

Soutenance Finale - Projet OCR

Groupe 5

Décembre 2021

Guy Noe DJOUFAING NZUMGUENG Emre ULUSOY

Thomas LEGRAND Youness BAKILY

Table des matières

1	Introduction	4
1.1	Introduction et présentation du projet	4
1.2	Présentation des membres du groupe	4
1.2.1	Guy Noé DJOUFAING NZUMGUENG	4
1.2.2	Thomas LEGRAND	4
1.2.3	Emre ULUSOY	4
1.2.4	Youness BAKILY	4
2	Organisation	5
2.1	Ressources utilisées	5
2.2	Répartition des tâches	5
2.2.1	Gestion de projet	5
2.2.2	Répartition des tâches	6
2.3	Répartition des charges	6
3	Binarisation de l'image	7
3.1	Pourquoi faisons-nous un pré-traitement de l'image ?	7
3.2	Les différentes techniques envisagées	7
3.3	Notre solution	8
3.3.1	Le niveau de gris	8
3.3.2	Le Floutage	9
3.3.3	L'image intégrale	10
3.3.4	La binarisation de l'image	10
4	Détection, découpage de la grille et suppression du bruit	11
4.1	Pourquoi faire une détection de la grille	11
4.2	Les différentes techniques envisagées	11
4.3	Difficultés	13
4.4	Notre solution :	13
4.4.1	Détection de la grille	13
4.4.2	Découpage de la grille	14
4.4.3	Suppression du bruit	14
5	Rotation de l'image	14
5.1	Pourquoi faire une rotation de l'image	14
5.2	Les différentes techniques envisagées	14
5.3	Notre solution	15
5.3.1	Rotation manuelle	15
5.3.2	Rotation automatique	15
5.3.3	Ses limites	16

6 Reconnaissance de caractères et résolution du sudoku	16
6.1 La preuve de concept du Xor	16
6.1.1 La recherche	17
6.1.2 Principes de fonctionnements	17
6.1.3 Les différents principes	18
6.2 Le réseau de Neurones final	19
6.2.1 Principe de fonctionnement	19
6.2.2 Entrainement	20
6.2.3 Exploiter ce réseau de neurones	20
6.2.4 Les Difficultés	20
6.3 Sudoku	22
6.3.1 Les méthodes	22
6.4 Le reader	23
6.5 Le writer	24
6.6 Difficulté	24
7 Sauvegarde et reconstruction de la grille	24
7.1 Pourquoi faire une reconstruction de la grille	24
7.2 Notre solution	25
8 Interface graphique et affichage	25
8.1 Pourquoi faire une interface graphique ?	25
9 BONUS	27
9.1 Palette de couleur	27
9.2 Site web	27
10 Conclusion	27
11 Galerie	27
12 Conclusion	29
13 Bibliographie	30

1 Introduction

1.1 Introduction et présentation du projet

Comme chaque année à Epita, il est coutume pour les Infos spé de s'atteler à un projet de reconnaissance de caractère dit O.C.R (pour Optical Character Recognition en anglais). Cette année il a été employé dans le cadre de la résolution d'un sudoku. Notre application devra proposer une interface graphique permettant de charger une image dans un format standard, de la visualiser, de corriger certains de ses défauts, et enfin d'afficher la grille complètement remplie et résolue. Notre groupe est composé de Guy Noe DJOUFAING NZUMGUENG en tant que chef de groupe ainsi que de Emre ULUSOY, Youness BAKILY et Thomas LEGRAND.

1.2 Présentation des membres du groupe

1.2.1 Guy Noé DJOUFAING NZUMGUENG

Moi c'est Guy Noe, aka UncleDad ou s3ite. J'ai 19 ans et je suis le chef du groupe paguet 18. J'ai toujours été un très grand des OCR. Et ma joie a été grande quand j'ai su que cette année je devais en réaliser un. C'était comme un rêve devenu réalité. Plus sérieusement, je suis un grand fan de jeux vidéo et de d'animés. J'ai décidé cette année d'être chef de groupe car j'avais décidé de me confronter à la gestion d'une équipe sur un projet qui a une importance capitale pour chacun. Et au vu des résultats je suis très satisfait de ce que mon équipe et moi avons réalisé.

1.2.2 Thomas LEGRAND

Je m'appelle Thomas Legrand , je n'ai jamais travaillé en groupe avec Emre , Guy Noé et Youness alors cela a été une première fois pour moi. Je suis très fier ce que nous avons pu réaliser pendant ce projet , notamment au vu de la difficulté de la tache qui nous a été attribué.

1.2.3 Emre ULUSOY

“L'informatique n'est qu'un outil, comme un pinceau ou un crayon.” tel est ma vision sur ce projet. Je suis Emre Ulusoy, 19 ans et jeune adulte. Je suis particulièrement attentif à tout ce qui bouge dans la vie. Et je peux l'écrire : je suis fier d'avoir pu produire avec mes camarades ce projet. Nous avons appris beaucoup de chose tel que l'esprit d'équipe où quand un camarade n'y arrive pas, nous partons vers lui, l'aider.

1.2.4 Youness BAKILY

Je me présente, je m'appelle Younes, je suis étudiant en deuxième année à Epita. J'adore voyager, le cinéma et la culture nippone. Mes hobbies sont le football, le dessin et les jeux vidéo plus spécialement sur le jeu FIFA où j'ai pour habitude de mettre une

raclée à mon camarade et chef de projet Guy-noé. En ce qui concerne la programmation, je ne suis pas spécialement un expert en la matière mais j'ai pour ambition de le devenir grâce notamment à Epita.

2 Organisation

2.1 Ressources utilisées

Pour ce projet, nous utilisons des logiciels open-source et des outils mis à disposition par Epita tel que :

- Overleaf (Un éditeur LaTeX)
- Sublime-Text 3, VSCode et VIM
- Git et GitKraken
- Le compilateur GCC
- Les librairies Gtk et SDL

2.2 Répartition des tâches

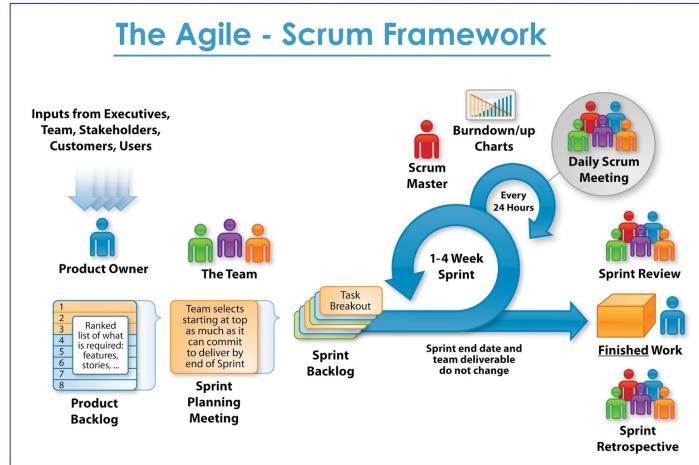
2.2.1 Gestion de projet

Pour ce projet, nous avons décidé de nous inspirer des méthodes de gestion de projet dites agiles, Les méthodes agiles est une méthodologie de gestion de projet qui place le client au cœur du projet et s'adapte tout le long du fil du projet. C'est donc une toute nouvelle façon de voir les choses, et d'aborder le développement d'un projet. Afin d'améliorer leur process et réduire leur taux d'échec. Parmis les méthodes agiles celle pour laquelle nous avons opté est le SCRUM.

Le SCRUM est un terme au rugby et qui signifie mêlée. Les équipes qui utilisent SCRUM se réunissent le plus souvent possible afin de vérifier que le projet avance correctement, toujours prêts à réorienter ce dernier au fil de son avancement. C'est donc une approche dynamique et participative de la conduite du projet, qui garantit pour le client le juste équilibre entre l'investissement prévu et le produit finalement livré.

