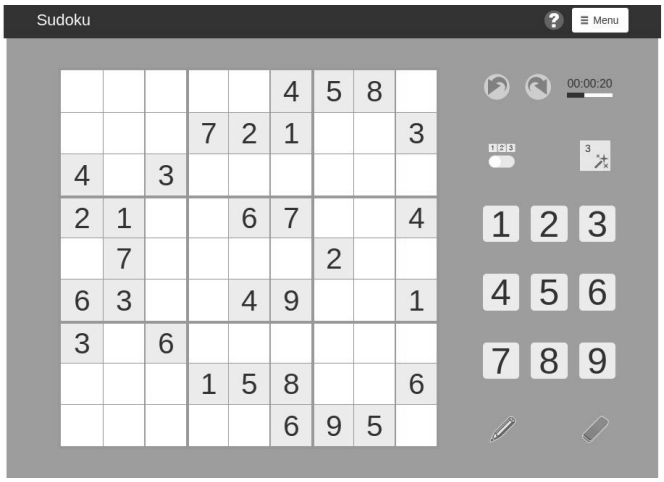
4.1.2 Binarisation de l'image

Pour faciliter la manipulation de l'image, il est nécessaire de réduire au maximum les informations de couleur pour obtenir une image binaire. L'objectif de cette étape est d'obtenir une image avec des pixels totalement blancs (1) ou totalement noirs (0).

Conversion en niveaux de gris et floutage :
Les informations RGB n'étant pas utiles pour ce projet, la première étape consiste à retirer les couleurs pour obtenir une image en niveaux de gris. Pour ce faire, nous parcourons chaque pixel de l'image et utilisons l'équation suivante pour calculer l'intensité en niveau de gris du pixel :

$$Intensity = 0.3 * R + 0.59 * G + 0.11 * B$$

avec RGB les intensités de couleur respectivement rouge, vert et bleu.



(a) Passage en niveaux de gris

Pour éviter du bruit pouvant affecter négativement la binarisation de l'image, nous appliquons également par convolution de matrice un flou gaussien.

Détection de contours

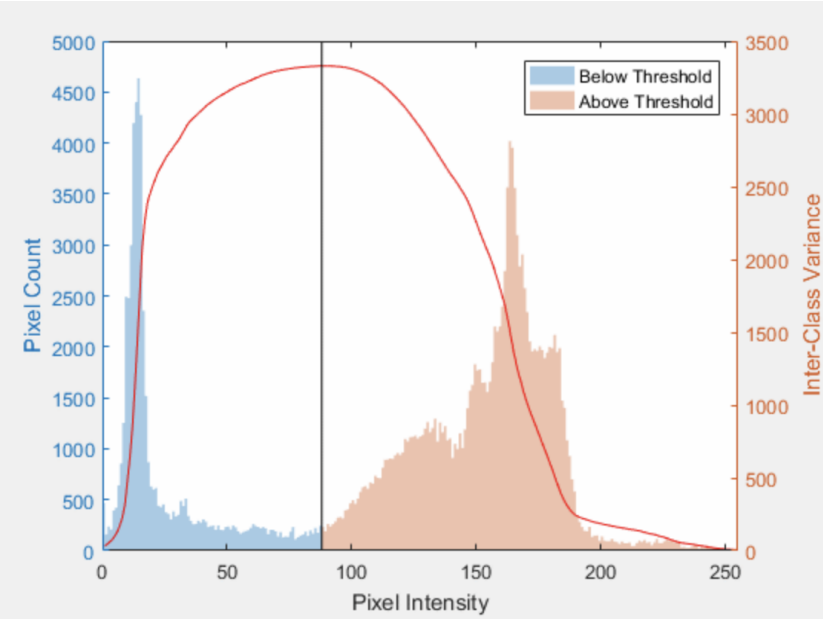
Suite à cela nous pouvons appliquer en premier lieu une détection de contours par filtres de Sobel. Ces derniers fonctionnent par convolution des matrices suivantes correspondant respectivement à l'axe X et l'axe Y :

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & -0 \\ 1 & 2 & 1 \end{bmatrix}$$

Une fois les informations sur les deux axes récupérées, nous pouvons calculer les valeurs de gradient telles que pour chaque pixel :

$$Gradient = \sqrt{Gx^2 + Gy^2}$$

La prochaine étape consiste à isoler les "vrais" contours de l'image en appliquant une étape de seuillage à chaque pixel. Nous définissons un ratio "haut" en utilisant le résultat renvoyé par la méthode d'Otsu sur l'image. Cette dernière consiste en la construction d'un histogramme des valeurs d'intensité des pixels de 0 à 255, puis à calculer la variance inter-classe maximum parmi les 256 valeurs de gris possibles.



