OCR - Soutenance Finale

Jacques SAUDUBRAY, Kylian GILBERT, Bastien GAULIER, Paul PAZART

December 2022

Table des matières

1 Introduction

Nous sommes une équipe composée de Kylian Gilbert, Jacques Saudubray, Bastien Gaulier et Paul Pazart. Au sein du projet, Kylian et Jacques s'occupent du traitement de l'image et du découpage des cases ; Paul s'occupe du réseau de neurones. Bastien, quant à lui, s'occupe du résolveur de sudoku et épaule Paul dans la construction du réseau de neurones.

Notre projet consiste à résoudre un sudoku utilisant la reconnaissance de caractère, grâce à l'intelligence artificielle d'un réseau de neurone et utilisant un traitement précis sur l'image.

2 Environnement de travail

Pour mener à bien ce projet, nous avons établi un dépôt Github afin que nous puissions échanger et progresser sur notre projet à tout moment. Nous avons également créé un serveur discord pour communiquer entre nous. Enfin, un Trello a été mis en place afin de suivre les tâches en cours et restantes en temps réel.

3 Attentes

Au début du projet jusqu'à la première soutenance, notre objectif pour ce projet était assez large : subvenir à chaque attente du cahier des charges. Nous pensions qu'il s'agissait d'un objectif atteignable et assez proche, même si notre avancement pour le premier rendu était discutable.

Cependant, il nous est apparu à un moment que nous ne progressions pas assez vite pour réaliser tout ce qu'il fallait. Cela était dû à différents facteurs, mais surtout aux problèmes rencontrés dans le développement du projet.

Nous nous sommes alors fixés l'objectif d'accomplir quelque chose de simple mais fonctionnel, un projet qui remplirait les objectifs principaux du cahier des charges.

4 Traitement de l'image

4.1 Introduction

Pour tout ce qui touche au traitement d'image, nous avons divisé le travail en plusieurs tâches que nous nous sommes ensuite attribuées. Comme dit ci-dessus, Kylian et Jacques se sont occupés de cette partie. Nous avons chacun réalisé les tâches de notre côté et nous les avons mises en commun sans problème. Pour ce qui est de la répartition de ces tâches, Jacques s'est occupé du prétraitement de l'image afin de rendre la grille la plus détectable possible. Kylian a de son côté réalisé la rotation manuelle de l'image ainsi que la détection des cases et le découpage de celles-ci. Il faut savoir que tout le prétraitement a été réalisé à l'aide de la librairie SDL2 compatible avec les programmes C. La répartition des tâches est similaire à ce que nous avions pour la première soutenance.

4.2 Prétraitement

4.2.1 Introduction

Cette partie a été réalisée par Jacques Saudubray. Dans cette partie, nous verrons comment l'image originale a pu être modifiée afin de faciliter la détection des lignes et ainsi des cases. Ce premier traitement d'image supprime les couleurs de l'image dans un premier temps afin d'obtenir des teintes de gris. Une modification de la luminosité est ensuite appliquée à celle-ci pour rendre l'image plus claire et éviter les zones d'ombres. Puis le contraste de l'image est à son tour modifié. À la suite de cela, l'image est binarisée, ne gardant que des pixels blancs ou noirs. Pour finir, on vient appliquer un filtre de Sobel afin de faire ressortir les lignes de la grille. S'ajoute à cela la rotation manuelle de l'image faite en parallèle.

Changements majeurs : Entre la première et la seconde soutenance, certaines étapes de ce traitement ont été modifiées. Tout d'abord, le flou gaussien n'est plus utilisé, car en ayant réduit le contraste de l'image, celui-ci venait perturber le bon rendu de l'image. Depuis la première soutenance, nous avons rajouté des fonctions agissant sur la luminosité de l'image et son contraste. De même, nous avons implémenté un filtre de Sobel pour faire ressortir les lignes et les contours. Concernant la binarisation, nous avons gardé la même méthode qu'à la première soutenance, à savoir la méthode d'Otsu mais n'avons pas gardé le seuil par zones.

Problèmes : Lors de la première soutenance, notre prétraitement avait rencontré

certains problèmes. Nous avons réussi à en régler certains mais d'autres sont encore présents. En ce qui concerne les zones d'ombres, l'augmentation de luminosité a suffi à régler le problème. Pour la suppression du bruit, nous avons essayé de faire un traitement avec seuil adaptatif afin qu'il n'y ait plus de taches et de bruits parasites. Malheureusement, cela posait problèmes pour d'autres images et le rendu n'était pas celui espéré.

Nous sommes donc restés sur notre méthode d'Otsu utilisée à la base qui marchait pour la majorité des images. Enfin, nous avons pu améliorer la détection de lignes et notamment les lignes au sein même de la grille qui pouvait ne pas être présentes au paravent. Cela a été fait en supprimant le flou gaussien qui gênait l'opération et en réduisant le contraste de l'image afin que les lignes internes étant très fines et peu visibles puissent être détectées. De plus, le filtre de Sobel permet de les faire ressortir encore mieux pour un rendu optimal.

Présentation : Nous allons prendre une image et montrer le rendu des différentes étapes du traitement d'image jusqu'au filtre de Sobel. Sur les 6 images fournies au départ, ce prétraitement marche sur 5 d'entre elle. Nous prendrons la troisième comme exemple.

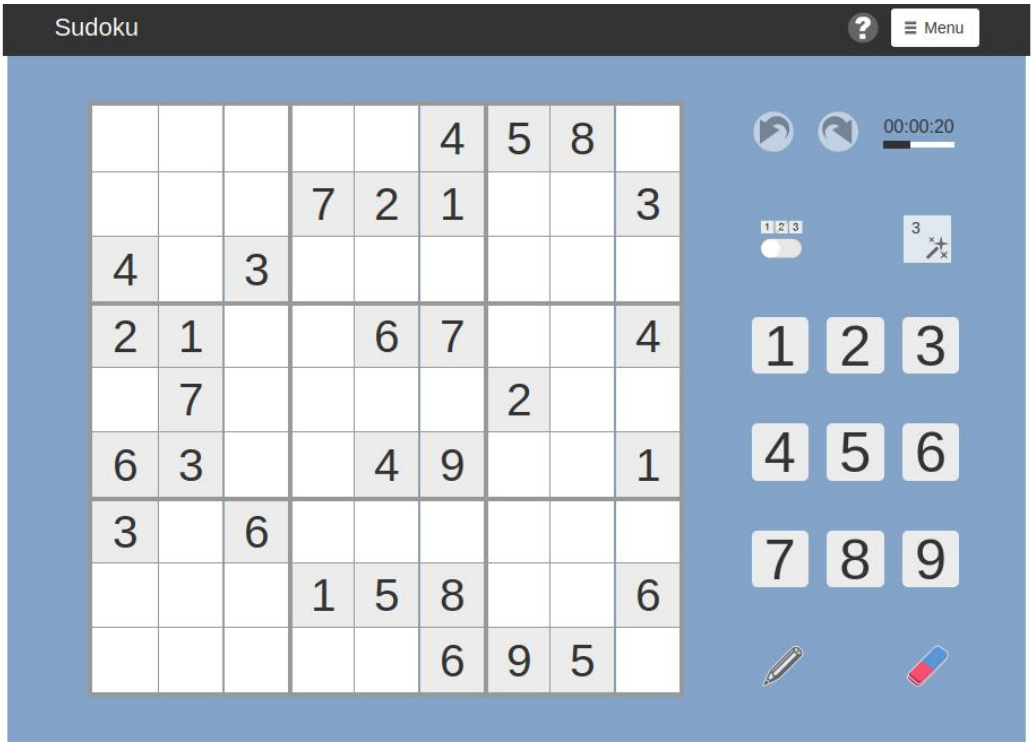


FIGURE 1 – Image de depart

4.2.2 Suppression des couleurs

La suppression des couleurs consiste à modifier l'image afin d'obtenir un panel de gris correspondant aux couleurs initiales de l'image. Cette partie a été réalisée lors d'un TP de programmation et cela était bien guidé. Nous n'avons eu aucun problème sur cette partie et cela ne nous a pas pris trop de temps. Il suffit de modifier les composantes r, g, b de chaque pixel de l'image afin d'obtenir du gris. Voici le rendu sur l'image donnée :

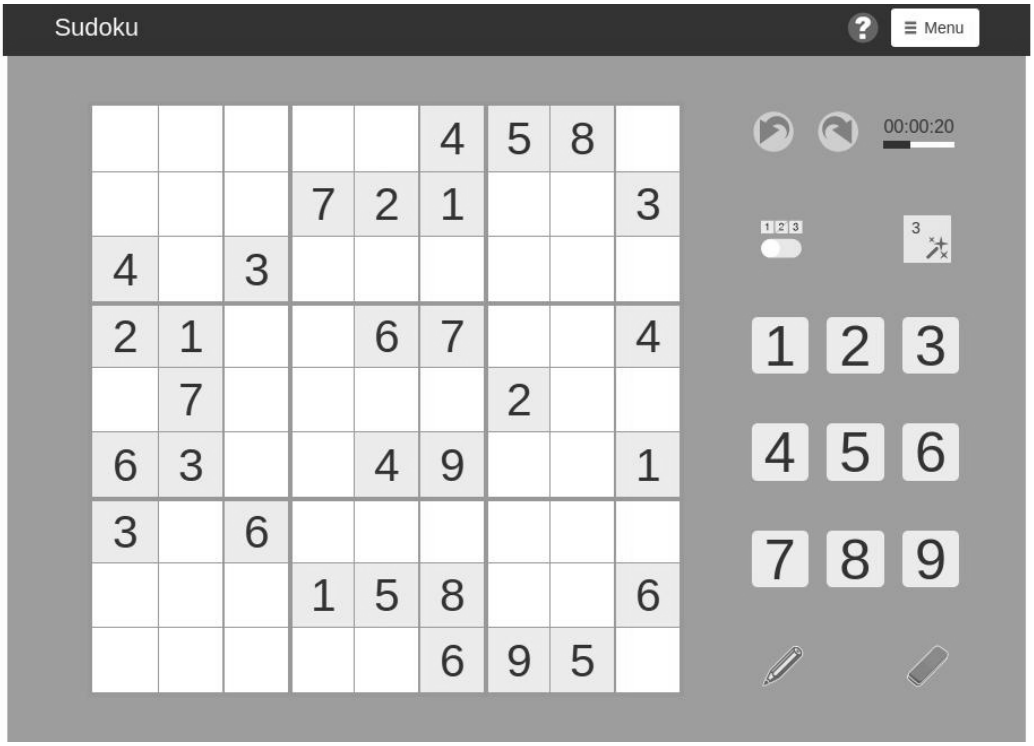


FIGURE 2 – Niveau de gris

4.2.3 Luminosité

Afin de réduire les zones d'ombres pouvant ainsi gêner lors de la binarisation, nous devions modifier la luminosité de l'image afin de la rendre plus claire. Cependant, nous voulions rendre en particulier les zones d'ombres plus claires et non toute l'image. Pour cela, nous avons créé une fonction respectant cette courbe :