Le projet se découpe alors de la façon suivante : - La mise en place du backlog : le backlog est un document qui définit les attendus de chacun et liste précisément les tâches à faire pour chaque sprint - Les sprints : un sprint est une phase séquentielle de travail de courte durée où un processus de développement souvent complexe est décomposé afin de le rendre plus simple et plus facile à réadapter et à améliorer en fonction du résultat des évaluations intermédiaires. - Les dailyscrum : Les dailyscrum

sont quant à eux des réunions quotidiennes de 15min d'avancement de projet. - La livraison du projet.



Étant donnée les contraintes liées aux cours, nous n'avons pas suivi parfaitement les différentes étapes de cette méthode. Mais avons essayé de s'y rapprocher au maximum

2.2.2 Répartition des tâches

2.3 Répartition des charges

Youness :

Youness était chargé de s'occuper de la partie de la rotation manuelle de l'image pour la première soutenance. Pour la deuxième soutenance il s'est notamment chargé de la rotation automatique ainsi que des bonus liés à cette même rotation

Thomas :

Thomas devait réaliser pour la première soutenance le solver de sudoku , ainsi que le Xor. Pour la deuxième soutenance il s'est occupé de toute la partie OCR , donc du réseau de neurones final

Guy Noé :

Guy Noé , était en charge de la détection de la grille , du découpage de la grille. correction des imperfections. Et il a notamment effectué la rotation automatique.

Emre :

Emre quant à lui s'est occupé de la binarisation de la grille , de la détection de la grille , de l'interface entre L'humain et la machine , Pré-traitement de l'image , segmentation. Il a notamment aidé sur le réseau de neurones.

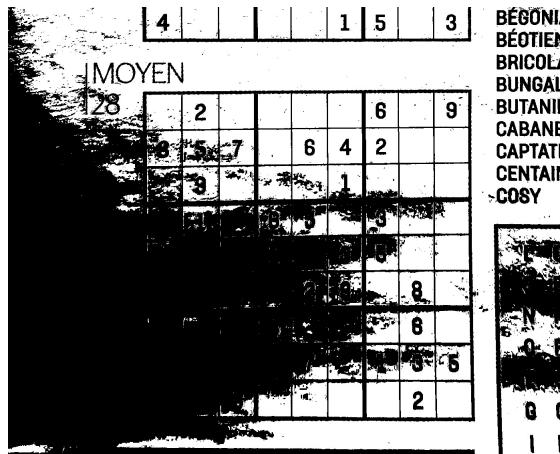
3 Binarisation de l'image

3.1 Pourquoi faisons-nous un pré-traitement de l'image ?

L'objectif de l'OCR est de réussir à lire les chiffres pré-remplis du sudoku. Nous ne pouvons pas lire avec une image tel qu'elle. Nous devons alors appliquer premièrement des pré-traitements : il en existe des milliers.

3.2 Les différentes techniques envisagées

Comme dit, il existe des milliers de méthodes et de techniques pour faire une binarisation d'une image efficace. Pendant nos trois mois de travaux sur ce sujet, nous avons utilisé, pour commencer, la méthode du threshold classique. Nous choisissons arbitrairement une valeur pour laquelle la valeur du pixel ne doit pas dépasser. Le résultat est très peu concluant.



C'est alors, après nos recherches que nous sommes tombé sur la méthode de Otsu qui est une technique semblable au threshold classique sauf que la valeur du pixel est donné en fonction du pixel et non en fonction de l'image. En lisant les recherches d'Otsu sur la binarisation, nous sommes tombé sur Sovola.

Method	Precision	Recall	FM	Time (s)
Sauvola MS _{kx}	0.97	0.94	95.0	170
Lelore	0.99	0.88	92.9	1625
Sauvola MS _k	0.97	0.89	92.1	170
TMMS	0.90	0.95	92.0	250
Wolf	0.99	0.85	91.4	125
Otsu	0.98	0.84	90.3	67
Sauvola	0.99	0.82	89.7	155
Kim	0.99	0.82	89.3	260
Sauvola MsGb	0.99	0.82	89.3	111600
Niblack	0.89	0.91	88.8	95
Su 2011	0.98	0.80	87.3	8800

En effet, la méthode de Sauvola pour la binarisation d'une image dans le but utile d'un OCR est la meilleure pour l'instant. Nous utilisons donc pour ce projet, la méthode de Sauvola pour la binarisation de l'image.

3.3 Notre solution

C'est pourquoi après beaucoup de recherches, nous allons procéder comme suit :

- Transformer l'image en niveau de gris
- Appliquer en chaque pixel, un principe de threshold locale
- Appliquer un filtre de floutage

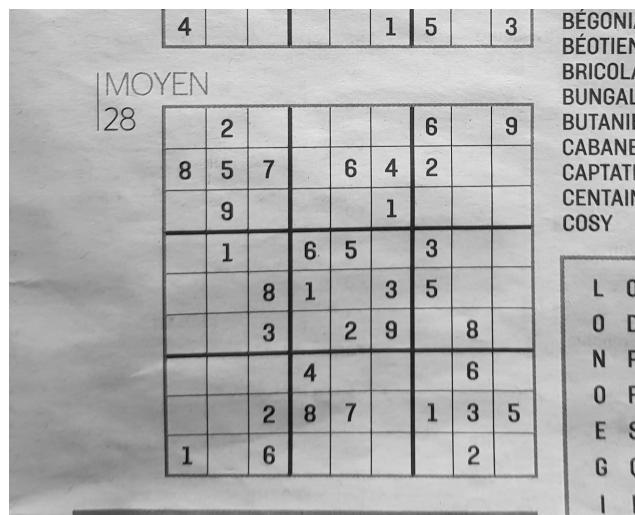
Partons de l'image ci-dessous pour que vous, lecteurs, puissiez suivre les différents traitements pour la binarisation d'une image.



3.3.1 Le niveau de gris

Avant de continuer dans les pré-traitements de l'image, nous devons transformer l'image en niveau de gris. Nous utilisons donc un simple programme qui permet, à chaque pixel, en sortir un pixel en niveau de gris. Cela a été permis avec cette formule, pour les couleurs RGB donné par un pixel alors ce même pixel en niveau de gris sera redéfinis en $(R+G+B) / 3$ pour ses 3 couleurs.

Voici ci-dessous l'image d'un sudoku transformé en niveau de gris.



3.3.2 Le Floutage

Après avoir appliqué la méthode de Sauvola pour la binarisation de l'image, nous avons remarqué que les nombres et la grille sont "pixélisé", c'est alors ici que nous utilisons un floutage pour réduire cet effet de pixel que nous retrouvons. Il y a beaucoup de techniques de floutage : nous avons tester premièrement le floutage de Gauss. Le floutage de Gauss est très efficace mais malheureusement trop lent quand nous l'exécutons. Le floutage de Gauss fonctionne avec un premier Kernel qui est identique à chaque pixel. Ce kernel est une sorte de moyenne entre le pixel et ses pixels aux alentours. Vous trouverez ci-dessous une image représentant le floutage de Gauss avec un niveau de profondeur de 15pixels.

Pour une exécution des programmes avec une rapidité optimale, le floutage de Gauss est assez complexe mais assez simple à mettre en place.

C'est pour cela que nous utilisons, à la place du floutage de Gauss, le floutage médian qui fonctionne avec similitude au floutage de Gauss à une différence : le kernel utilisé dans le floutage médian a tous ses coefficients égalent à 1. Cela permettra d'utiliser l'image intégrale (que nous verrons plus tard) qui facilitera grandement la complexité du programme et son temps d'exécution.



3.3.3 L'image intégrale

La seconde étape, pour une binarisation optimal et peu coûteux, est de construire une image intégrale de l'image. Le principe est assez simple, pour chaque pixel aux coordonnées XY, nous additionnons ses pixels voisins (X-1, Y et X, Y-1) moins le pixel aux coordonnées X-1 Y-1. Comme nous le montre l'image ci-dessous. L'image intégrale est très importante pour des pré-traitements efficaces : cela nous réduit considérablement le temps de calculs et la complexité de l'image.

Pour la binarisation, nous ferons aussi une image intégrale au carré, cela permettra de calculé en chaque pixel la variance de la moyenne des pixels aux alentours.