(a) La méthode d'Otsu permet de trouver la valeur d'intensité située sur la ligne noire

Une fois cette valeur "haute" trouvée, nous définissons la valeur "basse" tel que

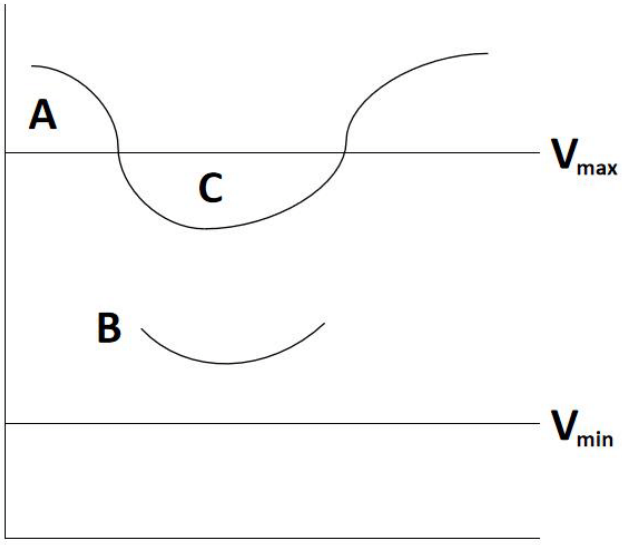
$$Low = 0.5 * High$$

Une fois nos deux valeurs définies, nous parcourons chaque pixel de notre image filtrée par Sobel précédemment puis appliquons sur le pixel une opération selon sa valeur d'intensité :

— **La valeur d'intensité du pixel est supérieure à notre seuil "haut"**
Le pixel est directement considéré comme étant un "vrai" contour et est gardé dans l'image finale. On classe le pixel comme étant "Fort" et on fixe sa valeur d'intensité à la valeur maximale, soit 255. Le pixel devient autrement dit parfaitement blanc. On l'empile également dans une pile dédiée pour pouvoir y accéder plus tard.

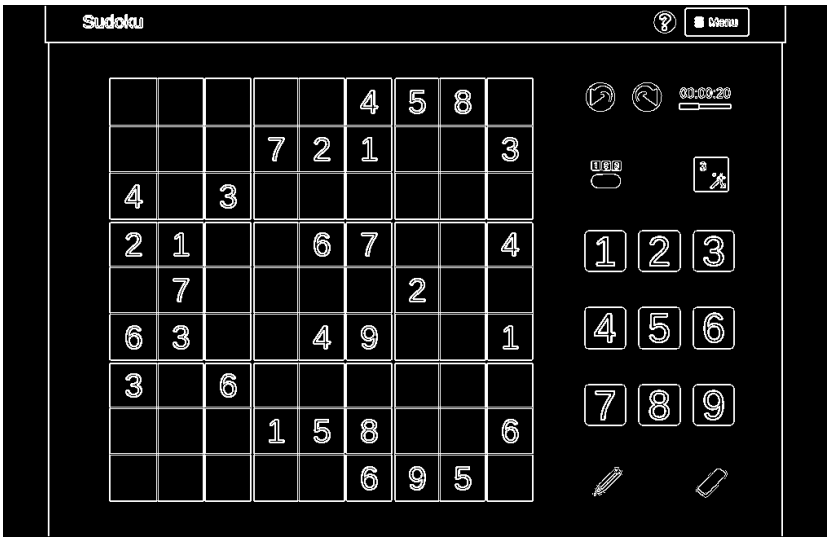
— **La valeur d'intensité du pixel est située entre le seuil "bas" et le seuil "haut"**
Dans ce cas, nous ne pouvons pas tout de suite déterminer si le pixel doit être éliminé ou gardé pour la suite. Nous l'enregistrons en pixel "faible" en lui appliquant une valeur d'intensité à 100.

— **La valeur d'intensité du pixel est en dessous du seuil "bas"**
L'intensité du pixel est mise à 0 et n'est pas considéré comme un contour.