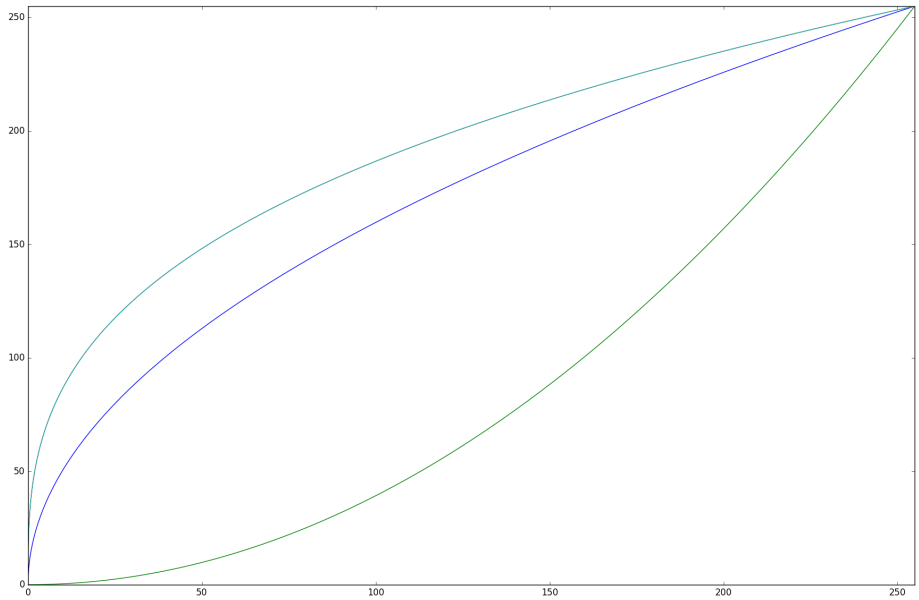


FIGURE 3 – Courbes de la fonction

Ces courbes sont représentées par la fonction suivante :

$$f(x) = 255.(x/255)^n$$

Ici le x représente la couleur du pixel sélectionné et n correspond à l'intensité de la luminosité. La courbe turquoise est obtenue avec n = 1/3, la bleue avec n = 1/2 et la verte avec n = 2. Pour n = 1, la courbe serait x = y, soit la fonction identité et le pixel resterait de la même couleur.

Ainsi, quand n > 1, la courbe se situe en dessous de la fonction identité et cela vient donc assombrir la couleur du pixel. À l'inverse, quand n < 1, cela va augmenter la luminosité de celui-ci. Nous avons donc choisi un n égal à 0.5 pour notre traitement et cela fonctionne.

Enfin, il nous suffit d'appliquer cette fonction à chacun des pixels de la surface afin d'appliquer le filtre de luminosité sur toute l'image et le tour est joué. Voici le rendu sur l'image donnée :

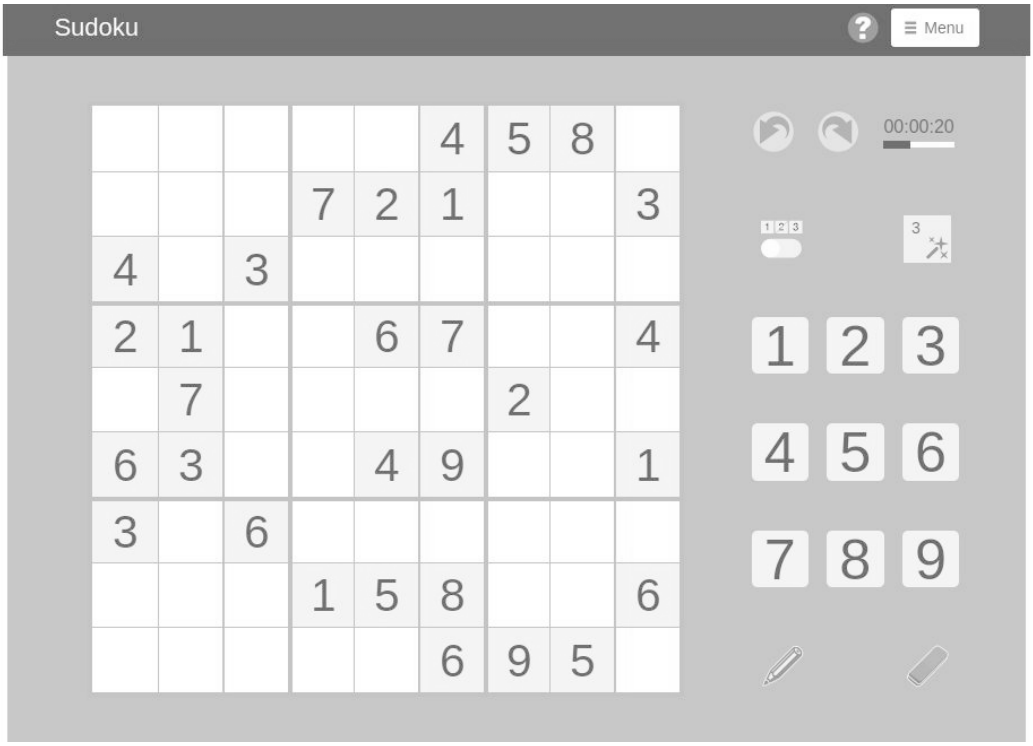


FIGURE 4 – Augmentation de Luminosité

4.2.4 Contraste

Le contraste est une partie tout aussi importante que la luminosité. Cela ressemble beaucoup à ce qui a été fait pour la luminosité, on va simplement venir changer la fonction qui s'applique à chaque pixel. On cherche ainsi à augmenter ou diminuer la luminosité des pixels étant soit très sombres, soit très clairs et on évite de trop toucher à ceux étant dont la couleur étant très grise. Les valeurs allant de 0 à 255, on va surtout modifier celles proches des extrémités. On va donc se retrouver avec ce type de courbe :

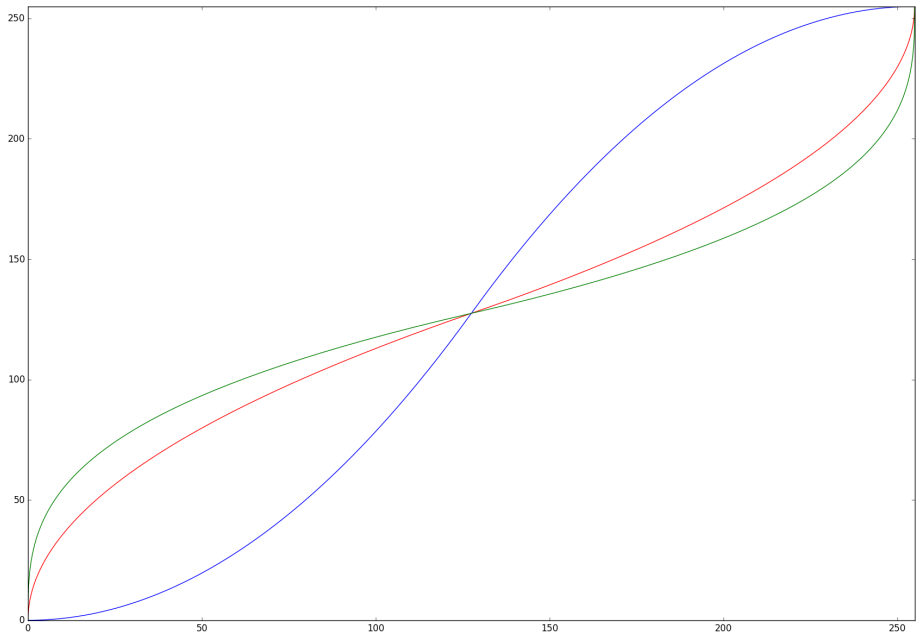


FIGURE 5 – Courbes de la fonction

Encore une fois, cela dépend de n, qui définit l'intensité du contraste. La courbe bleue est obtenue pour n = 2, la rouge pour n = 1/2 et la verte pour n = 1/3. Ainsi quand n est supérieur à 1, le contraste est amélioré et quand n < 1, le contraste est réduit. Nous avons d'abord pensé à renforcer les contrastes de l'image, mais cela pouvait ne pas prendre en compte certaines lignes intermédiaires de la grille de sudoku. Nous avons donc essayé l'inverse et en réduisant celui-ci, les lignes après la binarisation ressortent bien mieux et nous rencontrons aucun problème. Ainsi, nous avons choisi un n égal à 0.6 pour le contraste et nous l'avons appliqué sur toute l'image. Voici le rendu sur l'image donnée :

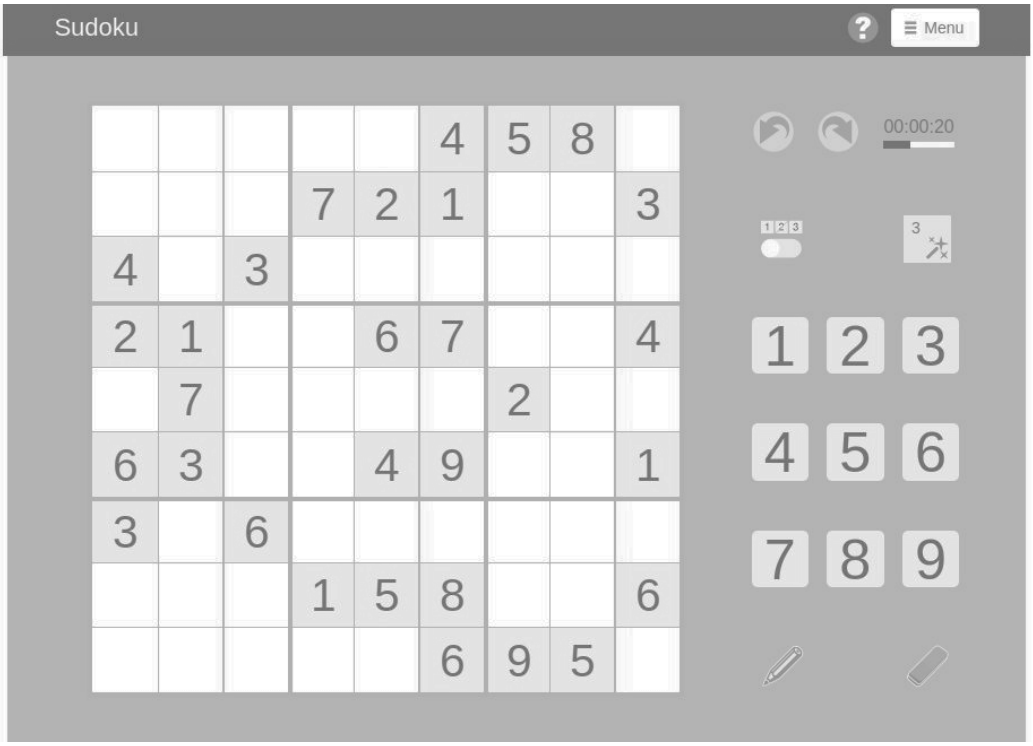


FIGURE 6 – Diminution du Contraste

4.2.5 Binarisation de l'image

La binarisation de l'image permet de se retrouver avec une image aux pixels uniquement noirs ou blancs. C'est une étape nécessaire pour la suite du traitement d'image. Plusieurs méthodes sont utilisées pour la binarisation d'image.