1	1	1
1	1	1
1	1	1

Image d'entrée

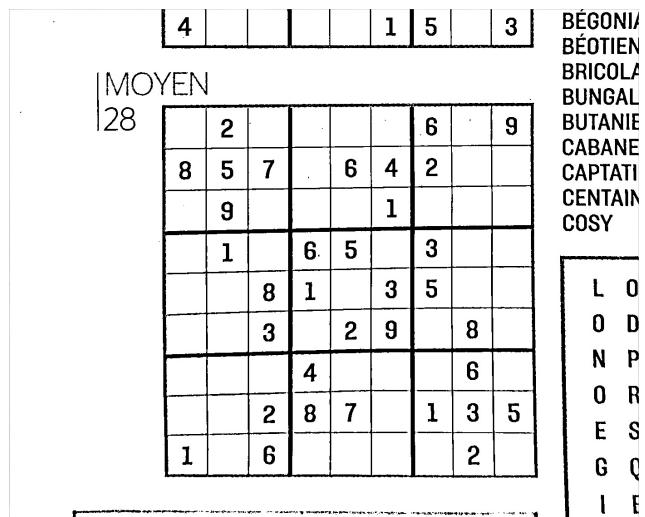
1	2	3
2	4	6
3	6	9

Image intégrale

3.3.4 La binarisation de l'image

Après avoir transformé l'image en niveaux de gris et après avoir déterminé les images intégrales de l'image, nous appliquons une méthode de binarisation pour transformer l'image en noir et blanc pur. Après avoir étudié beaucoup d'étude et de revue scientifique, nous en avons conclus que la technique de la binarisation de Sauvola est l'une des meilleures techniques au sujet d'un OCR. Le principe est très simple, nous utilisons un niveau de seuil unique en chaque pixel. Ce niveau de seuil est calculé à l'aide d'une formule où Sauvola y mélangeant moyenne et écart-type des pixels environnants le pixel X.

C'est pour cela que nous avons calculé précédemment les images intégrales. Si nous devions calculé la moyenne de 60x60 pixels en chaque pixel, le temps de calcul sera incroyablement élevé alors qu'avec les images intégrales le temps de calcul est complètement presque instantané. Vous trouverez ci-dessous l'image de sudoku où a été appliqué la méthode de Sauvola et un filtre de floutage (filtre médian).



4 Détection, découpage de la grille et suppression du bruit

4.1 Pourquoi faire une détection de la grille

Etant donné que les images que nous fournissons à notre algorithme ne sont pas essentiellement constitué de la grille du sudoku, il est donc essentiel que nous définissons quelles parties de l'image appartiennent à la grille et lesquelles n'en font pas parties. La détection de la grille impose toutefois que la binarisation de soit faite.

4.2 Les différentes techniques envisagées

Pour le premier rendu nous avions décidé d'implémenter l'algorithme de la [Transformée de Hough](#) qui est une technique de reconnaissance de formes inventée en 1959 par Paul Hough permettant de détecter des droites dans une image. En caractérisant une droite comme un ensemble de points plus ou moins alignés. Elle repose sur le paramétrage d'une droite par un angle et une distance .

Mais avec une pratique très fragile du langage C nous nous sommes confrontés à de nombreuses difficultés d'implémentations qui nous ont poussée a abandonné cette méthode. La seconde tentative a été de passer par l'utilisation des [Les listes chaînées](#). Mais celle-ci n'étant pas native au langage il a d'abord dans fallu l'implémenter.

Lorsque j'ai commencé à travailler sur cette tâche, j'ai tout d'abord regardé les méthodes généralement employées pour détecter des formes et les premiers résultats m'ont conduit vers la transformée de Hough. Mais cette méthode a été très complexe à comprendre et à mettre en application. Alors j'ai décidé de créer un algorithme qui me permettrait de remplir cette tache de plus ou moins simplement.

La détection de la grille avec les listes chaînées suivait le principe algorithmique suivant :

- On crée une matrice de même taille que l'image.
- Pour chaque pixel de l'image on vérifie si ce pixel est blanc (resp. noir)
- en vérifiant que ses valeurs RGB soit supérieur (resp. inférieur) à une valeur de seuil, ici 125.
- On rempli la matrice avec des 0 et des 1 en fonction que le pixel soit blanc ou noir
- On parcourt les lignes de la matrice en recherchant les lignes de pixel.
- On parcourt les colonnes de la matrice en recherchant les colonnes de pixel.
(Une ligne/colonne de pixel est définie comme une suite consécutive de 1 dans la matrice)
- Si la lignes/colonnes est suffisamment grande (> 10 pixel), on ajoute les coordonnées de chaque point la constituent dans un tableau des coordonnées *.
- On recherche ensuite dans le tableau des coordonnées, les coordonnes croisées, c'est à dire celle qui appartiennent à la fois au lignes et au colonnes
- On retourne ensuite les valeurs max et min de (x, y) des coordonnées croisées
- On extrait le sudoku de l'image initiale grâce à ses coordonnées
le schéma ci dessous est une représentation des lignes et colonnes et des coordonnées croisées. saut que dans notre cas, les lignes et colonnes ne sont pas aussi dispersés.

1	1	0	1	0	1	1	1	1	1	1	1	0	0	1	1	1	1
0	1	1	0	0	0	1	0	1	0	1	1	0	1	0	1	1	1
1	1	1	1	1	0	1	1	0	0	0	1	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1
1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1	0	0	0	1	1	0	1	1	1	1
0	0	1	1	1	0	1	1	1	0	1	1	0	1	0	1	1	1
1	0	1	1	0	1	0	0	1	1	0	1	0	1	1	1	1	0
1	0	1	0	1	0	0	1	1	1	1	1	1	0	1	0	1	1
0	1	0	0	1	1	0	1	1	1	1	1	0	1	1	1	0	1
1	1	1	1	0	1	0	1	1	1	1	1	1	0	0	0	1	0
0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	0	1
1	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	0	1	1	1	1	0	1	1	1	0	0	1	0	0	1
1	0	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	0
1	0	0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1

Représentation des lignes et colonnes et des coordonnées croisées.

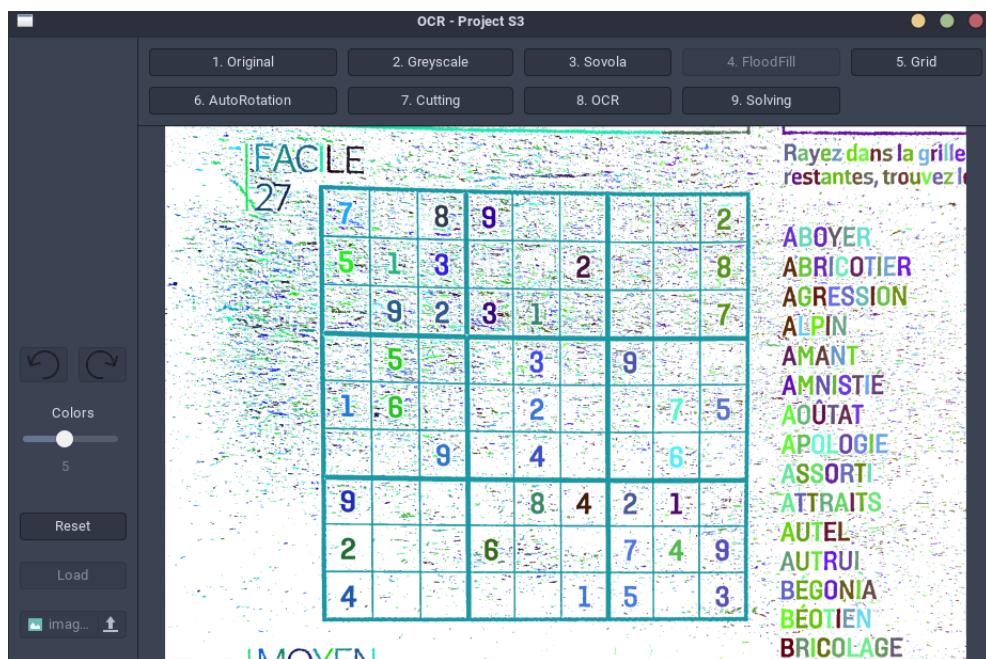
4.3 Difficultés

L'utilisation des listes chaînées qui corrigeait le problème de taille statique des tableaux rajoutait d'autres problèmes. Dans la théorie, cette méthode est censée fonctionner mais dans la pratique beaucoup moins. Tout d'abord le test de ligne sur chaque pixel de la grille s'avère très complexe. De plus la fonction de marche que si l'image en entrée ne contient que des lignes de la grille. Typiquement les images 2 et 5 (voir galerie) ne renvoient pas des données erronées. Nous avons alors décidé de laisser tomber cette méthode aussi.