(a) Illustration de l'étape du double seuillage. Ici, le contour A est fort, B et C faibles

Pour savoir si les pixels considérés "faibles" sont de vrais contours, nous devons savoir si ceux-ci sont reliés par un pixel "fort". Nous partons de chaque pixel "fort" à partir de la pile de stockage citée précédemment, puis nous faisons une recherche récursive sur les 8 pixels voisins. Pour chacun de ces pixels nous vérifions s'il s'agit d'un pixel faible. Si oui, celui-ci est relié à un pixel fort et nous pouvons alors le définir fort également. Si le pixel n'est pas faible, alors il s'agit ou bien d'un pixel fort ou bien d'un pixel éliminé. Dans ces deux cas il n'y a aucune opération à réaliser pour l'appel récursif en question. Enfin, la dernière étape consiste à nettoyer l'image en retirant tous les pixels faibles n'ayant pas été retenus par les étapes précédentes, ces-derniers sont alors éliminés.

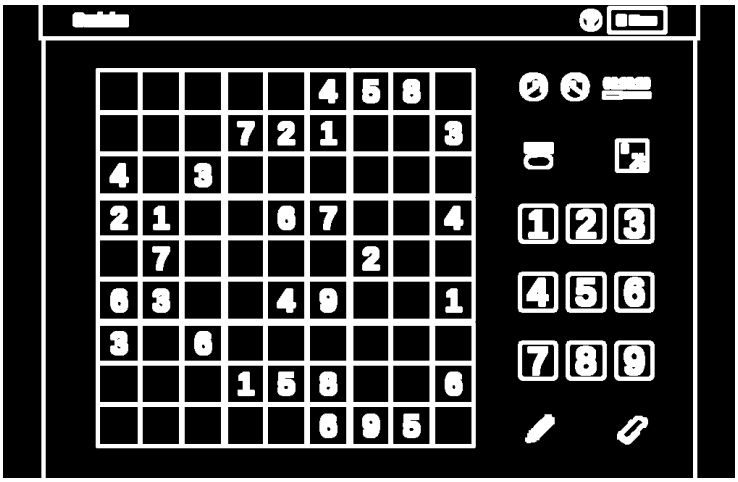


(a) Résultat final de la binarisation

4.2 Détection de la grille

4.2.1 Filtres morphologiques

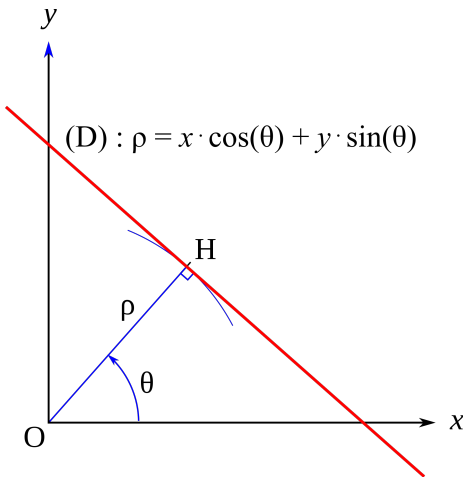
Avant d'appliquer notre détection de lignes, nous appliquons des opération de dilatation afin d'exagérer les lignes et faciliter les étapes suivantes. Pour chaque pixel de l'image, nous remplaçons le pixel actuel par le voisin ayant la plus grande intensité.



(a) Résultat de la dilatation sur l'image précédente

4.2.2 Détection des lignes

Pour détecter les lignes présentes dans l'image binarisée, nous utilisons la transformée de Hough. Cet algorithme utilise les coordonnées polaires pour identifier les lignes en fonction de deux paramètres : leur distance par rapport à l'origine de l'image (le paramètre ρ) et l'angle qu'elles forment avec l'axe X (le paramètre θ).



(a) La transformée de Hough utilise une réprésentation polaire

Pour déterminer les lignes d'une image, une matrice H de taille $\rho * \theta$ est formée et intialisée à 0 pour chacune de ses valeurs. Pour chacun des pixels formant un contour de l'image (pixel blanc), on calcule pour θ allant de 0 à 360 le paramètre ρ tel que pour un pixel de coordonnées (x,y) :

$$\rho = x * \cos\theta + y * \sin theta$$

on incrémente alors la valeur de H[ρ][θ].