Nous avons choisi d'utiliser la méthode d'Otsu, celle-ci étant la plus couramment utiliser et l'une des plus simples à implémenter. Elle consiste à déterminer un seuil tel que si la couleur d'un pixel est au-dessus de ce seuil, ce pixel devient noir et à l'inverse, celui-ci devient blanc. Ce seuil est calculé sur l'ensemble de l'image avec la couleur de chaque pixel. Voici une image de rendu après la binarisation :

Nous avons réussi à implémenter correctement cette méthode, mais un problème est survenu pour certaines images. En effet, lorsqu'il y a du bruit, cela créer des gros nuages noirs et peut nuire au traitement d'image. Pour cela, nous avons modifié la méthode afin de déterminer plusieurs seuils à des échelles locales. Ainsi, nous avons découpé l'image en plusieurs cases et nous avons déterminé un seuil pour chacune d'entre elles, étant ensuite appliqué à chaque pixel de cette zone. Cette méthode a

marché pour certaines images, mais d'autres peuvent encore rencontrer des problèmes suivant le nombre de zones que l'on prend pour découper l'image. Nous devions encore faire des modifications manuellement pour chacune des images afin d'avoir un rendu optimal. Finalement, nous sommes restés sur la version de base étant plus simple et plus sûre. Voici le rendu sur l'image donnée :

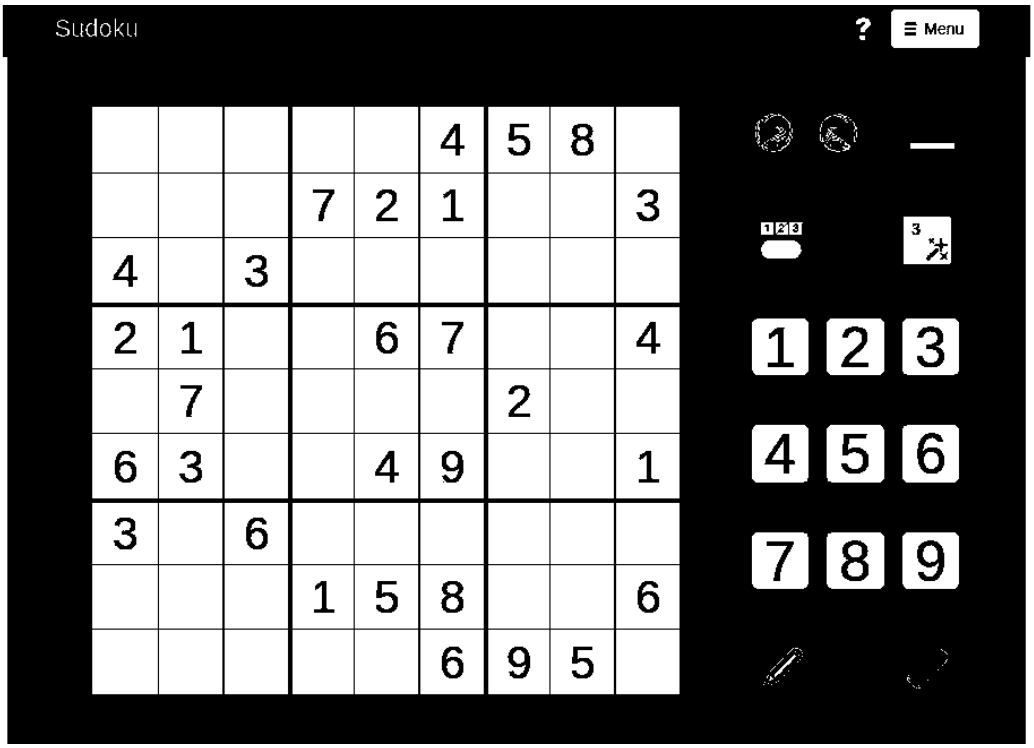


FIGURE 7 – Binarisation

Nous avons également essayé de calculer un seuil adaptatif pour chaque pixel de l'image. Seulement, le rendu n'était pas bon pour la plupart des images. Nous avons donc préféré garder la première méthode quitte à ce que cela ne gère pas entièrement la suppression de bruit, car celle-ci donne un meilleur rendu après binarisation sur les différentes images.

4.2.6 Inversion des couleurs

Une autre fonctionnalité a été rajoutée afin d'inverser les couleurs de l'image. En effet, pour un meilleur traitement de l'image et une meilleure reconnaissance des chiffres par le réseau de neurone, nous voudrions une grille blanche, incluant les

chiffres en blanc sur un fond noir. Pour cela, nous avons créé une fonction permettant de voir si la majorité des pixels sont blancs ou noirs et les modifier quand il le faut en inversant leur couleur. Aucun problème de ce côté a signalé, la fonction étant simple à réaliser. Voici le rendu sur l'image donnée :

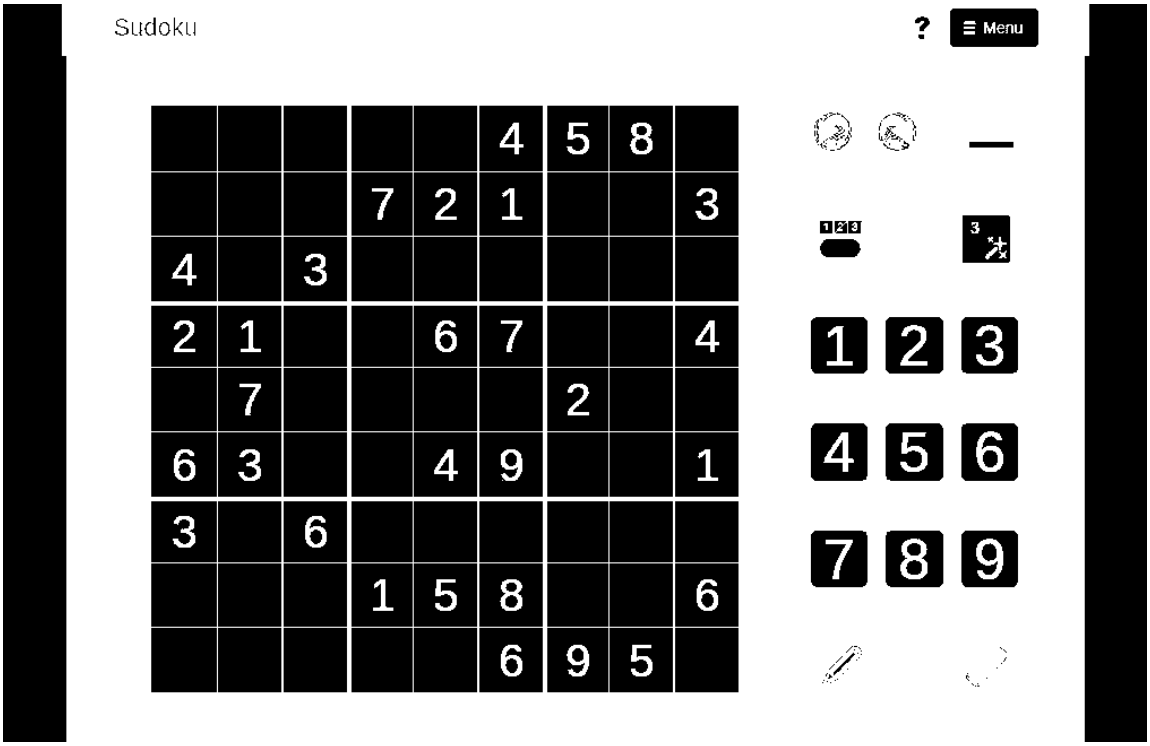


FIGURE 8 – Inversion des couleurs

4.2.7 Filtre de Sobel

Afin de faciliter la détection de ligne, nous avons décidé d'appliquer un filtre de Sobel a notre image. Cet opérateur permet de calculer le gradient de l'intensité de chaque pixel de l'image. Cela permet de voir les plus grosses variations du clair au sombre. Pour cela, on utilise des matrices de convolution. Une pour la différence de couleur sur les y et une pour les x. En chaque point, on obtient des approximations des gradients horizontaux et verticaux. On vient ensuite calculer la norme du gradient, celle-ci est calculée en faisant la racine carre des deux gradients aux carrés.

Cette fonction vient faire ressortir les différences de d'intensité de chaque pixel. Ainsi, on se retrouve avec des contours mis en avant, utile pour détecter les bords de la grille. Voici le rendu sur l'image donnée :