4.4 Notre solution :

4.4.1 Détection de la grille

La solution que nous avons finalement choisi pour la détection de la grille est celle utilisant l'algorithme de **Flood fill color**. Le principe est le suivant : - On parcours l'image binarisée, et pour chaque pixel noir, on applique le principe de applique récursivement la coloration sur le pixel et sur ses adjacents, et on compte tous les pixels impactées. - On change de couleur et on recommence jusqu'à ce que tous les pixels soient coloriées. La couleur de la grille est celle qui renvoie le plus grand nombre de pixels colorés. Une fois la couleur de la grille trouvée, on peut alors faire différentes opérations sur la grille comme la suppression de tout élément externe à la grille. Ensuite il suffit de parcourir l'image pour récupérer les coordonnées de l'image.



grille 4 après coloration

4.4.2 Découpage de la grille

Une fois la grille détectée on calcul la longueur d'un côté grâce au théorème de Pythagore. Considérons deux points a et b respectivement les points supérieurs gauche et droit de la grille alors la distance ab est donnée par Le sudoku étant de forme carrée, on déduit que chaque est de longueur $ab/9$ on découpe chaque case.

4.4.3 Suppression du bruit

Certaines images telles que l'image 4 sont susceptibles de contenir du bruit. Et ce bruit n'était résolu qu'en partie avec la binarisation de l'image. Mais grâce à la méthode de Flood fill color on peut détecter ces bouts d'images qui forment les imperfections et les repeindre en blanc.

5 Rotation de l'image

5.1 Pourquoi faire une rotation de l'image

Dans le prétraitement nous devons effectuer une rotation d'image. Il s'agit d'une partie, elle aussi importante puisqu'elle requiert d'être capable de corriger une image de travers. Ceci en prévision de l'éventualité où, l'image aurait été mal scannée, ce qui peut entraîner un découpage de la grille très compliquée à réaliser, il est donc nécessaire d'avoir une bonne rotation d'image. Pour cela, on a décidé d'implémenter la librairie SDL qui nous permet de manipuler les images plus simplement.

5.2 Les différentes techniques envisagées

Afin de programmer un algorithme de détection d'angle, nous avons d'abord effectué quelques recherches sur les techniques déjà existantes, ce qui nous a permis de bien s'informer, et de comprendre les difficultés qu'on aurait pu rencontrer à l'avenir. C'est donc une étape importante qu'il ne faut pas négliger pour éviter de perdre trop de temps et de bien orienter nos idées.

En ce sens, la véritable difficulté réside dans la détection de l'angle. Ainsi, lors de nos recherches, nous avons d'abord expérimenté une méthode qui emploie l'algorithme de la transformée de Hough qui consiste en une technique de reconnaissance de forme théorisée par Paul Hough en 1962. Le principe de l'algorithme étant le suivant :

Sur chaque point, on trace toutes les droites imaginables et on prend la droite ayant le plus de points, ce qui nous permet d'avoir la pente de la droite, et donc l'angle. Cependant, cet algorithme est réputé gourmand en temps de calcul, car le calcul de toutes les droites demande un temps de calcul considérable ce qui n'est pas très optimisé et de plus l'angle déterminé est assez limité. Sans compter sur le fait que lors de la

première soutenance nous avions tenté d'appliquer cet algorithme pour la détection de la grille sans grand succès car étant un algorithme assez complexe. Ainsi, nous avons décidé de renoncer à cette méthode.

5.3 Notre solution

5.3.1 Rotation manuelle

Pour la rotation manuelle, on utilise un algorithme qui prend en paramètre une image et angle préalablement choisi par l'utilisateur et renvoie l'image retournée.

Le principe étant le suivant :

On détermine la meilleure position sur la surface d'origine en appliquant une matrice de rotation inverse à l'aide de fonctions qui permettent de déterminer la valeur et d'écrire un pixel aux positions x,y. En ayant au préalable déterminé la valeur du pixel qui correspond le mieux pour la position i,j de la surface de destination à l'aide du centre de l'image.

Sans oublier de calculer la taille de l'image de destination ce qui permettra par la suite d'éviter le dépassement qui a été une des difficultés lors de la réalisation de la rotation.

5.3.2 Rotation automatique

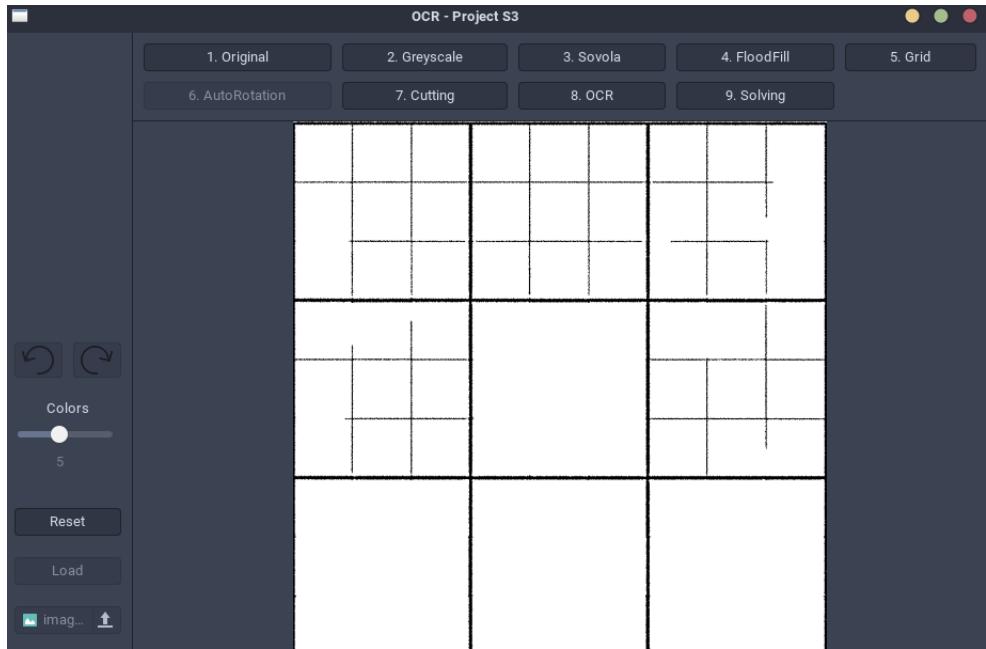
La rotation automatique est une étape importante du prétraitement de l'image. L'image passe en argument n'est pas toujours droite (image 5). Il est donc important de faire une rotation de la grille afin de faire un découpage correct.

Pour réaliser celle-ci il est essentiel de déterminer l'angle de rotation qui était auparavant fourni par l'utilisateur. La méthode utilisée pour le calcul de l'angle est plutôt simple. Sur une image ne contenant que la grille, on recherche les coordonnées sommet celu du haut, de droite et de gauche, que l'on nommera respectivement a, b et c. On définit ensuite le point o qui correspond à la coordonnée ax, by qui forme l'angle droit du triangle aob.

Puis on calcule les longueurs oa et ab avec la formule de pythagore vu précédemment, on cherche ensuite à de cosinus de l'angle oâb avec la formule de trigonométrie adjacent / hypotenuse. Puis on calcule l'angle oâb en faisant arc cos ($\cos(oâb)$). Si la rotation doit s'effectuer sur la droite ou la gauche. Pour cela, on calcule les distances co et ob.

Si ob est supérieur co cela signifie que l'image est plus inclinée vers la gauche et donc qu'il faut tourner vers la gauche sinon l'image est plus tourner vers la droite et

on fait la rotation vers la droite. Mais sur certaines images comportant du bruit sur la grille, les points a, b, c sont mal détectés et on se retrouve à faire une rotation sur une image qui est déjà droite. La solution a été de faire une fonction pour vérifier si une rotation est nécessaire. Celle-ci parcourt la grille à la recherche de ligne et si elle en trouve, cela signifie que l'image est déjà droite et que la rotation n'est pas nécessaire.