Une fois la matrice remplie, nous allons chercher les indices ρ et θ des maximums locaux, c'est-à-dire les points avec les valeurs les plus hautes. Pour ce faire, nous calculons une valeur seuil T :

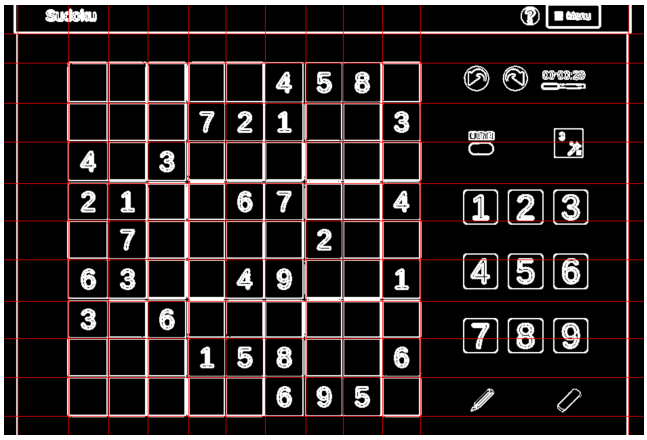
$$T = 0.5 * Max$$

avec Max la plus grande valeur contenue dans la matrice H.

Chaque paire d'indices récoltés représente alors une droite. Durant cette étape, nous ne retenons que les lignes horizontales et verticales, les diagonales sont ainsi

filtrées hors de la recherche en fonction de la valeur de θ. Nous appliquons également un algorithme de fusion des lignes similaires pour éviter des doublons.

Pour trouver les coordonnées des segments se situant sur les droites trouvéees, nous parcourons ces-dernières à l'aide de leur équation polaire puis nous enregistrons les coordonnées du plus grand segment trouvé sur cette équation.



(a) Les lignes rouges sont détéctés par la transformée de Hough

4.2.3 Extraction des cases

A partir de cette étape, nous avons une liste de segments verticaux et une liste de segments horizontaux. Nous allons chercher à trouver 10 segments horizontaux ayant en commun avec 10 autres segments verticaux un point d'intersection et inversement pour les segments verticaux. Nous allons ainsi extraire les segments constituant la grille.

Une fois cela fait, nous pouvons calculer la taille du carré de la grille et faire une division par 9 pour avoir les emplacements approximatifs des coins des 81 carrés constituant le Sudoku. Il ne reste plus qu'à enregistrer chacun de ces carrés dans une image.

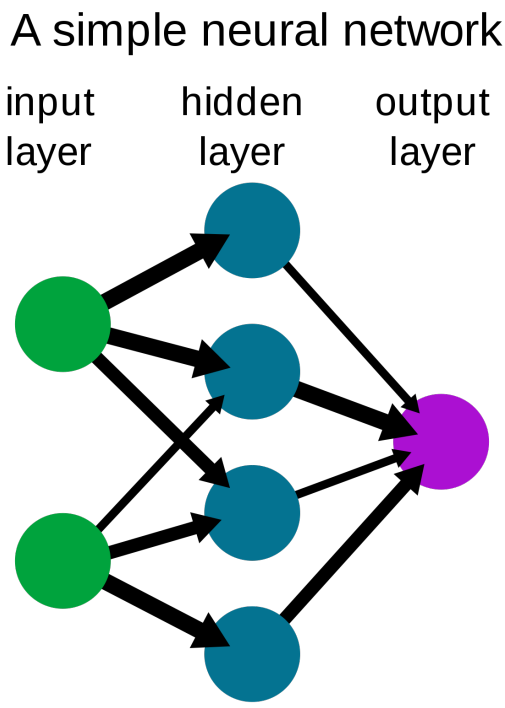
4.3 Reseau de Neurones

4.3.1 XOr

L'architecture :

Notre réseau de neurones XOR est composé de 3 couches. La première, la couche d'entrée ou noeuds d'entrée (Input nodes en anglais) est composée de 2 noeuds car dans l'opérateur logique OU EXCLUSIF, il y a deux entrées ([00],[01],[10],[11]). La couche intermédiaire (2ème couche, hidden nodes en anglais) quant à elle est composée de 4 noeuds (Plus de précision et rapidité que 2). Et pour finir la dernière couche (output nodes en anglais) n'est composée que d'un seul noeud car il n'y a qu'un seul résultat (soit 1 soit 0). Nous appliquons ensuite la propagation vers l'avant en partant des deux premiers neurones appartenant à la couche d'entrée. Et nous appliquons la rétropropagation en partant de la dernière couche.

Notre réseau de neurones peut être représenté de la manière suivante :



(a) Schéma de notre réseau de neurones

Comme il est indiqué sur le schéma, chaque noeud est relié par des "câbles" appelés "arêtes" et sur ces "arêtes" sont attribués un poids qui va multiplier les valeurs transmises d'un noeud à un autre. "Lintelligence" d'un réseau de neuronne est basé sur l'apprentissage et ce sont ces "poids" qui vont permettre à un réseau de

neurones de donner des bons résultats. Il est donc important de les attribuer et de les mettre à jour correctement. C'est pour cela que nous utilisons deux méthodes : la propagation vers l'avant et la rétropropagation.