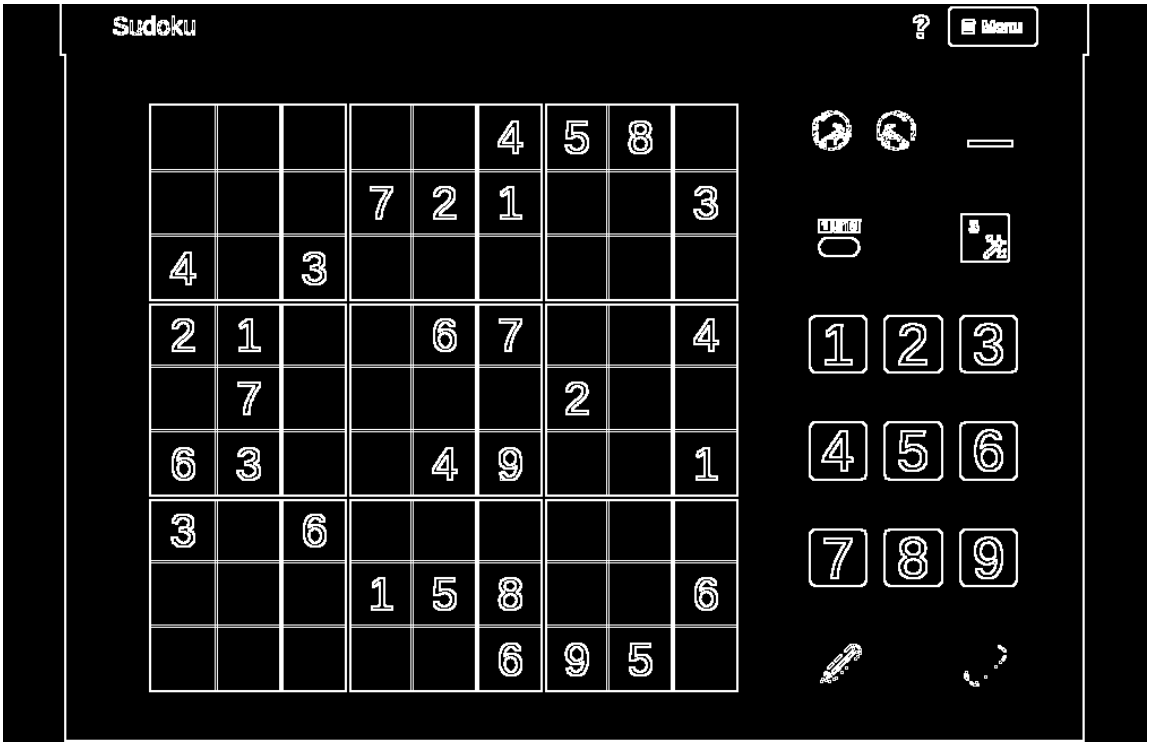


FIGURE 9 – Filtre de Sobel

4.2.8 Rotation

Pour ce qui est de la rotation, nous avons pour cette soutenance finale une rotation manuelle et pas de rotation automatique. En effet pour la rotation automatique il aurait fallu détecter la grille, chose que nous n'avons pas fait. Pour la première soutenance, Kylian était chargé de s'occuper de la rotation manuelle, mais cela ne marchait qu'avec une rotation de 90 degrés. Nous avons réussi à appliquer une rotation avec n'importe quel angle pour cette dernière soutenance. Nous avons créé 3 fonctions, la première étant la principale et qui applique la rotation a toute l'image. Une seconde permettant de convertir les angles de degrés en radians afin de faciliter les calculs. La dernière est le calcul de nos nouvelles coordonnées x et y après un calcul via des sinus et cosinus. Voici un rendu de la rotation manuelle avec un angle de -35 degrés sur l'image 5 :

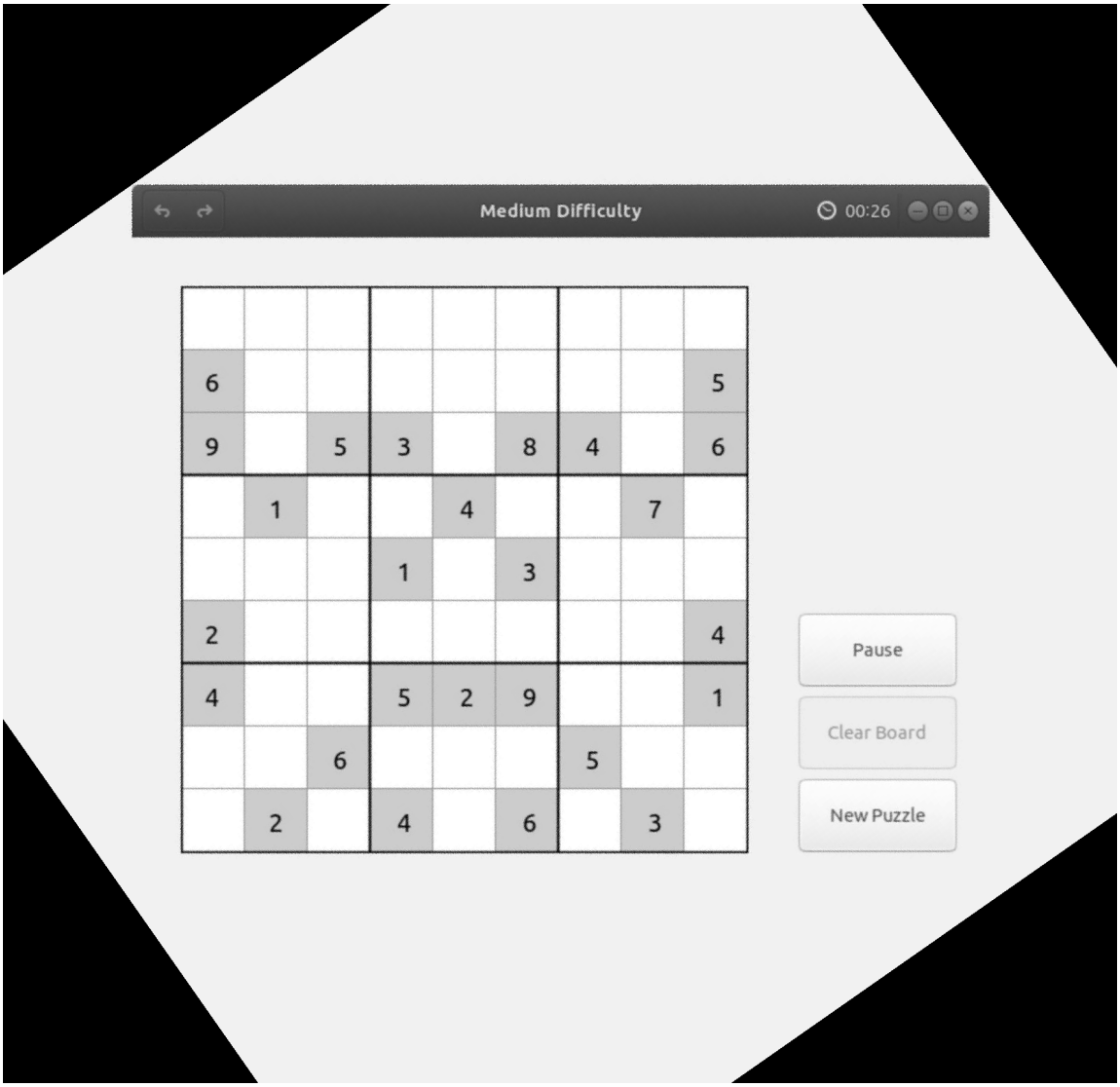


FIGURE 10 – Rotation manuelle

4.3 découpage de la grille

4.3.1 Hough

Pour découper la grille du sudoku, on a choisi d'utiliser la transformée de Hough qui permet de détecter les lignes. Pour cela, la première étape est donc de calculer la transformée qui nous produit une nouvelle matrice qu'on appelle aussi l'espace de

Hough, avec un des angles thêta en largeur et la distance de l'image en longueur. On la construit en parcourant toutes l'image, et lorsque qu'un pixel est blanc (dans notre cas après le prétraitement les lignes et les chiffres sont en blanc) pour chaque angle θ de la matrice, on calcule ρ avec, ρ = i cos (θ) + (Ny - j)sin (θ). Avec i et j la position du pixel dans l'image. Vous pouvez voir un exemple sur 3 rectangles du résultat.

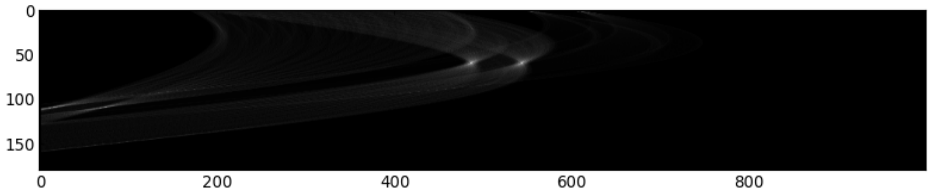


FIGURE 11 – Espace de Hough

4.3.2 Découpe brut

Ensuite, il faut récupérer les droites qui nous intéressent, donc pour ça on recherche dans l'espace de Hough les valeurs les plus importantes, celles qui sont audessus d'un certain seuil. mais comme vous pouvez le voir 12 toutes les lignes ne nous intéressent pas, et certaines qui nous intéresseraient n'apparaissent pas.

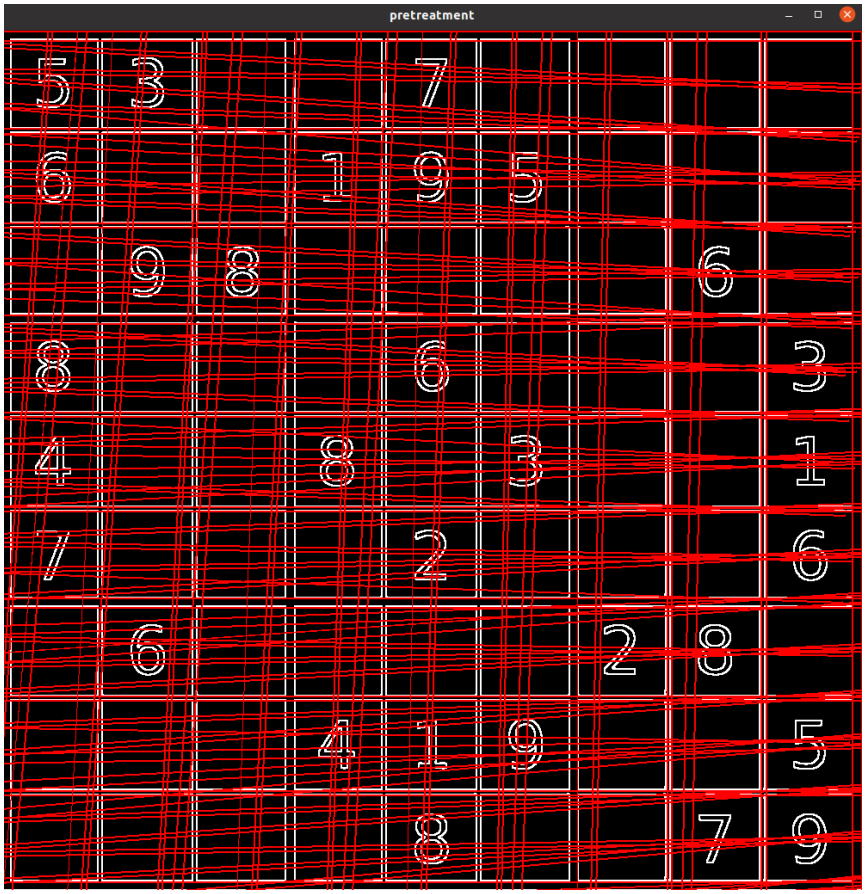


FIGURE 12 – Hough sans traitement

Et il y a beaucoup trop de lignes qui se chevauchent. D'autres qui sont en diagonales. Il nous est donc impossible d'envoyer une découpe comme ça à l'équipe du réseau de neurone. La découpe serait juste inutilisable.
Ce problème est lié au fait que la concentration de nombre donne l'impression d'avoir une ligne, l'algorithme est donc perturbé.