Rotation de la grille n°5

5.3.3 Ses limites

Bien que la rotation automatique fonctionne parfaitement pour la plupart des images, celle-ci se montre inefficace si l'angle de rotation dépasse 90 degrés.

Toutefois pour corriger cette limitation, deux boutons ont été ajoutés dans l'interface graphique afin d'effectuer une rotation de plus ou moins 90°.

6 Reconnaissance de caractères et résolution du sudoku

6.1 La preuve de concept du Xor

Pour ce projet nous avons eu dans notre cahier des charges 2 éléments liés aux réseaux de neurones à réalisé. Une preuve de concept d'un Xor ainsi que d'un réseau de neurones final qui lui sera capable de reconnaître des nombres à partir d'une image.

6.1.1 La recherche

Tout d'abord pour le Xor , nous ne savions pas par où commencer pour le réaliser. En effet la seule expérience avec les réseaux de neurones que le groupe a eu avant ce projet était un flappy bird réalisé pendant les tp du S2.

Il fallait donc se documenter au maximum afin de comprendre comment ce système fonctionne. Les sources de documents étant tellement vaste ainsi que la mauvaise méthodologie que nous avons entrepris , c'est-à-dire , regarder sur internet des modèles algorithmiques pour essayer de comprendre comment cela fonctionne était la pire idée que l'on a eu.

En effet , on ne trouvait aucun modèle qui expliquait ce qu'il se passait réellement derrière un réseau de neurones. On a alors stagné pour ensuite avoir trouvé la solution sur un site internet qui expliquait en détail comment un réseau de neurones fonctionne sur le plan mathématiques.

On a donc décidé d'implémenter à la lettre les formules mathématiques qui nous étaient présentées sur le site , et à notre grande surprise cela a fonctionné. En effet l'algorithme du Xor se réalisait sans soucis.

6.1.2 Principes de fonctionnements

- Le Xor utilise l'algorithme de Forward propagation afin de calculer la valeur d'Output ainsi que le taux d'erreur.
- Pour mettre à jour les poids , l'algorithme réalise une backpropagation
- Tout d'abord il fallait calculer la valeur des noeuds caché en faisant l'opération : Pour i allant de 0 à Nombre d'input Somme de (Poids de la liaison Input i -> hidden multiplié par la valeur de l'input en question) additionné au biais entre input et Hidden
- Pour calculer la valeur de l'output du hidden layer i , on récupère la somme précédente et on y applique la fonction de notre choix , ici comme conseiller sur le site mentionné auparavant , nous avons décidé d'utiliser la fonction sigmoid.
- On répète ce processus en remplaçant les inputs par les output de la couche cachée et on obtient un résultat final
- Ce résultat , on va l'utiliser pour calculer le taux d'erreur avec la fonction $\frac{1}{2}(cible - resultat)^2$ le tout au carré on réalise la backpropagation pour corriger les valeurs des poids , les opérations étant d'une longueur excessive. Elles ne seront pas

explicité dans ce rapport

6.1.3 Les différents principes

Lors de l'implémentation de ce xor , une allocation statique avec des listes de données de types double input[10] par exemple a été réalisé , lors de la première soutenance nous pensions que de réaliser une implémentation statique était une mauvaise chose , et qu'une implémentation dynamique permettrait de mieux corriger le nombres de poids. Finalement nous avons décidé de continuer notre approche d'allocation statique même pour la version finale de notre réseau de Neurones.

Lors de la première soutenance , notre groupe a été intrigué par la question demandée , on se demandait pourquoi on devait réaliser un Xor , finalement lors de la création de l'algorithme final nous avons compris , notamment car nous avons repris la structure du Xor avec la fonction Sigmoid.

Ensuite on nous demandait de pouvoir sauvegarder les valeurs des poids de notre réseau de neurones , comme ça on ne serait pas obligé de refaire l'entraînement. Pour cela , plusieurs solutions se présentait à nous , on pouvait réaliser un seul grand fichier qui lui contenait toute les valeurs des poids et nous utiliserons des séparateurs pour les distinguer , finalement , nous avons choisi de tout trier en 4 fichier différents avec une extension .weight pour les poids et une extension .bias pour les biais.

Nous avons donc à la fin de l'entraînement 4 fichiers contenant les valeurs de type double des poids. Pour lire ces fichiers et les implémenter dans le réseau de neurones , on lit simplement chaque ligne et on l'affecte à un des poids. Ces éléments furent importants pour le reste du projet notamment pour le réseau final.

```

Epoch 9000 == ErrorRate = 0.276079
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005504
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474577
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995653
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525393
Epoch 9100 == ErrorRate = 0.276078
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005470
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474578
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995678
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525393
Epoch 9200 == ErrorRate = 0.276077
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005437
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474578
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995703
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525393
Epoch 9300 == ErrorRate = 0.276077
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005405
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474579
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995728
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525392
Epoch 9400 == ErrorRate = 0.276076
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005374
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474580
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995752
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525392
Epoch 9500 == ErrorRate = 0.276075
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005342
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474581
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995776
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525391
Epoch 9600 == ErrorRate = 0.276074
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005312
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474581
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995799
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525391
Epoch 9700 == ErrorRate = 0.276073
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005282
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474582
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995822
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525391
Epoch 9800 == ErrorRate = 0.276072
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005252
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474583
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995845
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525390
Epoch 9900 == ErrorRate = 0.276071
Case 00 | Input 1 : 0.000000 | Input 2 : 0.000000 | Output: 0.005223
Case 01 | Input 1 : 0.000000 | Input 2 : 1.000000 | Output: 0.474583
Case 10 | Input 1 : 1.000000 | Input 2 : 0.000000 | Output: 0.995867
Case 11 | Input 1 : 1.000000 | Input 2 : 1.000000 | Output: 0.525390
Epoch 10000 == ErrorRate = 0.276071

```

La preuve de concept du Xor

6.2 Le réseau de Neurones final

6.2.1 Principe de fonctionnement

Faire un réseau de neurones capables de reconnaître des chiffres était une tâche imposante pour notre groupe , nous avons eu certaines difficultés lors de la première soutenance , mais nous avons quand même réussi à garder la tête haute et nous avons donc réussi à créer un algorithme capable de reconnaître des chiffres.

Pour cela nous avons repris la structure du Xor , avec la Forward propagation etc. L'algorithme fonctionne grâce à la fonction sigmoid principalement , en effet le calcul de l'output lors du passage dans la fonction “Forward” se réalise avec une fonction sigmoid

, et la fonction “Backpropagation” se réalise avec une dérivé de cette même fonction. L’algorithme à été simple à écrire , il fallait simplement reprendre ce qu’on avait réalisé pour le Xor et l’adapter pour une fonction prenant en argument des images en 28x28 , et il fallait donc calculer le bon nombre de neurones cachés a utiliser sous peines de ne pas avoir un réseau de neurones assez puissant.

6.2.2 Entrainement

Pour l’entraînement , nous avons utilisé un principe ingénieux , on parcours l’image renvoyé par la fonction de découpage de la grille et on la transforme en liste de bits représentant 1 pour la couleur noir et 0 pour la couleur blanche. Cette liste de bits sera complété par une liste représentant la valeur de ce nombre en mettant la valeur 1 à l’index , nous avons ensuite mit ces 2 éléments ensembles dans une structure , nous permettant de faire des opérations sur une liste de structures plutôt que de devoir initialisé à chaque fois les 2 listes pendant le calcul et l’entraînement.

6.2.3 Exploiter ce réseau de neurones

Pour l’utilisation du réseau de neurones afin de pouvoir détecter quel chiffre est présent sur l’image nous actualisons les poids précédemment sauvegardées dans le réseau de neurones , ensuite nous copions les bits dans notre structure directement dans les input de notre réseau. Nous réalisons ensuite la fonction forward propagation sur ce même réseau de neurones.