Propagation vers l'avant :
La propagation vers l'avant consiste à partir des neurones de la couche d'entrée pour aller jusqu'à/aux couche(s) de sortie tout en passant par chaque neurone de chaque couche dans le but de mettre à jour leur poids.

Lorsqu'un un neurone reçoit une valeur, il va pouvoir la modifier avec une fonction mathématique avant de l'envoyer vers son successeur. Cette fonction mathématique s'appelle la fonction "d'activation" et son but est de déterminer si oui ou non, il doit transmettre la valeur reçue au prochain. Si la valeur passe au prochain alors elle est proche de 1, sinon elle est proche de 0.

Chaque neurone a la possibilité d'ajouter à l'entrée de la fonction d'activation un "biais" qui est une petite valeur multipliée par le poids. Ces biais permettent aux neurones d'avoir une influence sur la fonction d'activation. Il existe plusieurs fonctions d'activations, celle que nous utilisons s'appelle la sigmoïd :

$$S(z) = 1/(1 + exp(-z))$$

L'atout de cette fonction est qu'elle renvoie des valeurs dans l'intervalle $[0\,;\,1]$ ce qui permet d'interpréter la sortie du neurone comme une probabilité. De plus, comme sa dérivée est assez simple, elle évite de ralentir encore plus notre fonction.

Rétropropagation :
Le but de la "rétropropagation", ou la propagation en partant de la sortie, est de minimiser les erreurs de poids. Pour calculer les erreurs nous utilisons une fonction mathématique très simple qui consiste à soustraire le résultat obtenu par le résultat attendu et multiplier le résultat par la fonction dérivée de la fonction sigmoïd appelée DSigmoïd :

$$S'(z) = S(z) * (1 - S(z))$$

Une fois que la valeur "d'erreur" a été calculée, on multiplie chaque poixs par leur valeur d'erreur.

Sauvegarde de la mémoire :
Nous avons dit plus tôt que l'efficacité et la précision d'un réseau de neurones dépendait des poids et c'est pour cela qu'à la fin de notre réseau de neurones nous sauvegardons les poids de chaque arête dans un fichier. Lorsque nous relançons le réseau de neurones, on ré-attribue les valeurs sauvegardées à chaque poids car sinon le réseau va se réinitialiser et perdre tout son "entraînement". Lorsque le réseau de neurones est lancé pour la première fois, on attribue des valeurs aléatoires aux poids.

4.3.2 Reconnaissance de Chiffres

Nous avons commencé à implémenter le réseau de neurones qui va détecter le chiffre sur une image. Celui-ci est capable de s'entraîner sans problème avec un jeu d'images allant de 0 à 9. La manière dont il s'entraîne est quasi identique à celle du XOR mais comporte tout de même quelques différences. En effet, au début du programme nous convertissons les pixels d'une image de chiffres en 0 et 1 (0 pour pixel noir et 1 pixel blanc) chaque pixel va correspondre à un neurone d'entrée et, comme les images sont de taille 28x28, il y a 784 neurones dans la première couche. De plus, la couche intermédiaire passe de 3 neurones à 20 pour plus de précision. Pour finir, la dernière couche, c'est à dire celle de sortie, possède 10 neurones car il y a 10 chiffres possibles dans un sudoku (0,1,2,3,4,5,6,7,8,9) et le neurone qui possède la plus grande valeur en sortie correspond au chiffre de l'image. Par exemple si c'est le neurone de sortie numéro 3 qui possède la plus grande valeur alors le réseau de neurones en déduit que le chiffre est un 3. La prochaine étape va donc être de pouvoir mettre un dossier possédant toutes les images du sudoku et mettre les résultats dans un fichier pour le solver.