4.3.3 Traitement de la découpe

La première étape pour éliminer les lignes qui ne nous intéressent pas, c'est de retirer les diagonales (on part du principe que l'image est déjà tournée dans le bon sens). pour ça on récupère pour les colonnes les position des deux x et on vérifie qu'ils ne sont pas trop éloignés l'un de l'autre. Et on fait la même pour les lignes en regardant les y De cette manière, on a nettoyé toutes les diagonales en trop. Ensuite,

on a regardé sur chaque lignes la quantité de pixel blanc

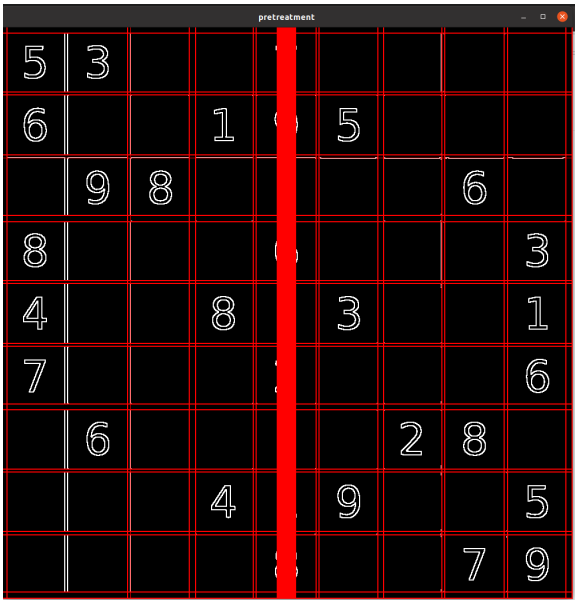


FIGURE 13 – 1er traitement

Ensuite, il a fallu retirer les lignes qui se collaient. Pour ça on reparcourt la liste des lignes. Et lorsque les colonnes sont trop proches entre elles, on en trace une au milieu. De ce fait, on se retrouve avec une seule ligne à chaque fois.

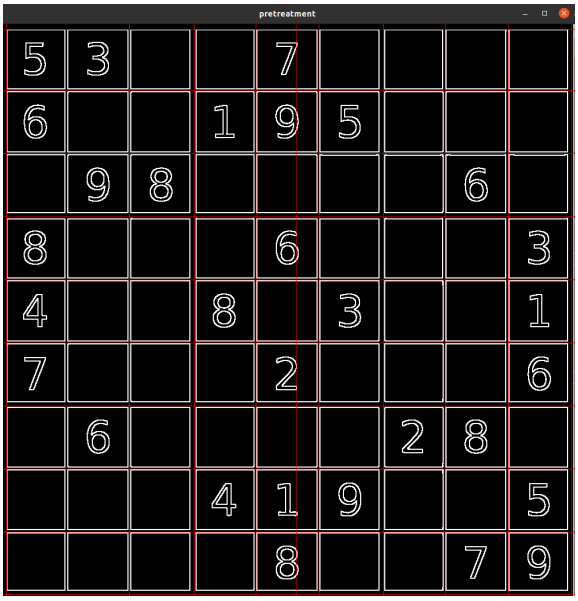


FIGURE 14 – Espace de hough

Mais il nous reste encore de temps en temps une ou deux lignes au milieu de l'image, la ou la concentration de nombre est importante. Ou alors une ligne ou deux qui n'apparaissent pas. Pour ça on va donc regarder la distance entre deux lignes, si elle est trop importante, on rajoute donc une nouvelle ligne, et là où elle est trop faibles on en retire une.

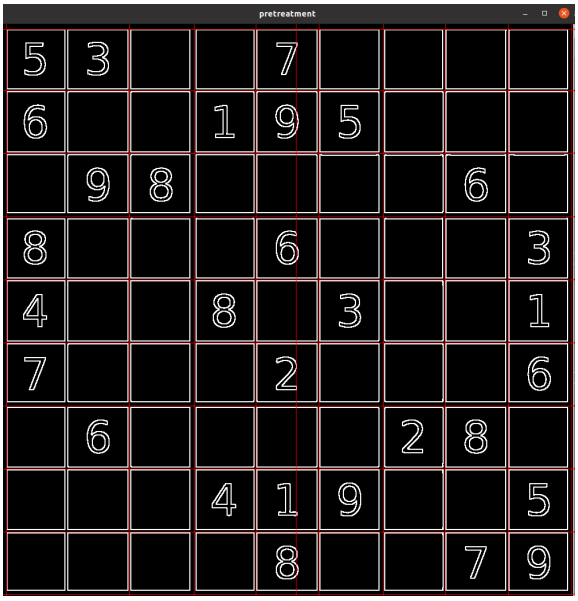


FIGURE 15 – 2ème traitement

4.3.4 Partage des cases

Enfin, la dernière partie est de récupérer chaque case et de la redimensionner en compressant pour envoyer une liste de tableau de 28 par 28 à l'équipe du réseau de neurones pour qu'elle reconnaisse chaque chiffre et résolve le sudoku.

5 Réseau de neurones

Pour le réseau de neurone, nous avons une structure mais nous n'avons pas réussi à l'entraîner à cause de nombreux problèmes causés par la backpropagation. Le réseau de neurone a reçu une refonte en profondeur. Il est composé de 784 neurones d'entrés, 16 neurones cachés et 10 neurones de sortie. Cette section a été réalisé par Paul PAZART, aidé par Bastien GAULIER.

5.1 La structure

Notre réseau de neurone s'est vu rajouter une structure "neuron" gardant tous les attributs d'un neurone, c'est-à-dire, ses poids, biais et valeurs après la fonction

d'activation. Ensuite, nous avons créé une structure "layer" qui, lui, garde les différents neurones d'une couche. Enfin, une structure "network" a été implémentée pour stocker les différentes couches du réseau.

```
typedef struct network network;
struct network {
    layer** array;
    long length;
};
```

(a) Structure du réseau comportant un tableau pour les couches et la longueur du tableau

FIGURE 16 – Structure du réseau

```
typedef struct layer layer;
struct layer{
    neuron** array;
    long length;
};
```

(a) Structure du layer comportant un tableau pour les neurones et la longueur du tableau

FIGURE 17 – Structure de la couche

```c
typedef struct neuron neuron;
struct neuron{
    long weight_length;
    double* weight;
    double bias;
    double value;
};
```

(a) Structure du neurone comportant un tableau comportant un tableau des poids, une longueur de ce tableau, un biais et une valeur après calcul

FIGURE 18 – Structure du neurone

5.2 Le forward pass

Cette fonction permet de calculer une donnée de sortie en fonction des données d'entrées. Tout d'abord, on a implémenté deux fonctions d'activations : LeakyReLu et Softmax.

La première est une variante de ReLu qui est la fonction d'activation utilisée dans les couches cachées. Les fonctions d'activation rectifiées servent pour séparer l'excitation spécifique et l'inhibition non spécifique dans la pyramide d'abstraction neuronale, qui a été entraînée de manière supervisée pour apprendre plusieurs tâches de vision par ordinateur. Les unités linéaires rectifiées, comparées à la fonction sigmoïde ou à des fonctions d'activation similaires, permettent un apprentissage plus rapide et efficace des architectures neuronales profondes sur des ensembles de données vastes et complexes.

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$$

FIGURE 19 – Fonction d'activation LeakyReLU

La seconde, la fonction softmax, également connue sous le nom de softargmax ou fonction exponentielle normalisée convertit un vecteur de K nombres réels en une distribution de probabilité de K résultats possibles. C'est une généralisation de la fonction logistique à plusieurs dimensions, et elle est utilisée dans la régression logistique multinomiale. La fonction softmax est souvent utilisée comme dernière fonction d'activation d'un réseau neuronal pour normaliser la sortie d'un réseau en une distribution de probabilité sur les classes de sortie prédites.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

FIGURE 20 – Fonction d'activation Softmax

5.3 La backpropagation

Cette fonction est la plus importante. Elle permet au réseau d'apprendre et reconnaitre les valeurs d'entrées au fil des entrainements. Dans l'ajustement d'un réseau neuronal, la rétropropagation calcule le gradient de la fonction de perte par rapport aux poids du réseau pour un seul exemple d'entrée-sortie, et ce de manière efficace, contrairement au calcul direct naïf du gradient par rapport à chaque poids individuellement. Cette efficacité rend possible l'utilisation des méthodes de gradient pour la formation de réseaux multicouches, en mettant à jour les poids afin de minimiser les pertes.

Au niveau de la couche externe, on utilise la dérivée de la softmax avec un calcul d'erreur pour mettre à jour les poids et les biais.

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_i(1 - p_j) & if \quad i = j \\ -p_j . p_i & if \quad i \neq j \end{cases}$$

FIGURE 21 – Dérivé de Softmax

Au niveau de la couche interne, on utilise la dérivée de LeakyReLU avec la somme des erreurs de la couche précédente couplée avec les valeurs des neurones des couches adjacentes.

$$f'(x) = \begin{cases} 0.01, & \text{if } x < 0 \\ 1, & \text{if } x >= 0 \end{cases}$$

FIGURE 22 – Dérivé de LeakyReLU

5.4 L'entraînement

Pour l'entrainement, on avait mis en place un système utilisant la base de données MNIST comportant plus de 60000 images d'apprentissage et 10000 images de tests. Elle est composée d'images en noir et blanc, normalisées et centrées de 28 pixels de côté, représentant des chiffres écrits à la main.

FIGURE 23 – Exemple d'images disponibles

Cependant, avec seulement cette base de donnée, notre réseau n'aurait pas été capable de reconnaitre les chiffres générés par ordinateur. C'est pourquoi, on a créé un script transformant une image en image de 28 par 28 compréhensible pour le programme. Ces chiffres proviennent de différentes sources comme d'un carnet de sudoku ou encore des exemples sources disponibles dans le dossier de présentation du projet.

5.5 Problèmes et regrets

Pour le réseau, nous avons dû changer toute la base du réseau. Pour ce faire, comme dit précédemment, nous avons utilisé une structure contenant tout le réseau. Il a fallu par la suite réadapter toutes les fonctions à la structure. La fonction de propagation vers l'avant (forward pass) est très similaire à la version précédente aux détails près qu'on utilise LeakyReLU comme fonction d'activation qui est plus adapté pour le cas d'un OCR et la fonction Softmax qui permet d'établir un vecteur de probabilité en fonction de la catégorie attendue.

Ensuite, il a fallu faire la backpropagation. Il s'agit d'une fonction permettant au réseau d'apprendre. Il y a donc plusieurs moyens de le faire dont l'objectif est d'avoir un entraînement rapide et efficace. Cependant, ça était la partie la plus difficile du réseau, car on avait des valeurs, après backpropagation, trop hautes au point où le système affichait "nan". Cela peut être très complexe pour avoir des valeurs pour l'entrainer.

