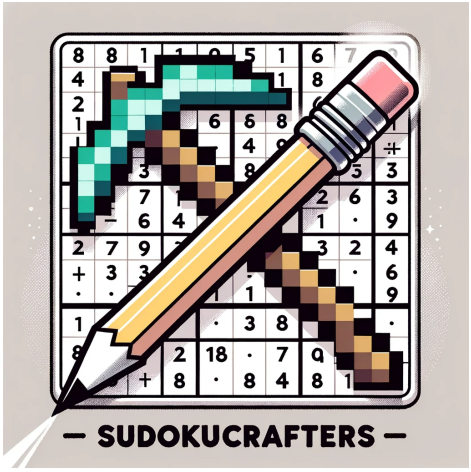


By SUDOKUCRAFTERS

Epita 2023-2024

# Projet OCR

Anthony Lawandos
Lucas Demuliere
Lucil Finkelstein
Lucile Pelou

14 décembre 2023

# Table des matières

# 1 Introduction

Dans le cadre de notre projet informatique de troisième semestre nous devions réaliser une application permettant de résoudre des sudokus à partir d'images. Ce rapport vous présente l'implémentation des différentes fonctionnalités demandées. Vous y trouverez aussi la présentation des membres de *SUDOKUCRAFTERS*.
Pour rappel voici les tâches qui étaient à effectuer lors du premier temps de travail :

— Chargement d'une image et suppression des couleurs
— Rotation manuelle de l'image
— Détection de la grille et de la position des cases
— Découpage de l'image
— Implémentation de l'algorithme de résolution d'un sudoku
— Réseau de neurones capable d'apprendre la fonction OU EXCLUSIF

Et les tâches qui étaient à effectuer lors du second temps de travail :

— Le prétraitement complet
— Le réseau de neurones complet et fonctionnel
— La reconstruction de la grille
— La résolution de la grille
— L'affichage de la grille
— La sauvegarde du résultat
— Une interface graphique permettant d'utiliser tous ces éléments

Tous les éléments cités précédemment sont expliqués tâche par tâche dans ce rapport. Nous avons également implémenté des fonctionnalités supplémentaires que vous découvrirez à la fin de ce rapport. Ce projet regorge de détails, de formules et de problèmes à régler. Tout cela fait de ce projet une découverte et un défi pour nous tous.
Nous espérons que ce rapport vous permettra de découvrir notre groupe et notre vision de ce projet.

# 2 Vue générale du projet

## 2.1 Présentation du groupe

Avant d'entamer la présentation du projet, une présentation du groupe et de ses membres s'impose. Le nom du groupe "SudokuCrafters" fait référence au célèbre jeu vidéo *Minecraft* dans lequel les joueurs ont pour but de "crafter" (fabriquer) toutes sortes d'objets. Dans notre contexte ce sont les chiffres.

**Anthony LAWANDOS** : Je suis originaire du Liban, un pays que j'ai quitté pour poursuivre mes études en France. Là-bas, je me suis découvert une passion pour l'informatique et le sport. Cette dualité entre la technologie et le monde sportif m'a permis de développer des compétences en persévérance et en esprit d'équipe.

Le projet S2 a été une étape marquante dans mon parcours, révélant l'impact de l'informatique dans la résolution de problèmes concrets. C'est donc avec un enthousiasme débordant que je me lance dans le projet OCR, surtout dans le réseau de neurones. Ce défi, qui allie technologie et innovation, promet d'approfondir mes compétences en informatique.

**Lucas DEMULIERE** : Lors de mon stage de découverte en troisième, j'ai été captivé par l'univers de l'informatique, une passion qui s'est renforcée avec mon amour pour le sport, m'enseignant la persévérance et l'esprit d'équipe. Le projet S2 m'a marqué révélant l'impact de l'informatique dans la résolution de problèmes concrets et aussi des différents problèmes. Aujourd'hui, je suis enthousiaste à l'idée de poursuivre avec le projet OCR, un défi qui fusionne technologie et innovation, et promet de développer davantage mes compétences en informatique tout en contribuant à l'avancement technologique.

**Lucil FINKELSTEIN** : Plongée dans l'informatique depuis de nombreuses années, Epita m'a confirmé le fait de vouloir travailler dans l'informatique, surtout en voyant changé notre monde si rapidement. Faire partie d'un monde en plein accroissement est un véritable accomplissement pour moi. De plus, le projet m'a permis de mettre en pratique mes connaissances. Ce nouveau projet est un réel défi pour mon groupe et moi-même. Je suis prête à évoluer, découvrir et apprendre dans le domaine de l'informatique grâce à un projet si complet.

**Lucile PELOU** : J'ai toujours été fasciné par tout ce qu'il était possible de créer avec l'informatique mais venant d'un lycée qui ne possédait pas la spécialité NSI, je n'avais quasiment jamais touché à la programmation en dehors de certains projets personnels. Ma première année en cycle préparatoire à l'EPITA a été enrichissante, notamment grâce au projet de S2, qui m'a permis de découvrir le travail en équipe sur un projet d'envergure et d'acquérir de nouvelles connaissances.

Ce nouveau projet m'a permis d'approfondir mes compétences en programmation en langage C, ainsi que pour d'en acquérir de nouvelles.

## 2.2 Répartition des tâches

Afin de mieux réaliser les différentes tâches demandées pour ce projet nous nous sommes répartis les tâches de cette manière :

| Tâches \| Membres | Anthony | Lucas | Lucil | Lucile |
|---|---|---|---|---|
| **Traitement de l'image** | | | | |
| **Prétraitement** | | | Principal | |
| **Rotation manuelle** | | Principal | Principal | |
| **Rotation automatique** | | | | Principal |
| **Détection de la grille et cases** | | | | Principal |
| **Découpage de l'image** | | | | Principal |
| **Résolution et affichage du résultat** | | | | |
| **Résolution de la grille** | | Principal | | |
| **Interface graphqiue** | | Principal | | Principal |
| **Autres** | | | | |
| **Réseau de neurones** | Principal | | | |
| **Bonus** | | | | |
| **Site internet** | | | | Principal |

TABLE 1 – Tableau de la répartition des tâches au sein du groupe

## 2.3 Moyens utilisés

Pour la réalisation de notre projet, nous avons mis en place plusieurs moyens de collaboration et de suivi. Tout d'abord, nous avons utilisé GitHub afin de travailler en collaboration sur les mêmes fichiers et de résoudre les conflits éventuels.

Nous avons également utilisé Discord pour communiquer, échanger des idées et travailler. C'est aussi sur cette plateforme que nous mettons tous les avancements, les problèmes rencontrés et les choses à faire.

Pour faire notre projet nous avons aussi utilisé le langage de programmation C ainsi que différent éditeurs comme *VIM* ou encore *CLion*.

En ce qui concerne les bibliothèques utilisées, c'est majoriterement les librairies *SDL2*

et *SDL2_image* qui nous ont servi. On peut également citer la librairie math pour utiliser les fonction cosinus, sinus ou encore la valeur absolue. Nous avons aussi eu recours à l'utilisation des librairies tel que *errx* pour la gestion d'erreurs, *stdio* pour écrire dans la console, *stdlib* pour les fonctions d'allocations, *SDL_ttf* pour l'écriture ou encore *stdbool* pour utiliser les types booleans et string pour nous permetttre de manipuler les chaînes de caractères aisément.
Pour la réalisation de l'interface graphique, nous avons utilisé la librairie *GTK3* ainsi que *Glade* pour la concevoir.

## 2.4 Ressenti général

Après un bon départ avec notre première soutenance, nous sommes satisfait de notre bonne continuation pour la soutenance finale. Nous avons réussi à gérer les différents problèmes rencontrés, notamment avec la transformée de Hough qui n'a pas été facile à implémenter au départ. Notre application finale propose toutes les implémentions demandées et même plus. Nous sommes satisfait du résultat et avons appris pleins de nouvelles connaissances en travaillant sur ce projet.

# 3 Prétraitement

Afin d'avoir une image apte pour la détection de la grille et l'analyse des chiffres, nous avons réaliser différentes étapes sur les images pour les rendre plus propre.

## 3.1 Rotation manuelle et automatique

La rotation est importante pour pouvoir donner une image droite à la détection de la grille. Pour ce faire, nous utilisons la librairie SDL2. Cette rotation manuelle est réalisé de cette manière : l'utilisation renseigne son degré de rotation et son image à modifier.

Pour le cas de l'angle, nous voulons travailler avec n'importe quel angle. De ce fait nous le ramenons à une valeur plus compréhensible pour la rotation avec un modulo 360.

Ensuite, nous utilisons les différentes fonctions de la librairie SDL2 et quelques unes de math.h pour pouvoir réaliser cette rotation.