Cela aura pour conséquence de changer les valeurs des inputs , nous avons donc besoin ensuite de récupérer la valeur du plus lourd Output afin de pouvoir détecter le nombre sur l’image. Cette opération sera réalisée sur les 81 images renvoyées par le découpage de la grille. Pendant que les calculs se faisaient , la matrice contenant toutes les valeurs du sudoku a été créée et remplie au fur et à mesure du parcours des fichiers contenant les images des nombres. Lorsque cela est terminé , le programme utilise la fonction writer du sudoku précédemment codé afin de créer un fichier grid made dans notre dossier contenant nos données.

6.2.4 Les Difficultés

Expliqué comme cela , les opérations paraissent simples à réaliser , mais en vérité notre groupe a été accueilli par une multitude de problèmes lors de la création de cet IA.

Tout d’abord , lors de l’entraînement , il fallait choisir le bon nombre de neurones , sous peine de se retrouver avec une boucle infinie car le réseau n’arrive pas à atteindre ses objectifs. Ensuite , nous avions eu une vision erroné de la valeur alpha et eta qui définissent le pas d’apprentissage , en effet , on pensait que plus la valeur était haute ,

meilleur était l'entraînement , ce qui nous donnait des réseaux de neurones capables de reconnaître des chiffres déjà appris , mais cela se limitait à 1 seule chiffre. On a donc changé ces valeurs a 0.01 et 0.1 respectivement , et celà à tout de suite changé la donne , un entraînement durait à la base 40 secondes ,puis après ce changement prenait 20 minutes

. Un autre souci s'est glissé dans la montagne de problèmes techniques que nous avons rencontrés pendant ce projet, ce souci était la trop grande diversité des systèmes d'exploitations.

En effet 2 personnes de notre groupe utilisait WSL sous windows , 2 autres personnes utilisait 2 distribution différentes de linux et enfin le projet était corrigé sous NixOs , nous avions eu donc beaucoup de soucis de comptabilité notamment entre windows et Linux , certaines fonctionnalités ne marchait pas sous linux alors qu'elles fonctionnait sous windows.

Par exemple , lors de la lecture des poids afin de générer l'ia pour la détection des nombres , une fonction n'était pas capable de convertir une string en format (0.0000) en double car il y avait un point , alors que la personne sous windows , pouvait le réaliser sans soucis avec la même fonction.

Ensuite , L'algorithme n'est pas un algorithme parfait , et malgré un taux d'erreur de 0.01 , l'algorithme arrive encore a mal comprendre des nombres , notamment à cause du bruit présent et celà malgré les différents traitements de l'image. Nous avons donc dû optimiser l'entraînement du réseau de neurones pour ne pas avoir de poids de neurones qui définissent le comportement de notre algorithme comme un if else , mais plutôt qu'il marche comme un vrai ocr , pour cela il fallait l'entraîner mais pas trop , et a 30 minutes par entraînement , la tâche a été compliquée

```

Epoch => 200
output[0] => 0.005755
output[1] => 0.000540
output[2] => 0.002787
output[3] => 0.076705
output[4] => 0.001427
output[5] => 0.050936
output[6] => 0.805798
output[7] => 0.007111
output[8] => 0.156498
output[9] => 0.000711
Error Rate => 6.825579
Epoch => 200
output[0] => 0.061198
output[1] => 0.070639
output[2] => 0.004630
output[3] => 0.010248
output[4] => 0.005006
output[5] => 0.000159
output[6] => 0.001962
output[7] => 0.960743
output[8] => 0.000374
output[9] => 0.006173
Error Rate => 6.830814
Epoch => 200
output[0] => 0.008960
output[1] => 0.004502
output[2] => 0.000901
output[3] => 0.110265
output[4] => 0.001678
output[5] => 0.003596
output[6] => 0.109963
output[7] => 0.016790
output[8] => 0.807741
output[9] => 0.000722
Error Rate => 6.861620
Epoch => 200
output[0] => 0.008579
output[1] => 0.007255
output[2] => 0.000291
output[3] => 0.060464
output[4] => 0.004295
output[5] => 0.052076
output[6] => 0.041635
output[7] => 0.072959
output[8] => 0.013809
output[9] => 0.000180
Error Rate => 7.368320

```

Le training du neural Network

6.3 Sudoku

Comme son nom l'indique , le solver de sudoku est un programme qui prend en argument une grille de sudoku au format .txt et qui la résout. Puis après résolution crée un fichier contenant le nom initial de la grille avec une extension .result.

6.3.1 Les méthodes

Pour le réalisation algorithmique du sudoku , nous avons eu plusieurs solutions , soit un réseaux de neurones ,étant dans le thème cela semblait la solution la plus logique , mais nous nous sommes finalement tourné vers la méthode de résolution par récursion

utilisé lors d'un tp de C pendant le S2. Nous avons pour cela ré-adapté le code destiné à du cSharp pour qu'il fonctionne en C.

Principe :

Le principe de l'algorithme est très simple , le programme prend un nombre entre 1 et 9 et le pose à la case libre où se trouve l'index de l'algorithme et réalise une série de test sur la ligne , la colonne , ainsi que sur la matrice 9x9 lié à cet index.

Si tout ces tests renvoient vrai , alors dans ce cas le nombre est validé et le programme continue récursivement sur une autre case. Cet algorithme permet de gérer les cas où l'algo a mal placé son premier nombre , car trop peu d'information au préalable,notamment grâce à la remontée de la récursive.

Après avoir implémenté l'algorithme en traduisant les composantes de CSharp en C.

programme ne fonctionnait absolument pas , il y avait bien des parcours de matrice qui se réalisait mais cela renvoyait les mauvaises valeurs , l'erreur fut trouvé plus tard, le système de matrice de type grid[SIZE][SIZE] ne prend pas ses arguments de manière grid[x][y] mais plutôt de manière grid [y][x]. Après avoir inversé les variables,le programme fonctionnait correctement.

6.4 Le reader

Lors de la première soutenance à la création du reader , nous pensions que cette tâche allait se révéler être facile , notamment par le fait que nous avons déjà abordé le principe des filestream l'année dernière en S2.

Principe de fonctionnement :

Le programme prend en argument le chemin d'accès du fichier ainsi que la matrice qui doit être complétée.

Il l'ouvre et lit chaque caractère 1 par 1 tout en incrémentant 2 variables permettant de se déplacer dans la matrice .

Si le caractère n'est pas un chiffre ou un point , alors il est ignoré , si le caractère est un point, alors dans la matrice apparaîtra un zéro à son index.

Nous avons choisi cet implémentation plutôt que de faire une matrice de caractères afin à ne pas avoir à gérer des conversions de type char vers des types int lors de la résolution.

Penser que cet algorithme était facile fut une énorme erreur,en c , les arguments d'une fonction sont renvoyés avec le pointeur argv.

Il n'y a donc pas de string , ce qui a rendu la tâche de lire l'input donné par l'utilisateur compliquée lorsque l'on ne sait pas que argv[0] renvoie la fonction écrite dans le terminal

et qu'il faut choisir argv[1] afin de recevoir le premier argument. Ensuite, lors de la lecture des caractères , On souhaitait ne pas prendre en compte le caractère "/n" ainsi que les espaces , or il semblerait que pour son cas simplement les exclusions ne fonctionnait pas et qu'il fallait vérifier chaque lettre avec le code ASCII de ce que l'on souhaite ne pas prendre en compte dans l'algorithme.

De plus, le problème d'agencement des matrices n'était pas connu par notre groupe lors de l'implémentation du code , donc il recevait des grilles fausses , à l'envers. Nous avons réussi à réparer cette erreur après la découverte de l'inversement

6.5 Le writer

Le programme permettant d'écrire dans un fichier le résultat de la matrice dans un fichier était de loin le plus facile , notamment car nous l'avons implémenté à la toute fin, lorsque tous les autres programmes liés au solver étaient fonctionnels.

Et donc qu'on avait compris comment marchait les filestream et les matrices en c. Son principe est très simple , il lit la matrice grid et réalise une simple fonction print dans le fichier , en ajoutant des caractères “/n” et des espaces avec des conditions.