Une autre partie complexe est le fait qu'il y a tellement de manière de faire une fonction backpropagation ce qui rend la recherche d'information très complexe et ralentit beaucoup la création du réseau, mais aussi la compréhension du fonctionnement de ce dernier. Je pense notamment au calcul de l'erreur et aux formules pour la propagation de l'erreur au sein du réseau.

Pour conclure, la création d'un réseau de neurone type XOR est très facile à trouver, car toutes les sources sont à peu près unanimes. Néanmoins, dès que l'on cherche des informations sur un réseau beaucoup plus gros, cela devient vite difficile, car tout va dépendre de l'implémentation de départ, de la qualité de l'article et des informations compris dedans. Par ailleurs, déboguer un réseau de neurone est une véritable épreuve puisque l'origine des bugs peut venir de différentes raisons : les valeurs des poids et des biais mal initialisées, les fonctions de forward et backward propagation cassées ou ne possédant pas la bonne formule ou encore une mauvaise implémentation, une erreur d'indexage... Ainsi, j'ai vite compris l'intérêt des cours sur les réseau de neurones.

6 Traitement du sudoku

En ce qui concerne le traitement du sudoku, nous avons respecté notre objectif en tous points. Tout d'abord, la résolution du sudoku lui-même a été réalisée : L'algorithme de résolution a été étudié, mis en place puis optimisé. Ensuite, le système de sauvegarde de la grille a été fait, de façon à ce qu'on puisse non seulement sauvegarder une grille mais aussi en charger une. Cette section a été réalisée par Bastien GAULIER.

6.1 Le sudoku

Commençons par rappeler l'objectif de notre projet : détecter et résoudre un sudoku. Tout d'abord, qu'est-ce que c'est ? Un sudoku est un jeu d'origine japonaise mettant en scène une grille de 9x9 cases, divisés en carrés de 3x3 cases appelés régions. Le but du jeu est de remplir cette grille avec des chiffres de 1 à 9 de façon à ce que chaque ligne, chaque colonne et chaque région ne comporte pas plus d'une fois le même chiffre. De plus, au début du jeu, certaines cases sont déjà occupées par des chiffres. Il faut aussi ajouter que, bien que tous les sudokus que l'on retrouve dans les magazines de jeux sont solubles, il existe certaines dispositions de chiffres qui, sans rendre la grille incorrecte, la rend infinissable.

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

FIGURE 24 – Un sudoku solvable.

6.2 Algorithme de résolution

Afin de résoudre la grille de sudoku qui sera détectée par notre OCR, nous nous sommes tournés vers l'algorithme du **Backtracking**. À la fois simple et rapide, il était pour nous le choix évident pour notre projet, d'autant que nous avions déjà eu l'occasion de le manipuler.

Son principe, récursif, est simple : il va parcourir toutes les cases du sudoku ligne par ligne en partant du coin supérieur gauche. Pour chaque case vide, on va essayer

d'insérer des nombres pour ensuite vérifier si les règles du sudoku sont respectées. Si l'algorithme trouve une solution au sudoku, il renvoie VRAI et la grille qu'on lui a donnée en entrée est modifiée pour représenter la solution. Sinon, il renvoie FAUX et la grille de départ est laissée intacte.

Ainsi, voici les grandes lignes de cet algorithme pour une quelconque case :

— Si cette case est en-dehors de la grille, on a potentiellement trouvé la solution : il faut simplement vérifier si la grille est encore valide. On renvoie ce résultat.
— Si on est revenu au début sans trouver de solution, c'est qu'il n'y en a pas : on renvoie FAUX.
— On vérifie si la grille respecte encore les règles du sudoku. Sinon, on renvoie FAUX.
— Si C n'est pas vide, alors on appelle l'algorithme sur la case suivante.
— Sinon, on y place tour à tour les chiffres de 1 à 9, et pour chacun, on appelle l'algorithme sur la case d'après.
— Si l'un de ces appels récursifs renvoie VRAI, il faut aussi renvoyer VRAI, car on a trouvé la solution.
— Si aucun appel récursif n'a trouvé de solution, on n'en a pas trouvé sur cette case : on renvoie FAUX.

Une fois que nous avons implémenté cet algorithme, il nous est vite apparu qu'il n'était pas très optimisé, voire assez bête. De plus, bien que le temps de son exécution soit bien en dessous de la seconde, nous voulions sauver le maximum de temps possible. En effet, les parties du traitement d'image prendront assez de temps comme cela, nous ne voulons pas ralentir encore plus notre programme.

Ainsi, nous avons mis en place un niveau d'optimisation à notre algorithme de backtracking, qui se remarque lors de l'étape d'insertion des chiffres dans la case courante. Désormais, l'algorithme n'essaye pas aveuglément tous les chiffres de 1 à 9 pour une case : il ne va placer que les chiffres qui ne rendront pas la grille invalide. Cela enlève l'étape de vérification très gourmande pour une rapide vérification pour chaque chiffre. Dès lors que nous avons instauré cela, l'algorithme s'exécute très rapidement, même pour les grilles insolubles.

Voici alors le nouvel algorithme :

— Si la case est en-dehors de la grille, on a trouvé la solution : on renvoie VRAI.
— Si on est revenu au début sans trouver de solution, c'est qu'il n'y en a pas : on renvoie FAUX.
— Si C n'est pas vide, alors on appelle l'algorithme sur la case suivante.

— Sinon, on y place tour à tour les chiffres qui gardent la grille valide, et pour chacun, on appelle l'algorithme sur la case d'après.
— Si l'un de ces appels récursifs renvoie VRAI, il faut aussi renvoyer VRAI car on a trouvé la solution.
— Si aucun appel récursif n'a trouvé de solution, ou il n'existe aucun chiffre ne compromettant pas la grille, on n'a pas trouvé de solution sur cette case : on renvoie FAUX.

6.3 Système de sauvegarde

Après avoir détecté et résolu un sudoku, notre OCR doit aussi être capable de le sauvegarder, pas simplement de l'afficher puis d'en effacer toute trace. Ainsi, pour cette soutenance, ce système a été réalisé, de façon à ce qu'une grille de sudoku, après avoir été résolue, est enregistrée dans un fichier sur la machine.

Le format utilisé dans le fichier texte de sauvegarde est très simple : la grille y est représentée telle quelle, les chiffres de 1 à 9 les uns à la suite des autres dans un carré de 9*9 caractères. Les cases vides sont représentés par un point('.').

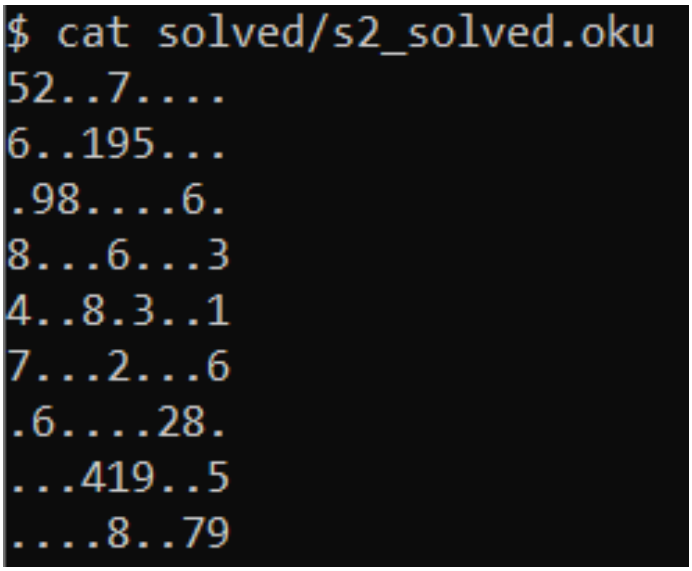


FIGURE 25 – Fichier de sauvegarde d'un sudoku insoluble

Il faut aussi noter que notre système de sauvegarde enregistre quand même un

sudoku même si il n'est pas soluble. En effet, il peut être utile de se rappeler une grille détectée par l'OCR, ou simplement pour montrer un exemple de sudoku insoluble. De plus, il est aussi possible pour ce système de charger un sudoku à partir d'un fichier de sauvegarde.

Enfin, en ce qui concerne les liens entre cette partie et le reste du projet, ils ont été assez simples à mettre en place. Tout d'abord, on constitue une grille de 81 cases avec chacun des résultats donnés par le réseau de neurones. Ensuite, la grille obtenue est soumise à l'algorithme de **backtracking** qui va la résoudre, ou pas. Enfin, l'utilisateur pourra voir ce résultat et décider ou non de l'enregistrer dans un fichier particulier.

Cependant, il nous est vite apparu que la sauvegarde d'un simple fichier texte n'allait pas suffire pour avoir un rendu propre. Nous avons alors remplacé ce système obsolète par un moyen de convertir une grille de chiffres en image véritable de ce sudoku au moyen d'images de chiffres déjà prêtes. Se référer à la par

7 Application graphique

Enfin, il nous faut évoquer l'application graphique qui régit toutes les différentes parties de ce projet. Une telle chose fournit à l'utilisateur un moyen interactif, intuitif et visuel d'utiliser l'outil mis à sa disposition. De plus, cela rend le projet bien plus propre et permet de regrouper assez facilement ses parties éparses. Notre application graphique a été réalisée principalement par Bastien GAULIER.

7.1 Outils

Afin de développer et d'exploiter une application graphique, nous nous sommes tournés vers les bibliothèques libres GTK, utilisées pour le développement d'interfaces graphiques. Nous avons aussi employé l'outil Glade, permettant de concevoir l'apparence des interfaces que concrétise ensuite GTK. Avec ces deux outils à notre disposition, nous avons réalisé une application qui, bien qu'assez simple, suffit largement aux besoins de notre projet.