Voici un exemple de ce programme :

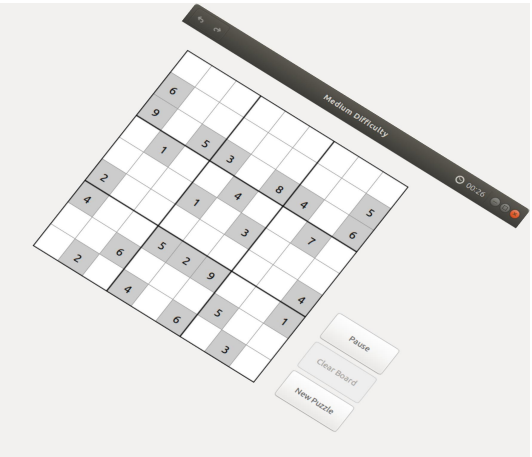


FIGURE 1 – Avant rotation

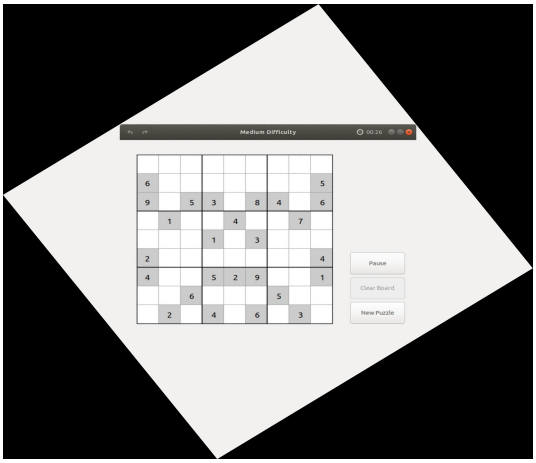


FIGURE 2 – Après rotation de -35 degré

Nous pouvons bien constater que cette rotation est nécessaire afin d'avoir une image droite pour le découpage de la grille et de tout ce qui en suis.

En ce qui concerne la rotation automatique, elle s'opère lors de la détection des lignes, c'est pourquoi son explication sera expliquée dans la partie concernée et non ici.

### 3.2 Suppression des couleurs et Renforcement des contrastes

La suppression des couleurs est tout aussi important pour la suite des étapes. Pour ce faire, nous utilisons la formule de grayscale afin de calculer le niveau de gris d'un pixel :

$$grayscale.pixel = 0.3 * r + 0.59 * g + 0.11 * b.$$

où r,g et b représentent les couleurs du pixel.

À partir de ce résultat, nous pouvons déterminer si le pixel est sombre ou clair ( qui est égal à dire si le résultat est en dessous d'un certain seuil ou non). Si il est sombre, alors il devient noir sinon il devient blanc.
En résultante, nous obtenons une image contrastée (en noir et blanc).

Voici un exemple de ce programme :

FIGURE 3 – Avant application des contrastes

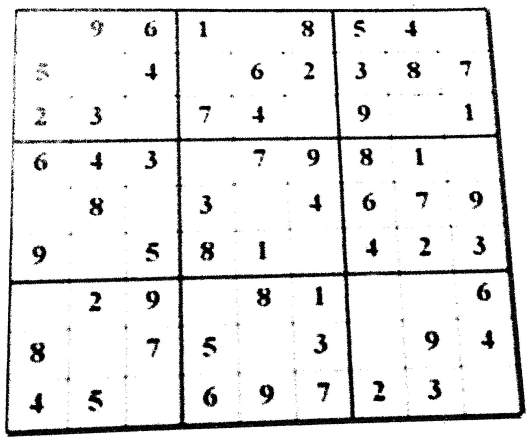

FIGURE 4 – Après application des contrastes

Autre que mettre l'image en noir et blanc, nous pouvons la mettre en gris. Cela est similaire à l'ajustement des contrastes juste que le pixel sera remplacé par son échelle de gris vis-à-vis du calcul ci-dessus.

## 3.3 Eliminations des bruits parasites

L'élémination du bruit se découpe en plusieurs parties : différentes méthodes peuvent être utilisées mais voici celle que nous avons réalisées.

### 3.3.1 Flou sur une image

Tout d'abord, nous allons appliquer un flou sur l'image c'est à dire qu'on va venir lisser cette image. Pour ce faire on va venir récupérer, les 8 pixels qu'il y a autour d'un pixel (et lui même) afin de réaliser une moyenne des couleurs (avec les composantes rouges, vertes et bleu) pour appliquer cette nouvelle couleur sur le pixel central.

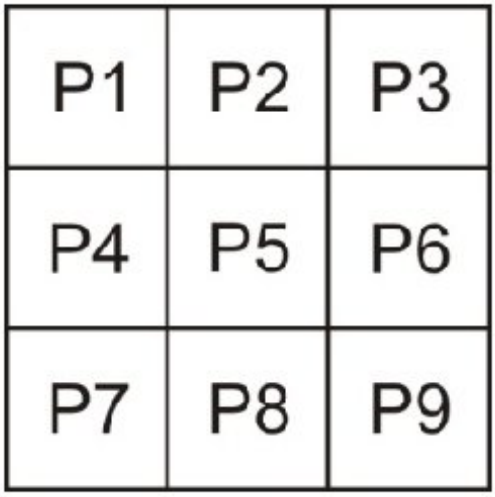


FIGURE 5 – Matrice afin de modifier le pixel 5

### 3.3.2 Réduction du bruit

Après avoir lissé notre image, nous allons réduire son bruit. Le bruit, c'est toutes les variations qui sont souvent aléatoires des valeurs des pixels.
Pour réduire le bruit, nous utilisons la même méthode que le flou sur une image (c'est à dire que nous allons aussi utilisé une matrice de taille 3 lignes, 3 colonnes). Pour se faire, nous allons vérifier la composante rouge du pixel central sur cette dernière est à 0, la valeur renvoyée du pixel sera noir, sinon elle serait blanc.

Enfin avec toutes ces valeurs renvoyées, le pixel est soit déterminé comme du bruit de ce fait, il est mis à jour, sinon il est laissé comme l'original.

Voici un exemple de ce programme :

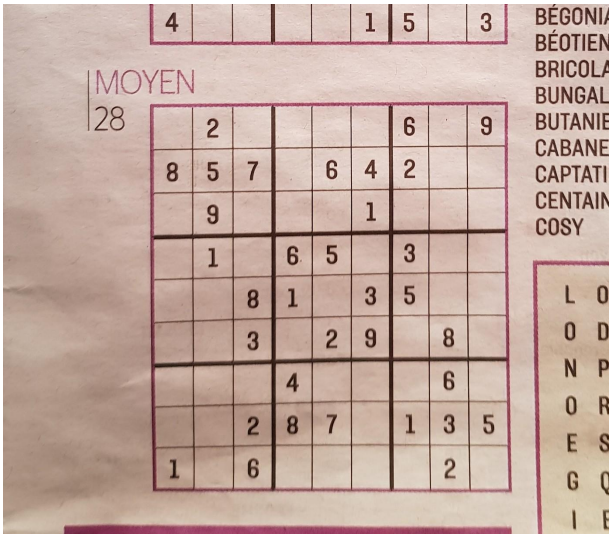


FIGURE 6 – Avant réduction du bruit

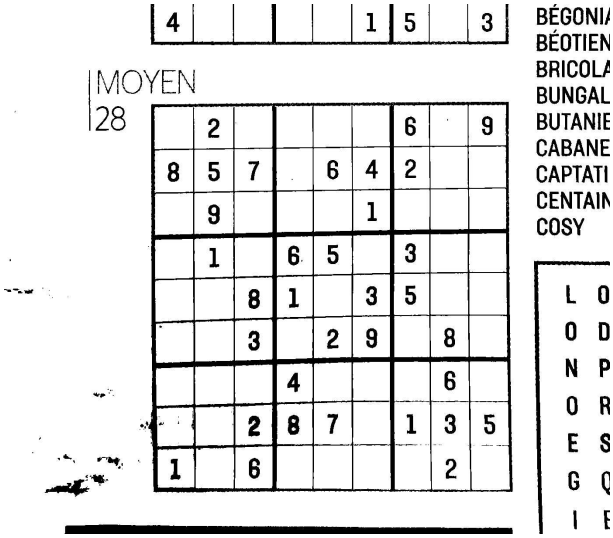


FIGURE 7 – Après réduction du bruit

### 3.3.3 La méthode d'Otsu

La méthode d'Otsu ainsi que son algorithme est utilisée pour effectuer un seuillage automatique à partir de la forme de l'histogramme de l'image. Pour ce faire, nous déterminons un seuil pour séparer les pixels de l'image en deux classes (comme objet et arrières plan). Puis en utilisant l'histogramme de l'image, on calcule la variance entre les deux classes pour chaque valeur de seuil et sélectionne celui qui améliorera la variance. Tout cela permet de binariser l'image en noit et blanc de manière adaptative.

# 4 Détection et découpage de la grille

Lors de la première soutenance nous avons eu du mal à implémenter l'algorithme de la transformée de Hough, c'est pourquoi nous nous sommes tournées vers une autre méthode. Pour la soutenance finale nous avons décidé de combiner les deux méthodes avant d'avoir une meilleure détection.

## 4.1 Détection des lignes

### 4.1.1 L'histogramme

La détections des lignes se déroule en plusieurs étapes. La première est un algorithme d'histogramme qui va nous permettre de détecter des potentielles lignes droite dans une image.