6.6 Difficulté

Lors de la première implémentation de cet algorithme , nous sommes partis sur des sous fonctions qui renvoie directement la grid , avec comme prototype de fonction int*reader par exemple .

Cette implémentation prenait énormément de mémoire car chaque matrice était recréée, donnant donc à la fin de l'exécution du programme 5 matrice de taille 9 fois 9 dans la mémoire.

Nous avons donc été conseillé par certaines personnes extrêmement compétentes dans la classe , d'utiliser des types void et de faire 1 seule matrice. Ce qui permet aujourd'hui d'avoir un programme qui crée 1 seule matrice et qui la modifie.

7 Sauvegarde et reconstruction de la grille

7.1 Pourquoi faire une reconstruction de la grille

Une fois la reconnaissances des caractères et la sudoku solver appliquées, il faut maintenant replacer les différents chiffres dans la grille.

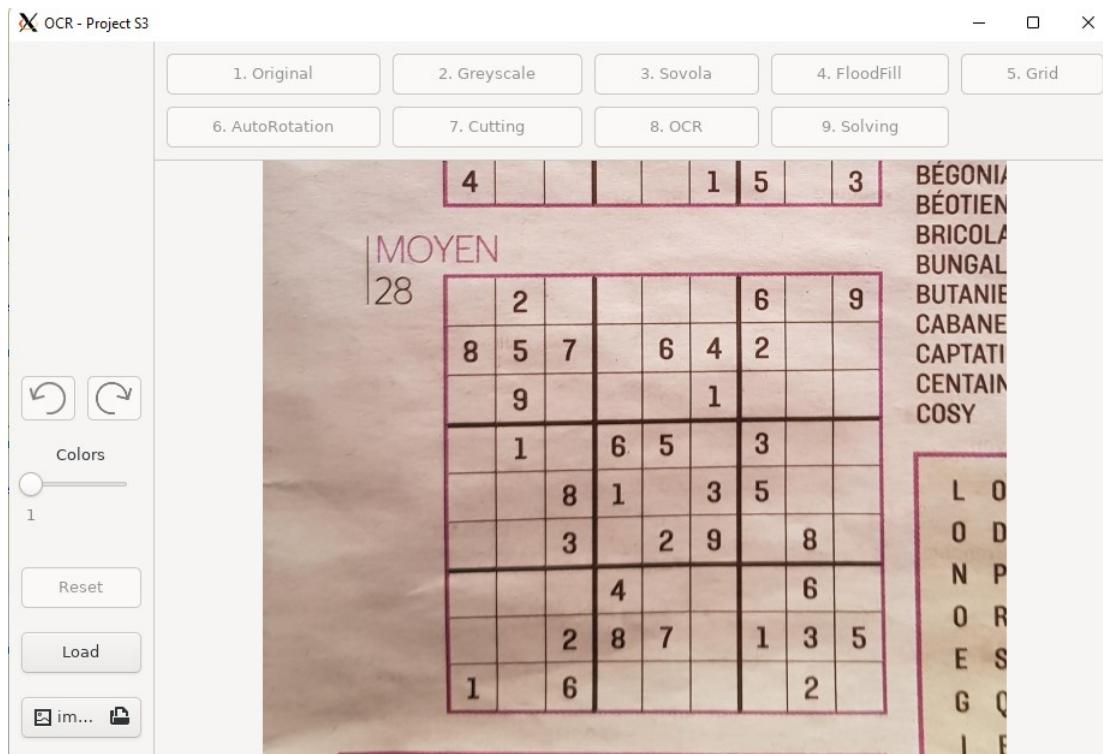
7.2 Notre solution

La première solution consistait à créer des templates de chaque chiffre, à lire le fichier de sauvegarde généré par le solver et à coller l'image du chiffre après la dimensionnement dans la grille de sudoku.

Bien que celle-ci soit fonctionnelle, nous avons décidé de changer ce système, pour avoir un code optimisé au maximum. Notre solution actuelle repose donc sur l'utilisation de la librairie *SDL_TTF qui permet d'crire sur une image*.

En ce qui concerne le remplissage de la grille qui est la dernière tâche effectuée dans l'interface graphique et qui permet d'avoir un rendu visuel plus qualitatif qu'un simple fichier texte. Pour cela on utilise le fait que les cases ne contiennent pas de chiffres sont blanches afin de dessiner les chiffres prélever dans le fichier .txt du solver pour y remplir la grille.

Après que l'utilisateur ait appuyé sur le bouton "Load", tout le processus de génération des images est enclenché. Toutes les images sont alors enregistré dans le dossier *datas/tmp* qui permettra de switch entre chaque image sans difficulté.

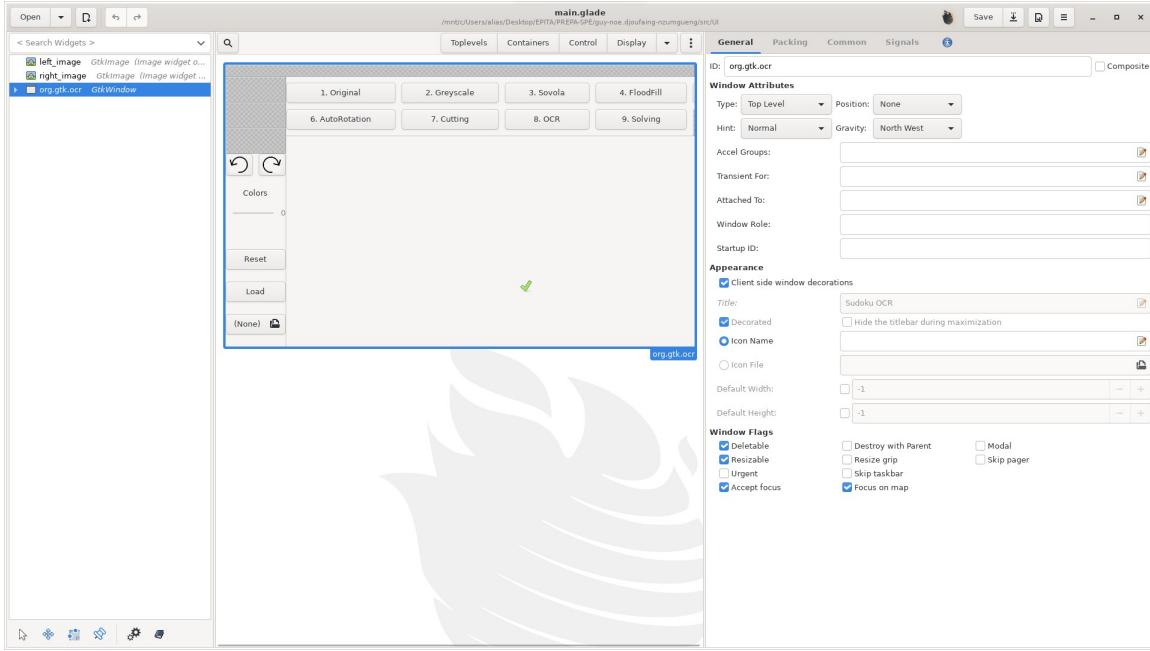


8 Interface graphique et affichage

8.1 Pourquoi faire une interface graphique ?

Pour tout programme et tout projet, il faut que l'homme interagisse avec ce dernier. Cela passe d'un simple levier physique ou d'une commande dans un terminal à

une grande machine graphique. Nous concernant, nous avons opté pour un interface graphique utilisant l'outil Glade. Le principe est très simple : nous plaçons les boutons et les menus là où nous en avons besoin.



Nous utilisons alors l'outil Glade et nous importons donc la librairie GTK que Glade utilise. Après nos péripéties sur Gtk, nous en concluons que la librairie Gtk est une librairie particulièrement très puissant et que même faire un jeu vidéo est complètement possible.

Nous avons alors pensé à un système assez simple mais très dynamique :

- Un bouton où nous choisissons le fichier
- Un bouton load qui permet de générer toutes les images qui seront ensuite activé et désactivé par les boutons

La fenêtre est assez simpliste. La partie de droite sert à charger l'image originale et contient aussi les éléments nécessaires pour Reset ou encore appliquer le bonus de couleur et faire des rotations manuels. Quant à la partie du haut, les boutons servent à se balader entre les différents traitements que nous appliquons sur l'image originel.