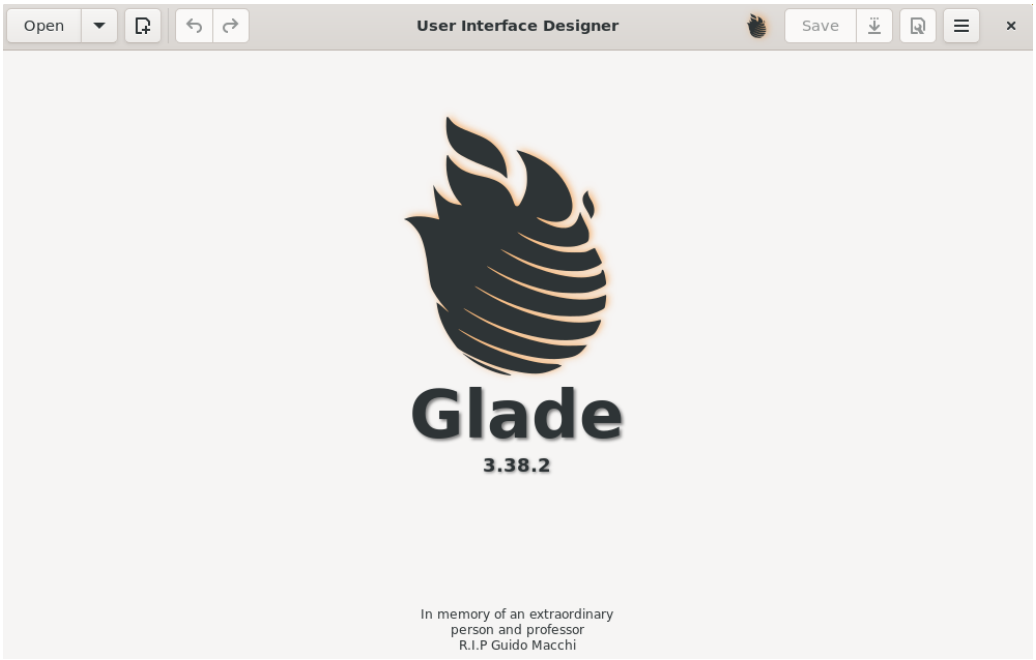


FIGURE 26 – L'outil Glade

7.2 L'interface

En ce qui concerne l'interface concrète, nous avions prévu pour ce projet un outil simple mais efficace. Il comprend des fonctionnalités faciles d'utilisation sans non plus en être saturé. Nous voulions aller au plus simple et réaliser quelque chose de fonctionnel.

Voyons ainsi ce à quoi notre interface graphique ressemble et les fonctionnalités qu'elle renferme :

FIGURE 27 – L'application de notre OCR

Il y a quelques éléments sur cette surface, mais pas trop non plus. Cela permet à l'utilisateur de savoir quoi faire immédiatement sans être dérangé par beaucoup de boutons. Ainsi, tous les boutons sont répartis sur la gauche de l'interface pour laisser le reste de la place à l'affichage d'une image. Il faut aussi noter que les fonctionnalités n'ayant pas d'utilité à un instant t ne seront pas utilisables. Cela se traduit par exemple par des boutons grisés non cliquables. Toutes ces fonctionnalités sont expliquées ci-dessous.

7.2.1 Le bouton de chargement

Lorsqu'on clique sur ce bouton, une fenêtre d'exploration s'ouvre. Celle-ci nous permet de parcourir les fichiers de la machine et de sélectionner une image parmi ceux-ci. Une fois cela fait, elle est immédiatement affichée sur la partie droite de l'interface. Il faut aussi noter que si le fichier spécifié n'est pas une image, rien ne sera changé et un message d'alerte sera affiché dans la sortie standard. Dans le cas contraire, c'est un message de confirmation qui sera envoyé vers cette sortie, où sera rappelé le chemin de l'image chargée.

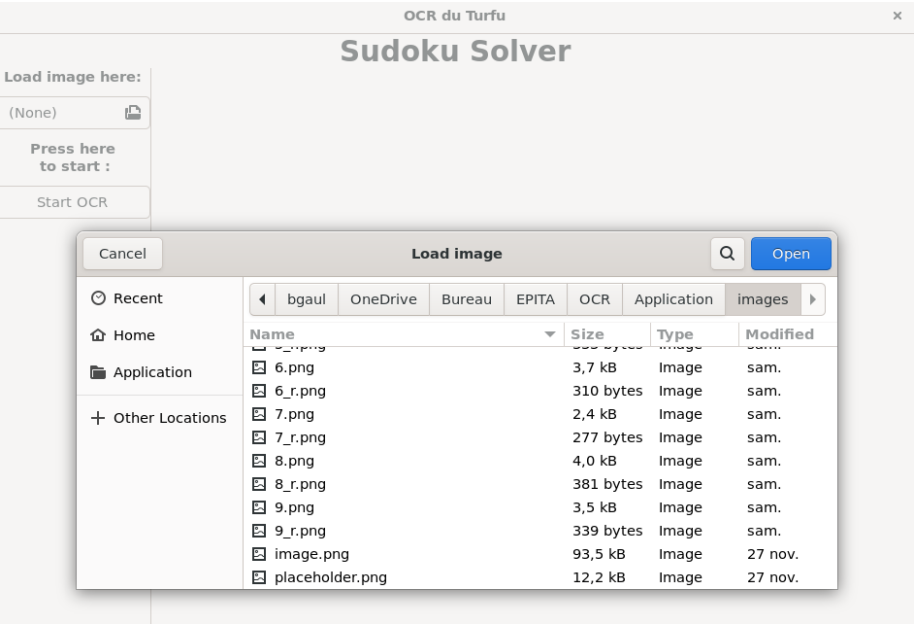


FIGURE 28 – La fenêtre d'ouverture d'image

Remarque : la fenêtre de recherche de fichiers est gérée par GTK et Glade ; ainsi, malgré son apparence compliquée, sa gestion par le code est en fait triviale.

7.2.2 Le bouton de lancement

Le deuxième bouton de l'application permet tout simplement de lancer la procédure globale du projet. Les étapes de prétraitement de l'image, détection de la grille et découpage des cases, extraction des chiffres et résolution de la grille vont s'exécuter les unes après les autres.

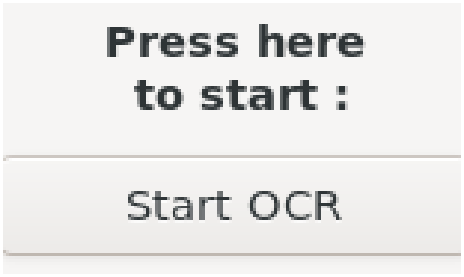


FIGURE 29 – Le bouton pour commencer l'OCR

Notons également que tant qu'une image n'a pas été chargée, il n'est pas possible d'appuyer sur ce bouton et donc de lancer le programme sur du vide.

7.2.3 Les boutons de cycle

Après la sélection d'une image, si tout s'est bien passé, le programme obtiendra un résultat. L'image aura été nettoyée, une grille de sudoku aura été extraite puis résolue. Ainsi, l'utilisateur pourra voir toutes les étapes de traitement par lesquelles est passée son image initiale. Cela est possible grâce à deux boutons jumeaux disposés côte à côte. Chacun va progresser d'une étape dans sa direction respective et ainsi faire changer l'image affichée. Cela correspondra à chacune des étapes de traitement, incluant l'image initiale non modifiée (l'affichage de base de ce 'cycle') et la grille résolue.

FIGURE 30 – Les boutons pour visionner chaque étape de l'OCR

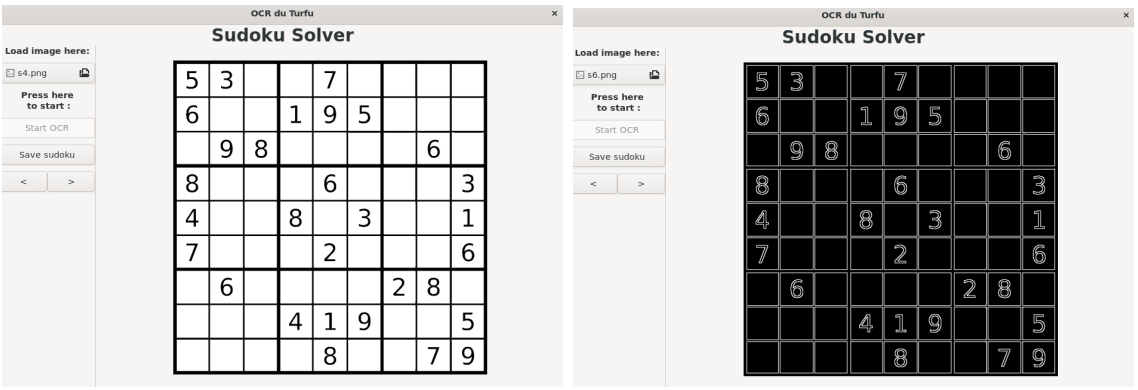
Ainsi, en appuyant sur la flèche droite, on fait défiler les étapes dans l'ordre de la première vers la dernière en se rapprochant de la grille de sudoku résolue. A l'inverse, quand on appuie sur la flèche de gauche, les étapes défilent vers le début, vers l'image initiale. Cependant, pour ne pas désorienter les nouveaux utilisateurs, il n'est pas possible d'accéder à la dernière étape à partir de la première en appuyant sur la flèche gauche, et inversement. En effet, si l'utilisateur observe l'image initiale (la première étape), le bouton de gauche est grisé ; et s'il observe la grille résolue (la dernière étape) le bouton de droite est grisé.

Avant le lancement du programme, ces boutons sont grisés et inaccessibles par logique : on ne peut pas voir des transformations n'ayant pas eu lieu.

Les différentes étapes ainsi visibles contiennent celles déjà expliquées par le traitement de l'image, à la section 4. Les voici dans l'ordre du cycle vers la droite.

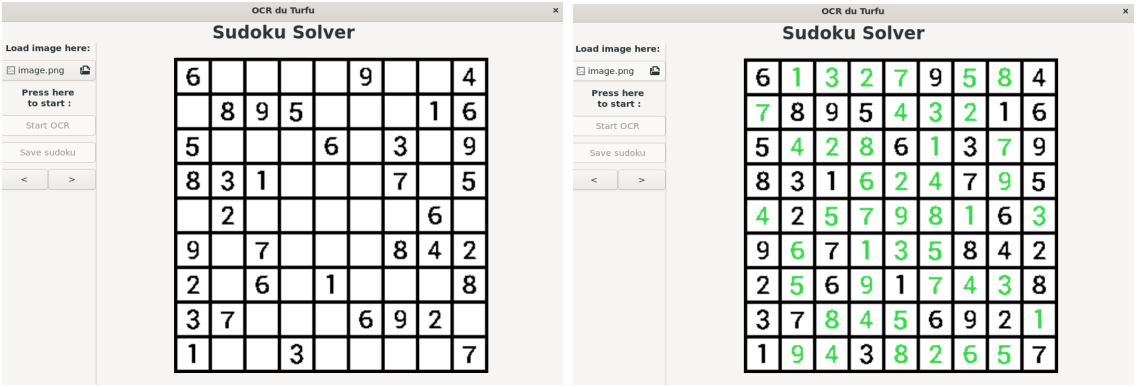
— L'image initiale, inchangée
— Le passage en niveaux de gris
— La modification des ombres et de la luminosité
— La retouche du contraste
— La binarisation : le passage de tout pixel en noir ou blanc

— L'inversion des couleurs
— L'application du philtre Sobel
— La grille extraite et ses chiffres, une image faite de toutes pièces
— Cette même grille mais résolue, avec les chiffres ajoutés aux originaux d'une couleur particulière



(a) Étape de binarisation (b) Étape du philtre Sobel

FIGURE 31 – Deux étapes du prétraitement



(a) Grille non résolue (b) Grille résolue

FIGURE 32 – Les deux grilles extraites : avant et après résolution

7.2.4 Bouton de sauvegarde

Après avoir chargé une image puis lancé l'OCR dessus afin d'obtenir une grille résolue, il faut aussi pouvoir enregistrer ce que l'on a trouvé. C'est l'objectif de bouton

de sauvegarde.