Le principe de l'algorithme est le suivant : il parcourt tous les pixels de l'image en longueur et ajoute leurs composantes r, g et b dans une variable. Si à la fin d'une ligne en longueur, une fois que nous avons parcouru tous les y pour un x, la variable est à plus de la longueur de l'image divisée par 2,5 alors une ligne est détecte. Pour la largeur cela fonctionne de la même manière. Si une ligne est détectée alors nous attribuons sa variable à l'emplacement x ou y (selon si c'est une colonne ou rangée) dans un tableau de int. Une fois cela fait, nous avons deux tableaux, un pour les colonnes et un pour les rangées. Nous pouvons donc savoir l'emplacement des potentielles lignes sur l'image.
Voila les résultats que nous pouvons obtenir si nous dessinons les lignes détectes par notre programme :

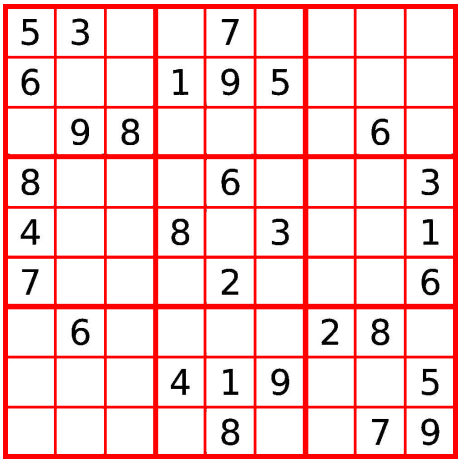


FIGURE 8 – Lignes sur une image parfaite

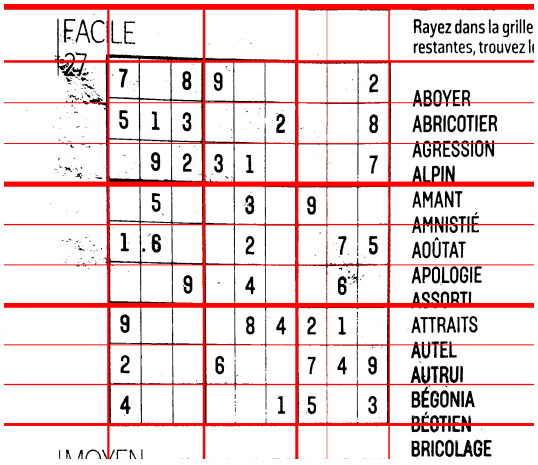


FIGURE 9 – Lignes sur une image déformée

Cette méthode ne permettant pas une très bonne précision pour les lignes n'étant pas droite, nous avons mis en place un second algorithme pour permettre de gérer ce genre de problème.

Après avoir appliqué un traitement qui va regrouper les lignes qui sont dans les mêmes zones, nous vérifions le nombre de colonnes et de rangées. S'ils s'avèrent qu'il n'y en a pas assez (il faut minimum 10 lignes pour les colonnes et 10 lignes pour les rangées) nous appliquons deux méthodes différentes selon le nombre trouvé.

S'il manque peu de lignes nous allons calculer la distance entre chaque ligne consécutive, supprimer le maximum et le minimum et diviser pour obtenir la distance moyenne entre deux lignes consécutives. Au final nous partons de la ligne la plus adaptée et nous vérifions s'il y a une ligne consécutive dans un certain intervalle, calculé avec la distance moyenne, et si ce n'est pas le cas nous la rajoutons. Cette méthode est appliquée pour les colonnes et les rangées séparément. Si il manque beaucoup de lignes, une méthode assez similaire est appliquée, avec quelques différences pour gérer le maximum de cas possibles.

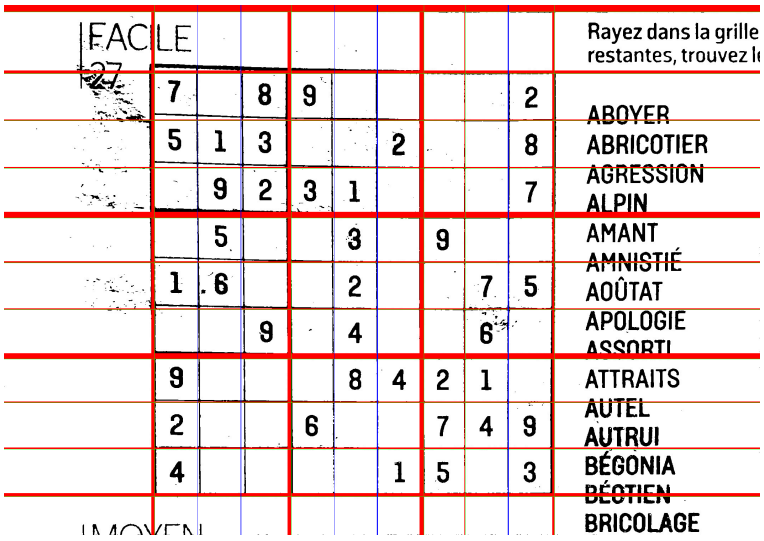


FIGURE 10 – Nouvelles lignes trouvées

Les lignes vertes symbolisent la simplification des lignes déjà trouvées auparavant. Les lignes bleues quant à elles sont celles ajoutées avec le second algorithme.

Une fois ces deux algorithmes effectués, il arrive que nous n'arrivons pas à a un résultat satisfaisant, c'est la que la transformée de Hough, que nous avons réussi à implémenter lors du deuxième temps de travail, intervient.

### 4.1.2 La transformée de Hough

Avant de pouvoir utiliser notre algorithme chargé de faire la transformée de Hough nous devons appliquer un filtre pour détourer les contours de l'image. Pour cela nous avons décidé d'utiliser l'algorithme de Sobel. Cette méthode repose sur la convolution de de chaque pixel sur l'axe x et y avec les noyaux de Sobel. On calcule le gradient pour chaque pixel de l'image qu'on ajoute à une variable total du gradient avec l'horizontal et le vertical séparé.

Ensuite la formule : $\sqrt{Gradienthorizontal^2 + Gradientvertical^2}$ est utilisé pour calculer le gradient total. Une fois ce gradient calculé, nous appliquons une tolérance d'acceptation pour les contours. Si le gradient total passe le test, le pixel est considéré comme un contour et sera coloré en blanc, sinon en noir. A noter que nous avons fait une fonction de binarisation dans le prétraitement pour inverser le noir et le blanc de l'image pour pouvoir appliquer notre algorithme de Sobel.

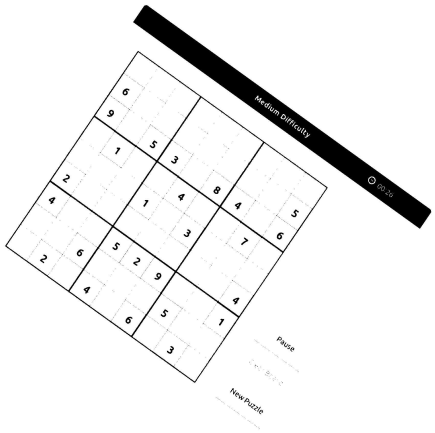


FIGURE 11 – Image après prétaitement sans inversion

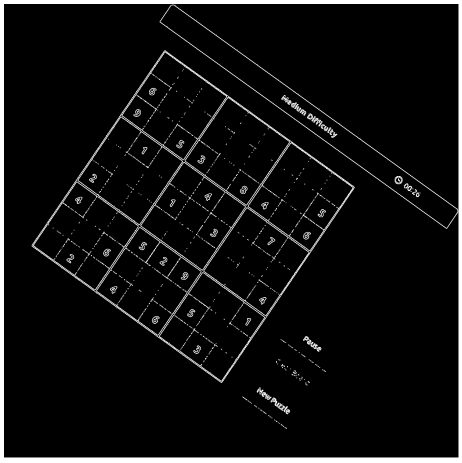


FIGURE 12 – Image après inversion et algorithme de Sobel

Après avoir effectué notre algorithme de Sobel, nous pouvons appliquer la méthode de la transformée de Hough. La transformée de Hough est une technique utilisée en traitement d'images pour la détection des formes géométriques simples, telles que les droites, les cercles ou les ellipses. Elle est particulièrement utile dans les cas où l'image contient des perturbations ou du bruit, et où les méthodes traditionnelles de détection de formes pourraient échouer.
L'utilisation que nous allons faire de cette méthode concerne uniquement la détection des lignes, qui est l'utilisation la plus courante de cette technique :

Pour la détection des lignes, Une ligne droite dans l'espace image (espace des pixels) peut être décrite par l'équation y=mx+b, où m est la pente de la ligne et b son intercept. Cependant, cette représentation n'est pas pratique pour les lignes verticales, car la pente serait infinie. Au lieu de cela, la transformée de Hough utilise une représentation paramétrique des lignes appelée forme normale de Hesse, donnée par l'équation :

$$\rho = xcos(\theta) + ysin(\theta)$$

$\rho$ est la distance perpendiculaire de l'origine à la ligne, et  est l'angle de cette perpendiculaire avec l'axe des abscisses.
Pour l'implémentation de l'algorithme, l'espace de Hough est discrétisé en une grille appelée accumulateur, avec les cellules correspondant aux valeurs possibles de $\rho$ et $\theta$. Pour chaque point de l'image, l'ensemble des cellules de l'accumulateur correspondant à toutes les lignes possibles passant par ce point est incrémenté. Les cellules ayant un nombre élevé d'incréments indiquent la présence probable d'une ligne dans l'image, avec les paramètres ($\rho$ ,$\theta$) correspondant à la position de ces cellules dans l'accumulateur.