Relier chacun des parties pour un seul programme est une tâche assez compliquée. Nous devons comprendre comment le code de l'un ou de l'autre fonctionne et rectifiait son code si cela nous empêche de continuer.

9 BONUS

9.1 Palette de couleur

Comme premier bonus, nous avons décidé d'ajouter à l'interface un slider permettant de modifier la gamme de couleur du flood fill.

9.2 Site web

Le second bonus est un site qui sert de vitrine à notre projet. Il est disponible à l'adresse <https://www.justwood-game.com/>

10 Conclusion

11 Galerie

image 1

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	

image 2

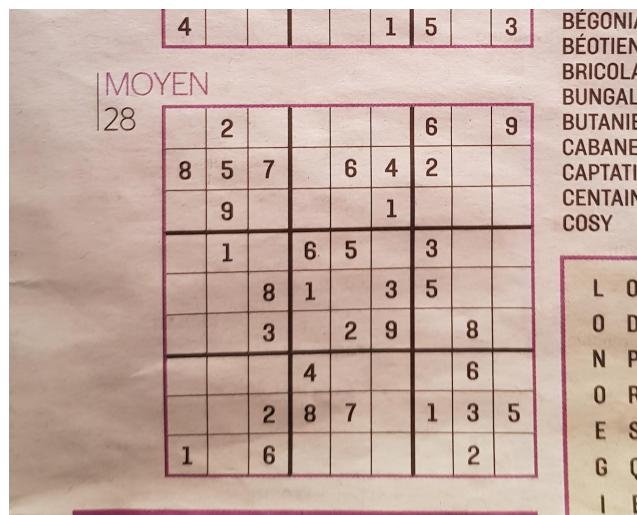


image 3

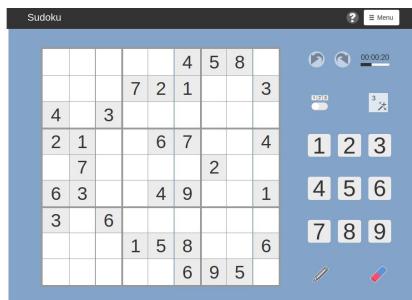


image 4

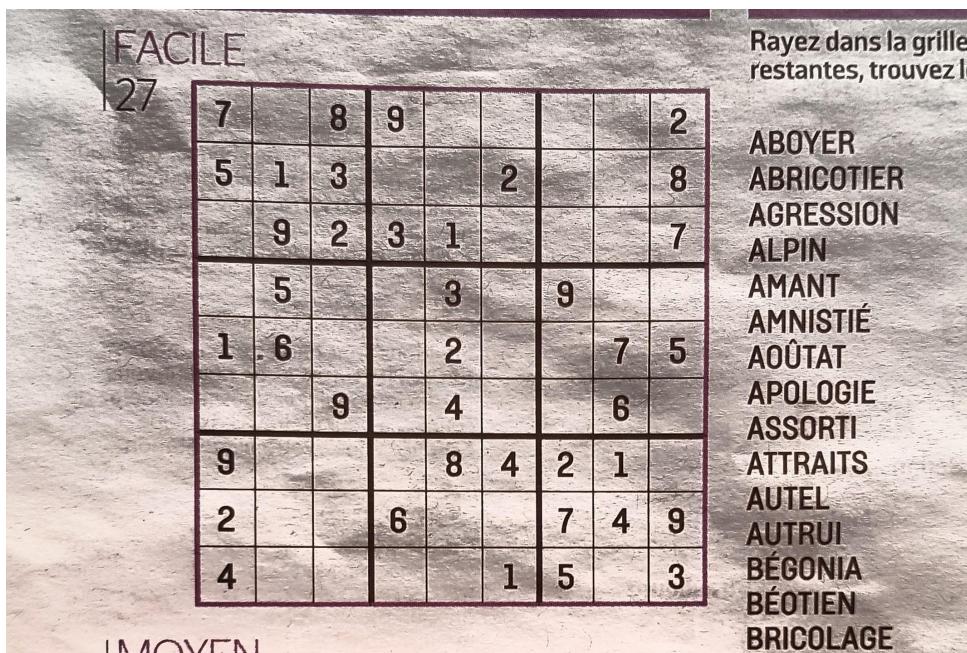
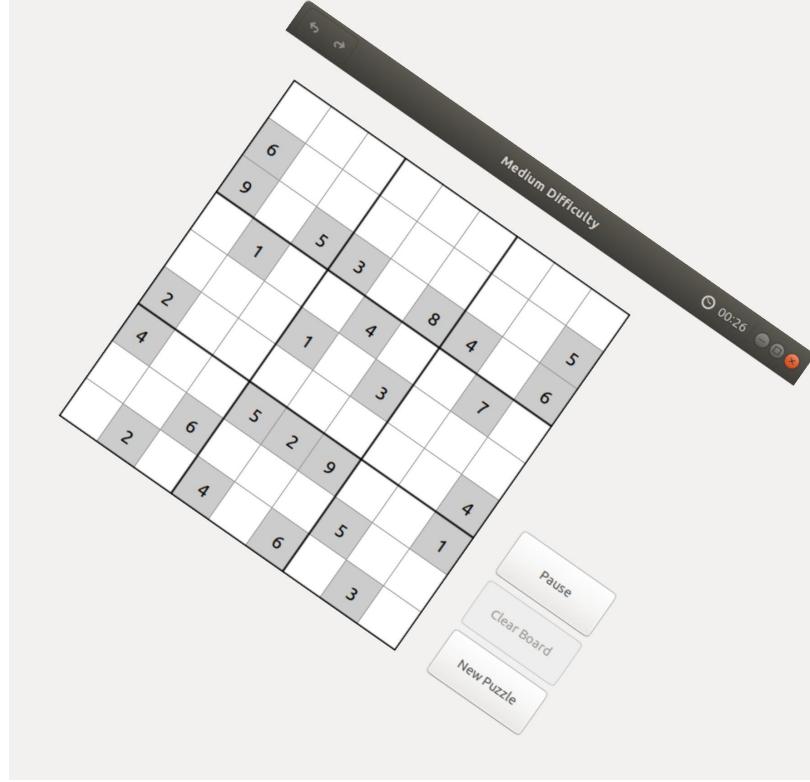


image 5



12 Conclusion

En conclusion, le projet OCR mené durant ce semestre, auquel nous avons dû faire face, touche à sa fin. En effet, ce projet nous a été très instructif dans son concept, que ce soit en terme de programmation notamment, car nous avons dû implémenter un réseau de neurones capable de reconnaître des caractères, le traitement d'image qui est une partie inédite pour nous et qui nous a permis d'acquérir une expérience dans ce domaine de l'informatique.

Mais également dans la réalisation d'un projet commun, en passant par plusieurs étapes, que ce soit la répartition des tâches, la mise en commun de nos parties, la gestion du temps et le respect d'un cahier des charges. Des compétences nécessaires et attendues dans le parcours d'un ingénieur. Bien que lors de notre première soutenance tout n'allait pas comme nous le souhaitons nous avons pu, à partir, de nos erreurs redresser la barre afin de réaliser une version du projet dans nos attentes, c'est-à-dire une version qui respecte au plus le cahier des charges.

Ainsi, cette conclusion sonne la fin de notre périple, nos émois et conclut tous nos efforts.

13 Bibliographie

- 1. M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13 :146–165, 2004.
- 2. N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1) :62–66, January 1979.
- 3. J. Bernsen. Dynamic thresholding of grey-level images. In *Proceedings of the International Conference on Pattern Recognition*, pages 1251–1255, 1986.
- 4. E. Gabarra and A. Tabbone. Combining global and local threshold to binarize document of images. In *Pattern Recognition and Image Analysis*, volume 3523 of LNCS, pages 173–186. Springer, 2005.
- 5. Y. Rangoni, F. Shafait, and T. M. Breuel. OCR based thresholding. In *Proceedings of IAPR Conference on Machine Vision Applications*, pages 98–101, 2009.
- 6. **HOUGH**.
- 7. **FLOOD FILL COLOR**
- 8. **GTK**