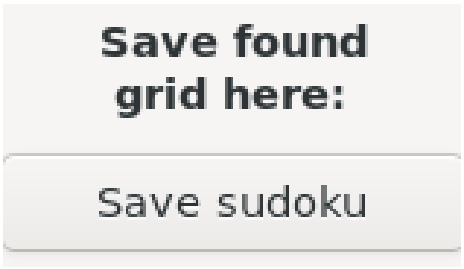


FIGURE 33 – Bouton de sauvegarde

Ce bouton n'est disponible qu'après avoir accompli les étapes de l'OCR avec succès. Sinon, il est grisé. Sa fonction va être d'enregistrer dans un fichier au chemin fixe l'image de la grille résolue, afin de conserver une trace du résultat trouvé. Cependant, on distingue 2 cas :

— La grille trouvée est valide et donc soluble : le fichier enregistré s'appellera *solved.png*.
— La grille trouvée n'est pas valide : le fichier enregistré sera simplement cette grille invalide, et se nommera *unsolved.png*.

De plus, quand on appuie sur le bouton, une petite notification s'affiche dans la sortie standard qui nous informe du chemin du fichier sauvegardé.

Nous avons décidé ce système de chemins arbitraires par sa simplicité. En effet, sauvegarder une grille résolue se résume à un appui de bouton. Nous avons pensé que faire saisir à l'utilisateur un nom de fichier pour sauvegarder serait à la fois trop long, trop compliqué et inutile. Ainsi, les utilisateurs retrouvent toujours leur dernière sauvegarde au même endroit, au même nom.

7.3 Améliorations possibles

Cette application nous plaît bien et nous sommes contents de comment elle s'est développée. Cependant, le manque de temps et le manque d'expérience de nos développeurs ont limité le potentiel de cette interface. Des idées n'ont ainsi pas pu être implémentées, mais cela ne veut pas dire que du temps n'a pas été passé à essayer de les développer, bien au contraire.

7.3.1 Rotation manuelle

Une première idée non implémentée est un système de rotation manuelle. Il aurait consisté d'un slider qui aurait permis de saisir un angle de rotation, et d'un bouton qui aurait lancé le procédé. Cependant, à cause du retard pris par la rotation et par la difficulté relative de l'implémentation des sliders avec GTK, cette idée n'a pas vu le jour.

7.3.2 Images en parallèle

Au départ, quand le système de cycle avec des boutons-flèches n'était pas encore imaginé, nous pensions afficher l'image initiale et la grille résolue côte-à-côte dans l'interface. Nous avons abandonné cette idée tout d'abord par son manque d'efficacité : les images auraient été petites et toutes les étapes n'auraient pas pu être visualisées facilement. De fait, l'idée des boutons-flèches ont rendu cette idée obsolète.

7.3.3 Leaks de mémoire

Nous avons remarqué qu'après exécution, le programme de l'interface graphique laissait derrière lui des leaks en mémoire. Nous pensons que la librairie GTK en est la responsable, car aucun nom de fonction que nous avons implémenté n'apparaît après inspection, mais nous ne savons toujours pas ce qui cause ces légers problèmes.

7.3.4 Divers

Nous avons également eu d'autres idées qui n'ont pas été implémentées, mais pour la plupart car elles comportaient des inconvénients particuliers. Par exemple, pour le cas expliqué en 7.2.4, réaliser un système de parcours de fichiers afin d'en enregistrer un autre aurait été trop compliqué et inutile par rapport à la tâche demandée. Un autre exemple serait de remplacer les boutons-flèches qui font défiles les étapes de l'OCR par des entrées clavier. Cela n'a pas été implémenté car ce n'aurait pas été intuitif, surtout pour un utilisateur qui n'est pas familier avec l'interface.

7.4 Conclusion

En somme, nous sommes très contents de notre application et de comment elle a évolué. Bien qu'assez simple, elle réalise plus que bien ce qui lui est demandé de manière concise, propre et intuitive. Bien que certaines choses auraient pu être améliorées, nous sommes fiers de cette interface graphique.

8 Conclusion

8.1 Conclusions personnelles

8.1.1 Bastien

Pour moi, ce projet a été un challenge conséquent pour ce troisième semestre à l'EPITA. Bien que le projet de jeu vidéo auquel j'ai participé le semestre précédent a été très ambitieux, cet OCR a tout de même été un gros obstacle à surmonter. Mais nous avons quand même réussi à produire quelque chose qui se rapproche assez du projet demandé pour que nous en soyons contents.

Au tout début du projet, juste après avoir reçu les consignes, j'étais assez pessimiste sur l'issue de ce projet. Pour moi, ce qu'on nous demandait était extrême et je ne nous voyais pas du tout capables de réussir ce projet. Cela était surtout dû à mon inexpérience du langage du projet, le C.

Cependant, en commençant à apprendre ce langage et en entamant le projet lui-même, j'ai pris confiance en notre groupe et à sa capacité à mener à bien ce projet. Je dois dire quand même qu'avant la première soutenance, je m'occupais uniquement de la partie de la résolution de sudoku, facile, et de la sauvegarde des poids du petit réseau de neurones, assez simple. Cela m'a permis d'appréhender le C tout en aidant mon groupe, mais je me rends compte à présent que j'aurais pu faire plus.

Ainsi, après la première soutenance, j'avais acquis une assez bonne connaissance du C surtout grâce aux TPs. C'est d'ailleurs le TP sur la librairie GTK qui m'a motivé pour me charger de l'interface graphique. De plus, le réseau de neurones m'intéressait, j'ai donc assisté Paul dans sa réalisation de cela. J'ai ainsi refait un système de sauvegarde des informations du réseau, et je l'ai aussi aidé à résoudre des problèmes récurrents.

Cependant, malgré tous nos efforts, le réseau n'a pas abouti. Comme Paul j'en suis très déçu, car j'ai participé à son développement, et j'aurais aimé le voir marcher. De plus, comme décrit dans 7.3, l'application graphique que j'ai développée aurait pu être améliorée.

Cependant, nous avons accompli bien plus que je ne l'aurai cru possible initialement. Ainsi, pour conclure, je dirai simplement que je suis fier de ce que nous avons accompli malgré ses lacunes et ses problèmes persistants.

8.1.2 Paul

Ce projet de troisième semestre a été une véritable épreuve. L'OCR qui semble à première vue assez simple se révèle être vite complexe.

Lorsque le dossier nous a été donné, j'ai été très surpris du peu d'information qui nous a été donné au niveau du réseau de neurone. Quant au prétraitement, je savais que les autres saurions s'en sortir : il s'agit de la partie la plus guidée. Alors pourquoi avoir choisi la partie du réseau de neurone, celle la plus obscure, me diriez-vous ? Eh bien, j'avais déjà commencé à réfléchir sur un réseau de neurone, enfin, c'était sur un petit réseau type xor, dès les grandes vacances. Cependant, je venais à peine d'apprendre le C et je n'étais pas complètement familiarisé avec. À ce moment-là, je me posais vraiment plusieurs questions sur comment faire. Heureusement, pour un xor, les différentes convergent vers une même source. J'étais très heureux, car le réseau a été fini pour la première soutenance avec l'aide Bastien.

Ensuite, il a fallu passer sur un réseau plus complet, avec d'autres fonctions d'activation et une toute nouvelle structure pouvant contenir les différentes couches et différents nombres de neurones. Je savais comment faire la structure, après, il a fallu connaître comment implémenter les différentes fonctions. C'est la partie la plus complexe, car il y a plusieurs manières de faire une fonction de forward propagation et de backward propagation. Par ailleurs, les sources tendent à se diverger. J'ai rencontré beaucoup de difficultés à les implémenter.

Au final, le réseau existe et calcule, mais il n'apprend pas et je n'ai pas réussi à trouver l'origine de ce bug. Je suis quelque peu déçu car je pensais que j'aurais réussi à la trouver.

8.1.3 Kylian

De mon côté, le projet a été assez compliqué à mettre en place. Avec le nombre de choses différentes à mettre en place. Avec les problèmes de mémoires que j'ai rencontrés réellement pour la première fois. Car certaines solutions que j'ai trouvées prenait trop de place en mémoire, j'ai donc dû trouver d'autres solutions. Mais au final, en supposant que l'utilisateur donne des images propres, il n'y aurait pas de problème.

Même si le projet n'est pas complétement terminé, je trouve que l'avancement n'est pas trop mal. Mais je suis tout de même déçu de ne pas avoir réussi à faire une solution plus efficace de détection de ligne qui permettrait la détection de la grille et la rotation automatiques. Mais sinon je suis content de ce que j'ai pu faire, même si j'aurais aimé avoir plus de temps pour tous finir correctement.

8.1.4 Jacques

Parlons un peu de ce projet. Ça a été un bon challenge pour le troisième semestre à EPITA. Nous n'avons pas réussi à finir le projet et c'est le seul regret que je peux avoir. Je trouve l'idée de ce projet très intéressante, il y a de quoi se répartir les tâches et ça joue sur le travail de groupe.

En parlant de groupe, j'aimerais remercier mes collègues car l'ambiance du groupe était parfaite. Quand il a fallu mettre en commun toutes les parties, nous nous sommes entraidés et avons adaptés pour que tout fonctionne. Pour nos parties respectives, nous avons eu l'occasion de s'aider aussi, je pense notamment à la rotation de l'image réalisée ensemble avec Kylian. En tout cas nous avons eu très peu de divergences et nous avons pu avancer tous ensemble en s'entraidant, ce qui est agréable et donne envie de s'investir.

Concernant le travail effectué, je suis un peu déçu qu'on ai pas rendu un projet complet et qu'il ne fonctionne pas entièrement. Nous avons pu néanmoins mettre en commun certaines parties et nous avons limité la casse en quelque sorte. Pour ce qui est de ma partie, j'ai pu réaliser ce à quoi je m'attendais. Je trouve juste dommage de pas avoir pu traiter la suppression de bruit, qui aurait pu se faire avec plus de temps. Pour le reste, je trouve que le travail fourni et les résultats sont là. C'était un très bon projet, voici les derniers mots de ce rapport.