Une fois l'accumulateur construit nous allons le parcourir pour récupérer les coordonnées en cartésien. Pour ce faire nous utilisons les formules suivantes :

$$x_1 = x_0 + (d \cdot \cos(\theta)) \qquad y_1 = y_0 + (d \cdot \sin(\theta))$$
$$x_2 = x_0 - (d \cdot \cos(\theta)) \qquad y_2 = y_0 - (d \cdot \sin(\theta))$$

Nous nous retrouvons donc avec les coordonnées des lignes détéctées. Cependant il arrive souvent que cela soit en dehors de l'image, c'est pourquoi nous avons implémenter l'algorithme de tracé de segment de Bresenham pour réaffecter les abscisses et ordonnées de nos lignes. Tout d'abord on itialise quelques variables clés pour faciliter les calculs : dx qui est la différence horizontale entre les points (x2 - x1), dy qui est la différence verticale entre les points (y2 - y1) et p qui est initialement calculé comme 2 * dy - dx.
On itère ensuite sur chaque valeur de x entre x1 et x2, et à chaque étape, on effectue les actions suivantes : 1. On trace le pixel à la position actuelle (x, y), 2. Si p $<$ 0, on met à jour p en ajoutant 2 * dy et sinon, on met à jour p en ajoutant 2 * dy - 2 * dx, et on incrémente y de 1.

Cette méthode a été utilisé pour le tracé des lignes et ajusté pour mettre à jour les points des lignes avant de les ajoutés dans une liste de lignes que nous allons renvoyer une fois l'algorithme de la transformée fini. Pendant le parcours de l'accumulateur nous recherchons également l'angle majoritaire de l'image. Nous créons un accumulateur de taille 90 et nous augmentons la valeur dans la place de l'angle dans l'accumulateur. Tout cela est bien évidemment fait après avoir calculé notre angle en degré et l'avoir modulo 90.

Voila ce que pouvons obtenir avec notre algorithme :

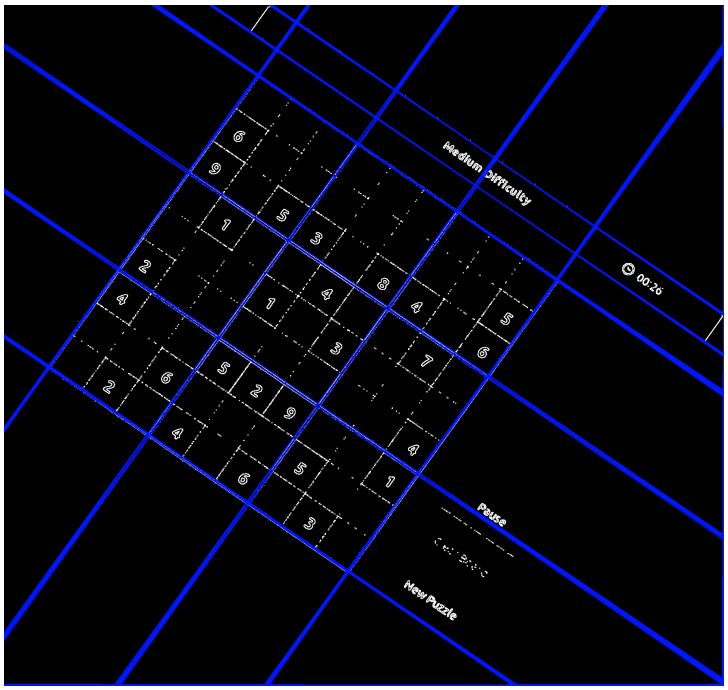


FIGURE 13 – Lignes détectées avec la transformée de Hough

Notre algorithme détecte en même temps l'angle majoritaire qui est ici de -35 degrés. En fonction de la puissance de l'angle majoritaire, nous allons faire une rotation ou non. Pour cela on utilise l'algorithme décrit plus tôt dans la section prétraitement.

Une fois l'image droite, nous pouvons bien détecter les lignes et pour cela nous allons utiliser notre algorithme d'histogramme.

Parfois ce n'est pas une histoire d'angle mais de malformation de l'image. Ce ne serait pas un problème si toutes les lignes étaient détectés mais ce n'est pas toujours le cas. Un algorithme va donc replacer les lignes manquantes aux besoins de sorte à qu'elles soient colinéaires aux autres lignes de sa rangée. Pour cela nous avons également rajouté un attribut à notre structure de lignes afin de savoir si elles sont verticaux ou horizontaux.

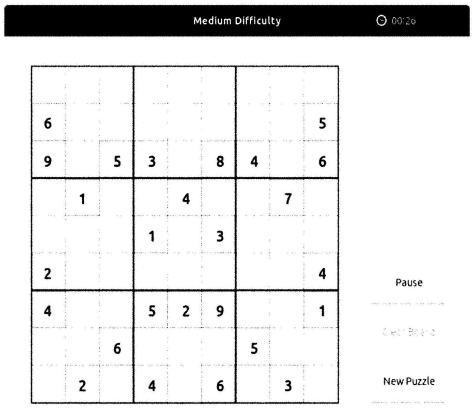


FIGURE 14 – Image après rotation

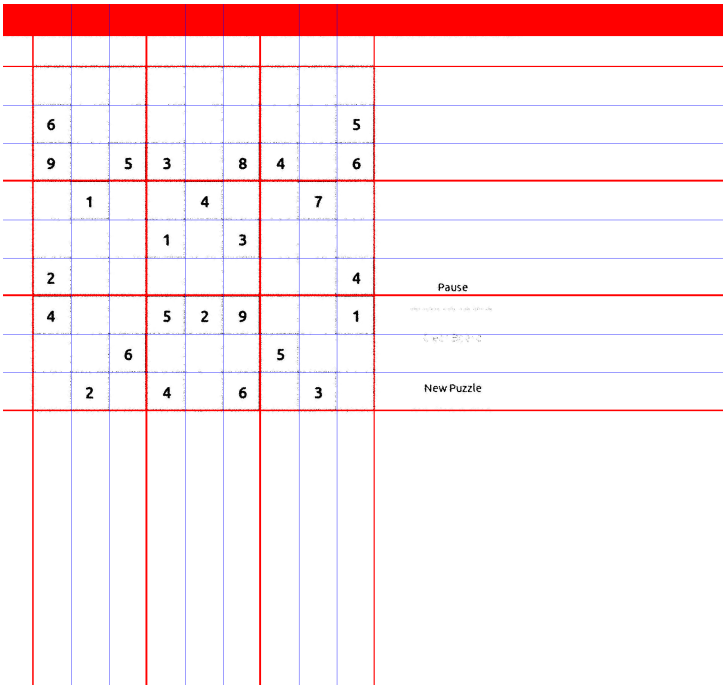


FIGURE 15 – Détection des lignes sur la nouvelle image

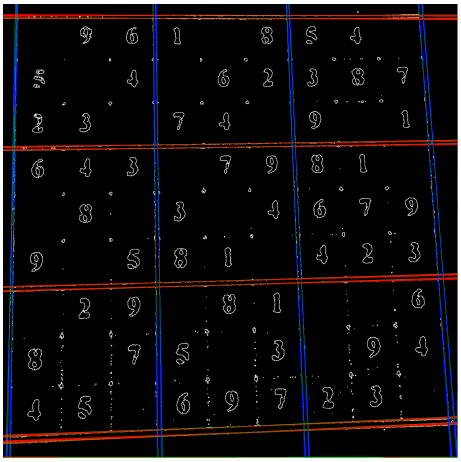

FIGURE 16 – Après la transformée de Hough

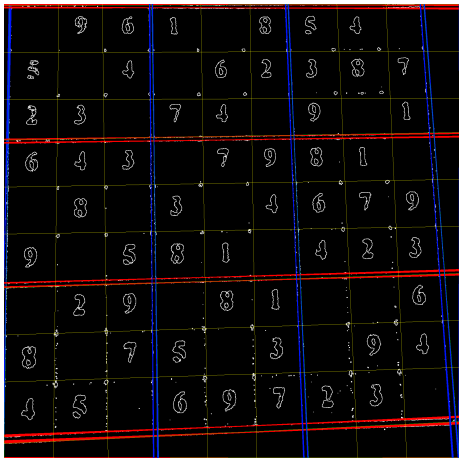

FIGURE 17 – Lignes ajoutées en marron

Nous avons réussi à finir l'implémentation de l'algorithme de la tranformée de Houghde manière opérationnelle pendant ce deuxième temps de travail.

Ces deux méthodes nous permettent un taux de fiabilité beaucoup plus élevé pour la détection des lignes, cependant il arrive encore qu'elle ne soit pas bien détectée. C'est pourquoi après la transformée nous appliquons un algorithme similaire à celui décrit plus haut pour simplifier les lignes déjà existantes et ajouter celles manquantes.

### 4.2 Détection de la grille et des cases

Une fois les lignes détectées et simplifiées, il ne nous en reste plus beaucoup. Nous pouvons donc appliquer un algorithme permettant de récupérer tous les carrés de l'image. Pour cela nous allons parcourir toutes les lignes trouvées et regarder toutes lignes avec lesquels elles ont une intersections en communs. Ces lignes seront parcouru et nous appliquerons la même méthode que les lignes précédentes tout en prenant soin de ne pas les retester avec celles-ci. Nous appliquons cela jusqu'à obtenir 4 lignes et 4 points d'intersection distincts.

Pour retrouver si deux lignes ont une intersection, nous nous basons sur leurs orientations. Chaque fois que nous trouvons un carré, nous vérifions si un carré aux coordonnées similaires n'a pas été ajouté et si c'est le cas, nous ne l'ajoutons pas. Pendant ce processus nous calculons aussi le carré avec la taille de la plus grande, c'est ce qui va nous permettre de détecter la grille de Sudoku.