4.4 Solveur de Sudoku

Le solveur de sudoku est la pierre angulaire du projet. Il s'agit du programme qui, à partir d'une matrice de chiffres, va être capable de résoudre un sudoku. Il prendra donc en paramètre un fichier comportant les chiffres du sudoku précédemment

scannés à l'aide du réseau de neurones. Ici, les zéros sont représentés avec des points.

```
$ cat grid_00
... ..4 58.
... 721 ..3
4.3 ... ...

21. .67 ..4
.7. ... 2..
63. .49 ..1

3.6 ... ...
... 158 ..6
... ..6 95.
```

Entrée du programme Solver.

```
$ cat grid_00.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
```

Sortie du Solver

Fonctionnement du Programme
Ce solver, afin de compléter la grille, va utiliser un algorithme de backtracking, ou de retour sur trace. L'algorithme consiste à sélectionner une case à remplir, et d'essayer récursivement de résoudre le sudoku avec le numero sélectionné pour cette case, en appliquant l'algorithme sur les autres cases, prenant en compte les changements précédents. Si il est impossible de le résoudre dans ces conditions, alors le systeme va revenir légèrement en arrirèe sur ses décisions afin de sortir du blocage (d'où le nom de retour sur trace). Il va, par la suite, réessayer sur la case initiale avec de nouveaux paramètres, et sur les autres cases, de manière récursive.

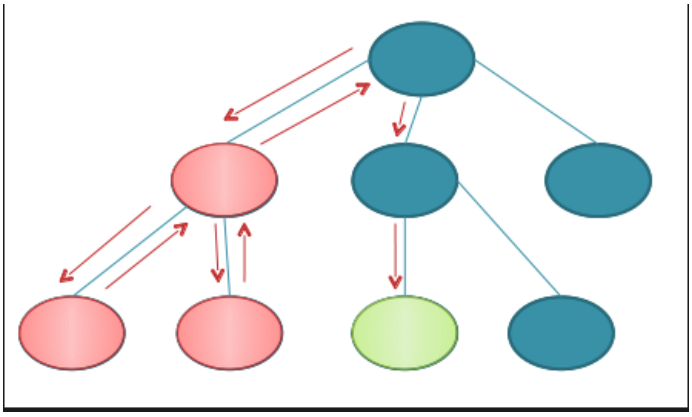


Illustration de l'algorithme de backtracking, les noeuds représentant les différentes possibilités de résolution du sudoku.

4.5 Interface Graphique

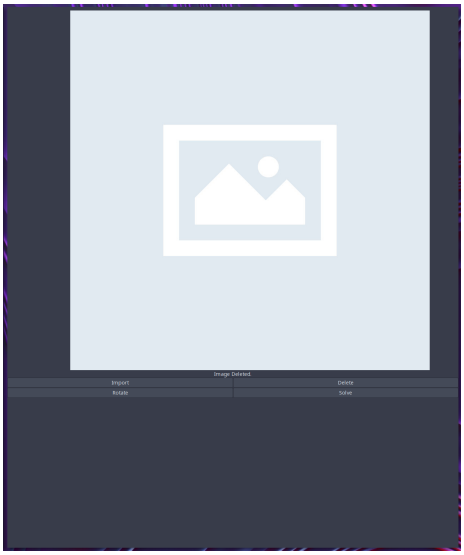
Afin de proposer une agréable expérience à l'utilisateur, nous avons mis en place une interface graphique permettant de manipuler toutes les fonctions de l'application. Cette dernière va permettre de gérer manuellement les aspects techniques vu précédemment tels que la rotation ou la résolution de la grille de sudoku.

Construction du GUI
Pour ce dernier, nous avons utilisé la librairie GTK, qui permet de créer des applications de manière simple en C. De plus, nous avons incorporé une structure créée à l'aide de l'outil Glade, permettant de réaliser le cadrage, les boutons et les labels

(presque) sans lignes de codes. GTK restant néanmoins nécessaire à l'exécution des scripts au contact des boutons, nous n'utilisons Glade que pour la forme de l'application et non le fond.

Etat Actuel

Actuellement, le programme est fonctionnel et permet une importation d'image. Pour faire fonctionner les scripts, il suffira de changer l'assignation des boutons aux scripts afin d'automatiser les taches.



Version Initiale du GUI

Avancements pour la 2e Soutenance

Pour la soutenance finale, nous allons majoritairement faire un travail d'automatisation des tâches sur l'interface graphique. L'objectif étant de résoudre le sudoku d'un simple clic de souris. Nous devrons donc mettre en place un script permettant d'assembler les différents programmes tels que le reseau de neurone ou la détection de grille. De plus, nous essayerons également de rendre l'application plus flexible en ajoutant des fonctionnalitées comme le choix du dossier de sauvegarde.

5 Conclusion

Nous espérons que notre projet vous aura convaincu. Nous sommes heureux de la tournure qu'a pris ce travail de groupe, et nous espérons continuer sur cette lancée pour le reste de la conception.