Une fois la récupération des carrés terminés et la position de la grille connue, nous éliminons tous les carrés qui ont au moins un point en dehors de celle-ci, avec une

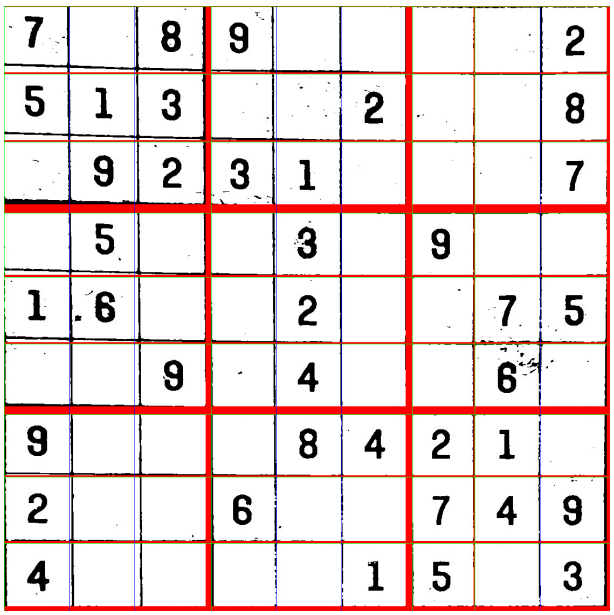


FIGURE 18 – Grille trouvée pour le sudoku de la figure 4

limite de tolérance selon la taille de l'image.

Plus nous avançons, plus nos attentes s'affinent et plus notre liste de carrés se rétrécit, mais elle reste encore assez importante. Un autre algorithme va permettre de récupérer uniquement les cases de la grille. Il va calculer la taille des 10 plus petits carrés de notre liste, enlever le minimum et faire une moyenne. Ensuite nous récupérons tous les carrés qui répondent à nos nouveaux critères, il faut que leur taille soit comprise dans un certain intervalle, calculé avec la taille moyenne des plus petits carrés.
C'est enfin que nous obtenons notre liste finale contenant uniquement les cases de la grille. Pour finir nous les trions et les découpons afin de les enregistrer en image .jpeg faisant une taille de 28 pixels x 28 pixels.

La suppression des résidus a été retravaillé pour la deuxième soutenance. Un algorithme parcourt tous les contours de l'image et supprime tout ce qui pourrait s'apparenter à des lignes sur les bords. Cela nous assure un meilleure découpage et surtout de ne pas découper des nombres qui seraient collé au bords.

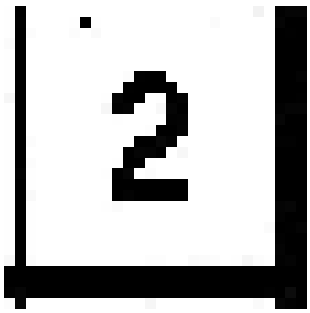

FIGURE 19 – Case coupée avec résidus

FIGURE 20 – Suppression des résidus

# 5 Réseau neuronal

## 5.1 Mini réseau

Pour commencer le réseau de neurones, nous avons décidé d'implémenter un réseau séparé qui comprend la fonction XOR pour nous aider à bien comprendre les bases et le fonctionnement du réseau de neurones. Ce réseau est découpé en 3 parties : propagation avant, rétropropagation et enfin la sauvegarde et affichage des résultats. Bien que nous ayons réussi à implémenter ce mini réseau nous avons constaté qu'il n'est pas efficace pour implémenter le réseau principale vue que nous utilisons des matrices pour représenter le réseau et les différentes couches surtout si on désire augmenter le nombre des couches cachés. Donc on voulait faire un réseau plus structuré. En revanche, ce mini réseau nous a beaucoup aidé à comprendre le fonctionnement d'un réseau de neurones, la propagation, l'activation des neurones et la diminution de l'erreur.

## 5.2 Le réseau principal

### 5.2.1 Structure globale

Pour le réseau principal, nous avons choisi d'implémenter une structure classique mais bien claire et découpée. Pour faire cela et éviter de travailler avec de grandes matrices, nous avons créé trois structures représentant les éléments fondamentaux d'un réseau de neurones :

— Le réseau global composé d'une liste de couches et caractérisé par le nombre de couches, le nombre d'entrées, le nombre de neurones par couche cachée, et enfin le nombre des sorties.
— La couche (layer) composée d'une liste de neurones et ainsi caractérisée par le nombre d'éléments dans cette liste.
— Le neurone, qui est caractérisé par le nombre de poids entrant dans le neurone (stockés en plus dans une liste), la valeur du neurone ainsi que son biais et le delta, qui nous sert à calculer l'erreur.

Ensuite le réseau est composé de différentes parties : initiation du réseau, propagation avant, rétropropagation, gradient descent et la sauvegarde des résultats.

#### 5.2.2 Initialisation du réseau

Cette partie est découpée en deux.

Création du réseau :

Notre réseau principal est composé de trois couches :
— La couche d'entrée est constituée de 784 neurones, correspondant à chaque pixel des images (28*28), où les pixels noirs constituant les chiffres sont représentés par un 1 et les pixels blancs par un 0.
— La couche cachée est constituée de 100 neurones.
— La couche de sortie est constituée de 9 neurones représentant les chiffres que l'on peut rencontrer dans les sudokus (1-9).

Initialisation du réseau :

Cette étape attribue des valeurs aléatoires aux poids et aux biais des neurones selon la formule d'initialisation suivante : (rand ()) / ((RAND MAX) / 2.0) - 1.0. Ainsi, leurs valeurs sont comprises entre -0.5 et 0.5.

Il est également possible d'initialiser le réseau avec des valeurs déjà sauvegardées.

### 5.2.3 Propagation avant :

La fonction de propagation avant (forward propagation) est cruciale pour le fonctionnement de notre réseau de neurones. Son rôle est de calculer les valeurs de chaque neurone dans le réseau en utilisant une fonction d'activation.

L'algorithme de la propagation avant commence par attribuer les valeurs d'entrée aux neurones de la couche d'entrée. Ensuite, il calcule les valeurs des neurones pour les couches cachées et la couche de sortie en utilisant une formule. Pour chaque neurone, l'algorithme calcule d'abord une activation en ajoutant le biais, puis en effectuant une somme pondérée des poids des neurones de la couche précédente.

Afin d'activer les neurones, nous utilisons la fonction sigmoïde sur la valeur obtenue par le calcul pour ramener la valeur des neurones entre 0 et 1. La formule de la fonction sigmoïde est la suivante :

$$f(x) = \frac{1}{1 + \exp(-x)}.$$

Nous appliquons cette formule sur les couches cachées seulement (ici une couche cachée). Cependant, étant donné que nous avons plusieurs sorties, nous avons remplacé la formule de calcul par la fonction softmax (seulement sur la dernière couche), qui divise les valeurs obtenues en probabilités pour une meilleure précision.

Nos formules et calculs fonctionnaient très bien, et nous obtenions de bonnes précisions. Cependant, pour être plus efficaces et après des recherches approfondies, nous avons adopté une autre méthode d'activation des neurones, la méthode RELU (Rectified Linear Unit), qui renvoie la même valeur si x est positive et 0 sinon. Lorsque cette méthode a été bien implémentée, nous avons obtenu des meilleures précisions par rapport à celles obtenues avec la fonction sigmoïde, et ce, en beaucoup moins de temps.

### 5.2.4 Rétropropagation

La rétropropagation est une fonction cruciale dans l'apprentissage de notre réseau de neurones. Elle permet au réseau de comprendre ses erreurs et d'ajuster ses poids en conséquence. Pour ce faire, elle utilise la dérivée de la fonction sigmoïde, qui est donnée par la formule :

$$f'(x) = x * (1 - x)$$

Lors de la rétropropagation, le réseau compare les valeurs de sortie calculées avec les valeurs cibles (attendues) et calcule l'erreur pour chaque neurone de la couche de sortie. L'erreur est ensuite propagée en arrière dans le réseau, couche par couche, en utilisant les poids des connexions. Cette propagation de l'erreur permet de calculer les deltas pour chaque neurone.

En adaptant la méthode RELU, sigmoidPrime a été remplacé par la derivé de la fonction RELU qui donne 1 si la valeur du neurone est positive et 0 sinon, conformément à la nature de la fonction RELU.

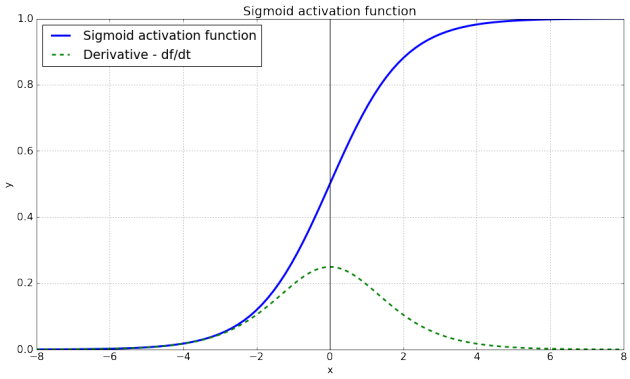


FIGURE 21 – Fonction représentant la fonction sigmoïde ainsi que sa dérivée

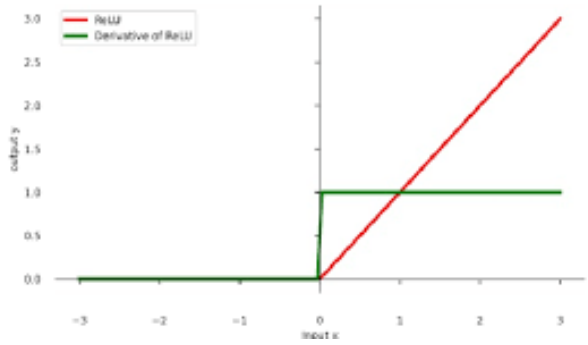


FIGURE 22 – Fonction représentant la fonction RELU

### 5.2.5 Descente de gradient

Les valeurs des deltas calculées par la rétropropagation sont utilisées pour ajuster les poids des connexions lors de la mise à jour du réseau. Les poids des neurones sont mis à jour en fonction de leurs valeurs de delta, des valeurs des neurones de la couche précédente et d'un taux d'apprentissage.

### 5.2.6 XOR

En implémentant le nouveau réseau, nous avons paramétré le réseau pour qu'il puisse prendre en compte différentes valeurs de l'opérateur XOR (ou exclusif) avec leurs sorties attendues correspondantes :
Inputs : [0, 0], [0, 1], [1, 0], [1, 1]
Expected Outputs : 0, 1, 1, 0 Pour obtenir les meilleurs résultats, nous avons choisi

de configurer le réseau avec une couche cachée composée de deux neurones. Cette structure permet au réseau de capturer les relations complexes entre les entrées et de générer des sorties correspondant au comportement de la fonction XOR. Les poids et les biais du réseau sont adaptés au fur et à mesure de l'apprentissage, ce qui lui permet de converger vers une approximation de la fonction XOR. Cette configuration de réseau est capable de résoudre le problème du XOR avec une grande précision.

La fonction XOR n'utilise pas softmax vu qu'il y a juste une seule sortie.

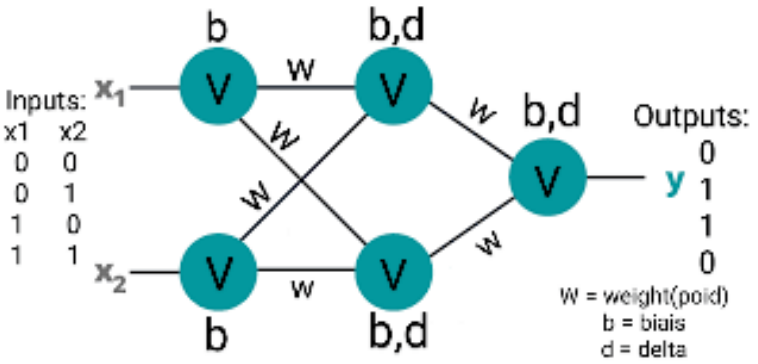


FIGURE 23 – Représentation du réseau de neurones pour la fonction XOR

### 5.3 Bibliothèque d'images

Pour entraîner notre réseau, nous avons besoin d'une bibliothèque d'images présentant différents chiffres écrits avec différentes polices, et incluant également quelques défauts. Pour cela, nous avons utilisé des bibliothèques trouvées sur internet et avons tout sauvegardé dans un fichier texte. Dans ce fichier, le chiffre représentant l'image est le premier caractère de chaque ligne, et le reste des pixels représente le reste de la ligne (1 pour les pixels noirs et 0 pour les pixels blancs).

De même, nous avons créé nos propres fonctions pour construire nos propres bibliothèques ou ajouter des images aux fichiers que nous avons déjà.

Pour réaliser cela, nous suivons les étapes suivantes :

— Chaque image est transformée en une liste de 784 éléments représentant les 28*28 pixels de l'image. Comme mentionné précédemment, les pixels noirs sont représentés par des 1 et les pixels blancs par des 0.
— Chaque image est caractérisée par le chiffre qu'elle représente.
— Avec les informations que nous avons, nous ouvrons le fichier où nous souhaitons stocker les données. Ainsi, dans la première ligne vide du fichier, nous écrivons le chiffre associé à l'image comme premier caractère, puis nous plaçons le contenu de la liste des pixels sur le reste de la ligne.

## 5.4 Entraînement

### 5.4.1 Déroulement

Après avoir implémenté les fonctions principales du réseau et créé le fichier contenant les bibliothèques d'images, nous sommes arrivés à l'étape délicate du fonctionnement de notre réseau de neurones : l'entraînement.

Avant de commencer l'entraînement, il y a des étapes importantes que nous suivons :

— Tout d'abord, nous initialisons notre réseau de neurones avec 3 couches (couche d'entrée de 784 neurones, couche cachée de 100 neurones et la couche de sortie de 9 neurones). Nous attribuons des valeurs aléatoires aux biais et aux poids (valeurs comprises entre -0.5 et 0.5).
— Nous choisissons si nous voulons remplacer les valeurs des poids par des valeurs déjà sauvegardées.
— Nous définissons le taux d'apprentissage du réseau. Nous avons choisi un taux d'apprentissage de 0.01.
— Nous précisons le nombre d'entraînements que nous voulons effectuer.
— Nous indiquons le fichier où sont stockées les images de la bibliothèque sur laquelle nous souhaitons entraîner le réseau.

Après cela, voici comment se déroule l'entraînement de notre réseau :

— Pour chaque ligne du fichier, nous remplissons la couche d'entrée avec les pixels de l'image, et tous les neurones de la couche cachée sont positionnés à 0, sauf la valeur du neurone qui stocke la probabilité de la valeur du chiffre associé à l'image, représenté par un 1.
— Puis, pour chaque donnée, nous lançons la propagation avant, la rétropropagation ainsi que la descente du gradient.
— Pour observer l'avancement du réseau, nous imprimons le pourcentage de succès après chaque epoch. Ce pourcentage est calculé comme suit : pour chaque donnée dans la bibliothèque, nous calculons la valeur donnée par le réseau (la probabilité la plus grande), puis selon les valeurs obtenues, nous calculons le taux de réussite en suivant la formule :

$$\frac{numberofcorrectPrediction}{dataSize} \times 100$$

Après l'entraînement, nous comparons le pourcentage de succès du réseau avant et après l'entraînement. Si ce pourcentage a augmenté, nous sauvegardons les nouvelles valeurs des poids.

### 5.4.2 Résultat de l'entraînement

En adaptant le réseau avec la fonction sigmoïde et sa dérivée, nous avons obtenu de très bonnes précisions, mais cela prend du temps. En revanche, en adaptant le réseau avec la méthode RELU et sa dérivée, nous avons obtenu des précisions qui ont dépassé nos attentes en moins de temps. Donc, nous avons choisi de continuer avec cette méthode.

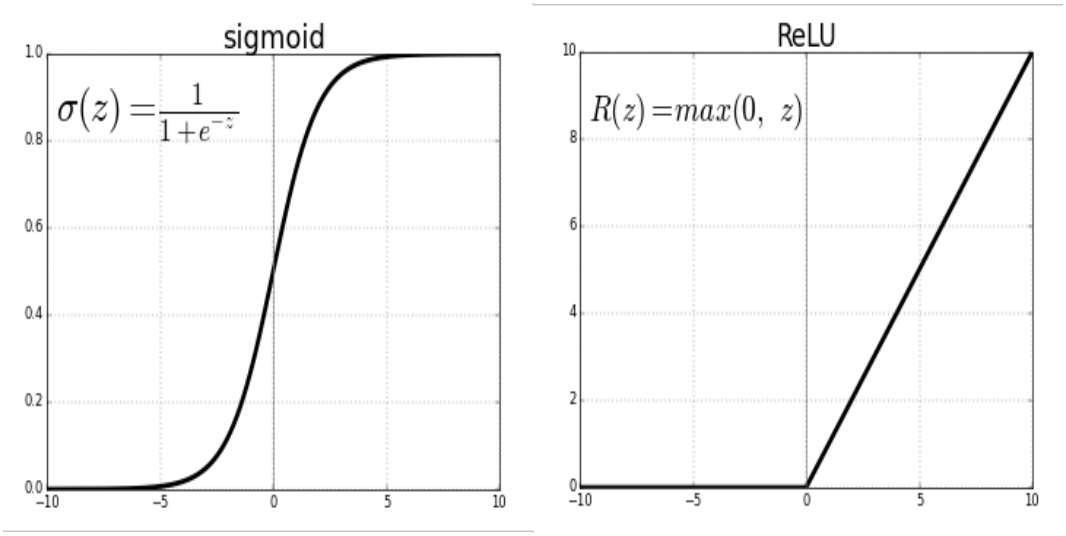


FIGURE 24 – Comparaison des deux méthodes utilisées

## 5.5 Création du sudoku à partir du réseau

Après avoir entraîné le réseau, il est temps de l'adapter avec le reste des étapes de la résolution du sudoku. Pour cela, nous suivons les étapes suivantes :

— Sélection de l'image du sudoku.

— Application d'un prétraitement complet sur cette image. À la fin du prétraitement, nous obtenons les 81 images représentant toutes les cases du sudoku.

— Analyse des images obtenues :
Chaque image est analysée individuellement. Nous vérifions d'abord si l'image contient un chiffre en comptant le nombre de pixels noirs. Si le nombre de pixels noirs est suffisant pour constituer un chiffre, nous procédons aux étapes suivantes du réseau. Cela inclut la transformation de l'image en une liste, l'assignation des valeurs dans la couche d'entrée et la récupération du résultat.

Le résultat obtenu est ensuite écrit dans un fichier texte en fonction de sa position dans le sudoku.

Si l'image ne représente pas un chiffre, nous renvoyons un point pour permettre au solveur de résoudre le sudoku.

À la fin de ce processus, nous obtenons un fichier dans lequel la grille de sudoku est prête à être résolue et affichée à l'utilisateur. Ce fichier contient les valeurs prédites par le réseau pour chaque case du sudoku, en se basant sur l'analyse des images traitées.

# 6 Résolveur

Pour résoudre un puzzle de Sudoku, nous adoptons une approche récursive en utilisant le "backtracking".

## 6.1 Solubilité du Sudoku

La première étape consiste à vérifier l'absence d'incohérences qui pourraient rendre le puzzle de Sudoku insoluble. Une incohérence peut être, par exemple, la présence de deux chiffres identiques sur la même ligne, dans la même colonne ou dans la même région 3x3. Si une telle incohérence est détectée, le programme signale une erreur, indiquant que le Sudoku est impossible à résoudre.
Dans le processus de résolution, si le programme parcourt toutes les possibilités sans trouver de solution viable, il renvoie également une erreur. Cette situation signifierait que le puzzle est insoluble avec l'état actuel des entrées.

## 6.2 Résolveur

Le résolveur utilise une technique de "backtracking" pour résoudre le Sudoku. Le backtracking est une méthode de force brute qui teste systématiquement toutes les combinaisons possibles jusqu'à ce qu'une solution valide soit trouvée. Ce processus implique les étapes suivantes :

— Choisir un emplacement vide dans la grille de Sudoku.
— Essayer tous les chiffres de 1 à 9 dans cet emplacement.
— Pour chaque chiffre, le programme vérifie si cela conduit à une configuration valide du Sudoku.
— Si une configuration valide est trouvée, le programme continue récursivement et passe à la case suivante.
— Si aucune configuration valide n'est possible, le programme annule le dernier chiffre inséré (d'où le terme "backtracking") et essaie une nouvelle valeur.
— Ce processus continue jusqu'à ce que toutes les cases soient remplies correctement.

### 6.3 Sauvegarder des Résultats

Après avoir résolu le Sudoku, nous avons deux méthodes pour enregistrer le résultat :

— **En tant qu'image** : La grille de Sudoku résolue est sauvegardée sous forme d'image, avec une mise en évidence des chiffres qui ont été ajoutés par le résolveur. Cela permet de visualiser facilement les modifications apportées au puzzle initial.

— **En tant que fichier** : La grille résolue est également sauvegardée sous une forme qui peut être lue par l'exécutable du résolveur. Cela pourrait être un fichier texte avec une certaine structure ou un format de fichier spécialisé qui représente la grille de Sudoku.

En résumé, le programme de résolution de Sudoku travaille méthodiquement pour remplir la grille tout en respectant les règles du jeu, enregistre les solutions trouvées et signale des erreurs lorsque le puzzle ne peut pas être résolu en raison d'incohérences ou de contradictions.

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

FIGURE 25 – Rendu final

# 7 Interface graphique

Dans le cadre de notre projet nous devions développer une interface graphique capable de faire fonctionner notre résolution de sudoku à partir d'une image ainsi que d'autres fonctionnalités demandés. Nous avons utilisé la librarie GTK3 et l'outil de conception glade pour que cela plus facile à développer. Cela nous a permis de développer une interface qui assure une expérience à l'utilisateur agréable et intuitif.

Glade nous a permis d'accélérer le processus de développement et car les modifications de l'interface peuvent être réalisées rapidement et facilement.

Notre approche est d'abord de créer une maquette avec Paint. Cela nous a permis d'avoir une vision claire de l'aspect souhaité pour l'interface avant de commencer l'implémentation réelle. Cette étape de planification est cruciale pour s'assurer que l'interface finale répond à nos besoins et nos attentes.

Nous avons également intégré un style css à notre interface afin de la rendre plus attrayante et en accord avec l'identité visuelle de notre application.

En résumé, le développement de l'interface avec GTK3 et Glade, en commençant par une maquette simple et en intégrant des styles CSS, est une approche solide.

Notre interface possède trois pages que nous gérons avec des GtkStack. Chaque page est une GtkGrid pour nous permettre de placer plus facilement les différents widgets.

## 7.1 Page principale

La première page sur laquelle l'utilisateur va arriver est la page d'accueil de notre application. Il va pouvoir accéder au reste des pages et à toutes les fonctionnalités depuis celle-ci.

Nous avons tout d'abord placé une GtkBox possédant 5 boutons et le logo. Le premier bouton va permettre à l'utilisateur de charger une image et de l'afficher dans l'application. Pour l'afficher nous avons placé un GtkFixed, afin de pouvoir gérer plus facilement les dimensions de l'image chargée par l'utilisateur.

Le deuxième bouton va permettre à l'utilisateur d'accéder à la deuxième page de notre application, le traitement d'images. Nous allons revenir dessus dans le section concernée. Ensuite nous avons le boutons qui va permettre de résoudre le sudoku de l'image chargé et d'afficher le résultat.

Il arrive que notre détection du sudoku ne soit pas exacte, dans ce cas un quatrième bouton va permettre à l'utilisateur d'accéder à la troisième et dernière page de notre application. De là il pourra modifier les nombres trouvé pour la grille chargée et réessayée de la résoudre. Nous allons revenir dessus plus tard.

Et enfin il y a également un bouton permettant de quitter l'application.

Voila à quoi ressemble cette première page :

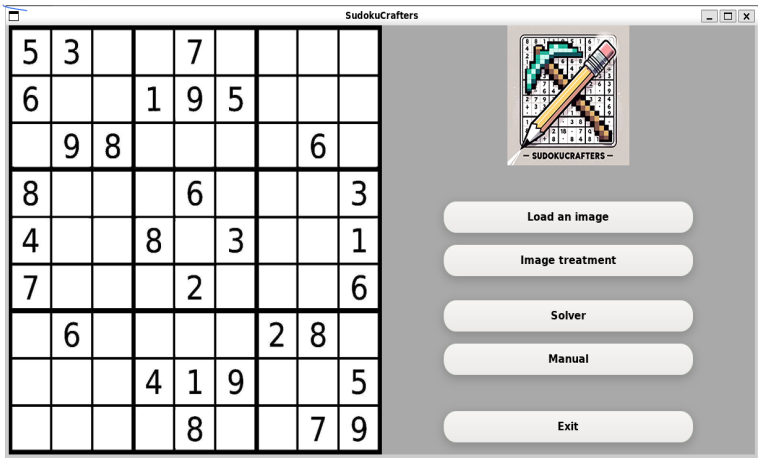


FIGURE 26 – Page principale de l'application

## 7.2 Page pour le traitement d'image

Quand l'utilisateur clique sur le deuxième bouton de la page principale, il peut accéder à la deuxième page qui est consacrée au traitement d'images. De là il va pouvoir effectuer tous les traitements séparément ainsi que celui au complet.

Comme pour la première page, nous avons placé une GtkBox pour placer les différents boutons. Le premier est consacré à la rotation automatique, elle va s'effectuer si besoin il y a. Le deuxième permet à l'utilisateur de mettre l'image en niveau de gris. Ensuite il y a un bouton qui va lui permettre de faire tout le processus de prétraitement.

Le bouton suivant permet de remettre l'image à 0, avant application des différents processus, que cela soit la rotation automatique, le grayscale ou encore le prétraitement complet. Un bouton sauvegarde est ensuite là pour comme son nom l'indique sauvegarder l'image actuellement traité. Et enfin un bouton pour retourner à la première page.

En plus de ces boutons, un GtkAdjustment va permettre à l'utilisateur de faire une rotation de l'image sur 360 degrés en temps réel. Tous les traitements apportés par les différents boutons vont être visible sur l'image qui va être placée à gauche des boutons dans un GtkFixed.

Voici à quoi ressemble la deuxième page après chargement d'une image et application d'une rotation ainsi que du prétraitement complet :

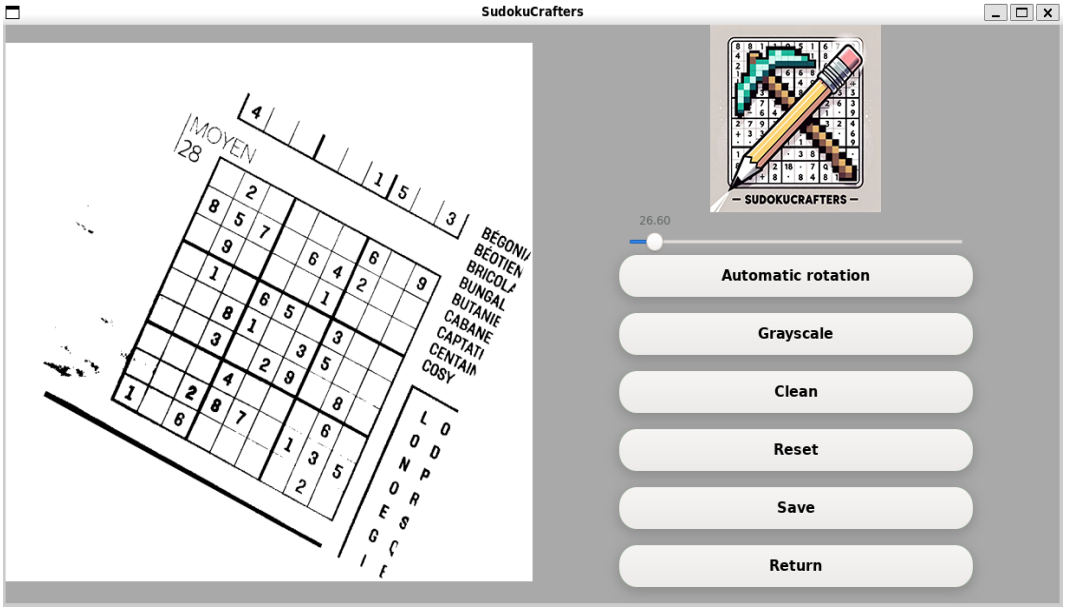


FIGURE 27 – Page sur le traitement d'image

Il est à noter que si l'utilisateur ne sauvegarde pas l'image sur lequel il a appliqué différents traitement et qu'il retrourne à la page principal, c'est l'image chargée d'origine qu'il va retrouver. Et s'il retourne sur la page du prétaitement, il va retrouver son brouillon, l'image avec les traitements qu'il a appliqué.

## 7.3 Page pour résoudre les éventuels problèmes

Quand l'utilisateur lance la résolution d'une grille de sudoku il arrive que notre résolution de soit pas exacte. C'est pourquoi nous avons mis en place cette dernière page lui permettant de modifier les chiffres par lui-même.

Tous les chiffres trouvés par notre application vont être affichés dans des cases qu'il va pouvoir modifier. Une fois les modifications effectuées il va pouvoir cliquer sur le bouton "try" qui va essayer de résoudre le sudoku modifié par l'utilisateur. La solution s'affichera sur la page principale s'il y en a une.

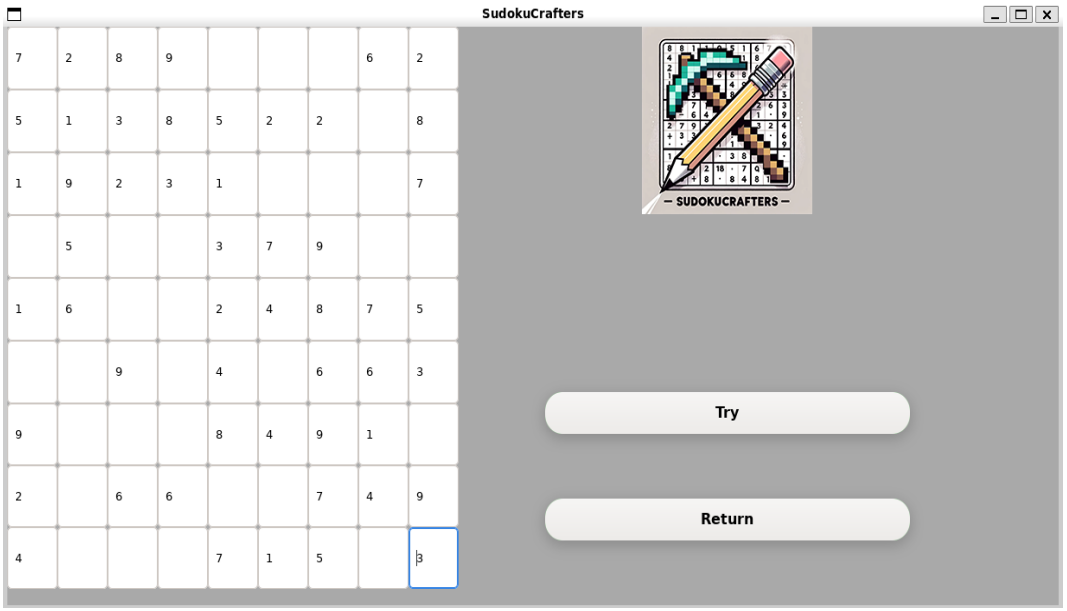


FIGURE 28 – Page manuelle

# 8 Site internet

La réalisation d'un site internet pour le projet était un bonus que nous avons décidé de faire car nous trouvons que c'est toujours bien d'avoir un site pour présenter un projet. Nous avons fait un site de A à Z en se basant sur du HTML et du CSS, il est entièrement dynamique. On peut y retrouver des informations concernant les membres du groupe ainsi que les outils utilisés. Tous les documents peuvent aussi être téléchargés avec l'application de notre projet.


FIGURE 29 – Page principale du site web

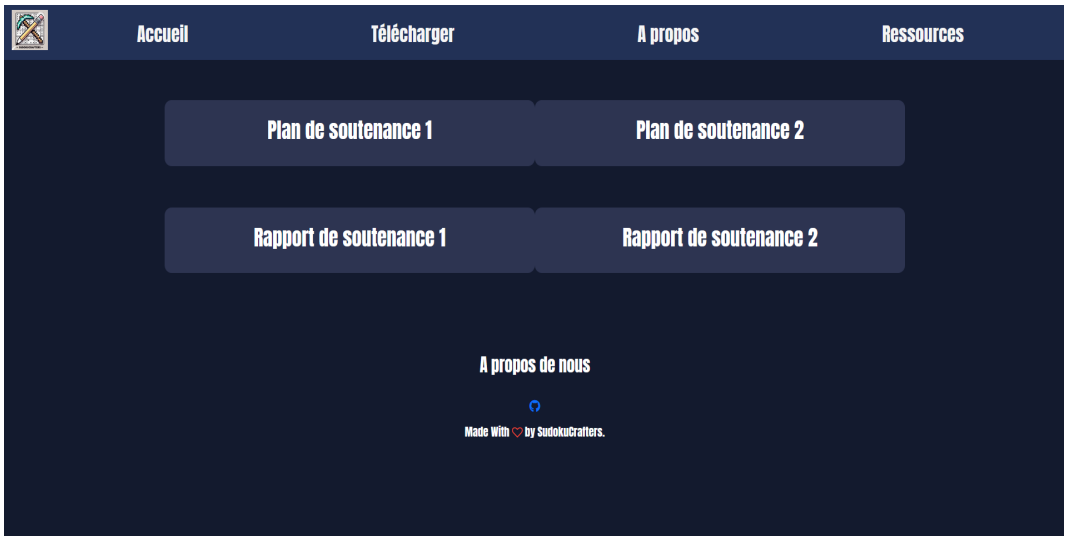


FIGURE 30 – Page des ressources

# 9 Bilan

Voici un tableau représentatif de nos avancées et de nos réalisations.

| Tâches | Réalisé | Prévu |
|---|---|---|
| **Traitement de l'image** | | |
| **Prétraitement** | 100% | 50% |
| **Rotation manuelle** | 100% | 100% |
| **Rotation automatique** | 100% | 100% |
| **Détection de la grille et des cases** | 100% | 100% |
| **Découpage de l'image** | 100% | 100% |
| **Résolution et affichage du résultat** | | |
| **Résolution de la grille** | 100% | 100% |
| **Interface graphique** | 100% | 100% |
| **Autres** | | |
| **Réseau de neurones** | 100% | 100% |

FIGURE 31 – Tableau final des avancées

## 9.1 Implémentations demandées

Si nous comparons notre avancement par rapport aux attentes demandées pour la soutenance, nous avons bien réussi à les développer.
Pour le prétraitement, nous avons bien réussi à charger l'image et à supprimer les couleurs. La rotation manuelle est entièrement opérationnelle et fonctionne pour n'importe quel degré de rotation. Pendant le deuxième temps de travail nous avons également réussi à implémenter le prétraitement complet, à savoir la rotation automatique, l'élimination des bruits parasites et le renforcement des contrastes.

La détéction de la grille nous a posée quelques problèmes, nous avions du mal à implémenter la méthode de la transformée de Hough. Pour palier à cela nous avons mis en place une deuxième méthode lors du premier temps de travail. Nous avons finalement réussi à implémenter l'algorithme et avons l'avons combiner avec celui fait précédemment pour permettre une meilleure detéction des lignes. Pour le découpage des cases, tout est fonctionnel et elles sont également récupérer dans une dimension adaptée pour le réseau de neurones.

Nous avons bien mis en place un algorithme permettant de résoudre une grille de sudoku. Il renvoie la grille résolue sous forme de matrice ou sous forme d'image avec les chiffres nouveaux chiffres affichés dans une différente couleur.
L'interface graphique a bien été développer dans le but de répondre à toutes les attentes. L'utilisateur peut effectuer les différentes étapes du prétraitement, résoudre sa grille de sudoku et même résoudre les problèmes au besoin, en prenant la main manuellement.

Pour ce qui est du réseau de neurones, il est bien capable d'apprendre la fonction XOR. Nous avons implémentés toutes les bases du réseau lors du premier temps de travail. Pendant le deuxième temps nous avons réussi à développer une nouvelle méthode qui nous permet de détecter les chiffres de 1 à 9 et d'être très rapide. Cela nous a permis de bien entraîner notre réseau pour avoir un grand taux de réussite et reconstruire les sudokus donnés à partir du réseau.

Nous avons également fait un site web pour présenter notre projet.

# 10 Conclusion

Ce projet fut une réelle découverte dans notre cursus d'Epita. Pour le moment nous avons pu expérimenter nos connaissances et les développer davantage.

Aujourd'hui, nous sommes contents du travail que nous avons réalisé. De plus, nous avons pu nous avancer pour mieux nous préparer à la deuxième soutenance et à la fin de ce projet. Nous avons pu réaliser une rotation manuelle, une suppression des couleurs ainsi qu'une détection de la grille et un découpage de l'image. En plus de tout cela, le programme de la résolution de la grille ainsi que le réseau de neurones sont opérationnels.

# 11 Annexes

# Table des figures