SOUTENANCE 1

2021-2022

Projet OCR

Les MédiOCRes PrOCRastinateurs

Alexandre Devaux-Riviere
Jules Girod
Esteban Arroyo
Romain Ludet

Table des matières

1 Introduction - Présentation du sujet

L'objectif de ce projet est de réaliser un programme capable de résoudre un Sudoku à partir d'une image de la grille et uniquement à partir de cela.
L'image peut être de bonne qualité tout comme elle peut ne pas l'être, elle peut également avoir été tournée ou non.
Pour contourner ces problèmes, nous devons réaliser un traitement d'image afin d'obtenir une image la plus épurée possible et nette pour pouvoir reconnaître les chiffres à l'aide d'une intelligence artificielle. Cette étape de reconnaissance est appelée Optical Character Recognition d'où le nom du projet OCR.
Une fois les chiffre reconnus, nous devons résoudre le Sudoku et afficher le résultat à l'utilisateur via une interface.

2 Présentation

2.1 Présentation du groupe

Nous sommes "Les Médiocres Procrastinateurs" mais il ne faut pas s'y méprendre, nous travaillons beaucoup car nous sommes médiocres en procrastination.

2.2 Présentation des membres

Alexandre Devaux-Riviere :

Passionné d'informatique depuis mon stage d'observation de 3e, être considéré comme le technicien familial ne me suffisait pas et j'ai donc rejoint EPITA où j'ai acquéri la plupart de mes connaissances en programmation.
Aujourd'hui, en tant que chef de projet, mon rôle est de rester à l'écoute, faire tout mon possible pour épauler mes camarades ainsi que maintenir une organisation efficace afin que le projet aboutisse et soit une réussite. Ce projet est une bonne opportunité pour moi de contribuer au développement d'un OCR qui me plaît et dont je peux être fier.
Je vais et j'ai déjà acquéri de nombreuses connaissances et normalement assez d'expérience pour pouvoir passer du rang de « gueux » au rang « soup lord ». Hâte de passer ce palier symbolique.

Jules GIROD :

Bonjour, je suis Jules GIROD, j'ai 19 ans, je suis passionné d'informatique depuis longtemps. Avant EPITA, je m'étais déjà intéressé à la programmation C et un peu à la SDL, ce qui m'a donné des facilités pour commencer celle-ci et expliquer des méthodes/fonctions à mes camarades.
J'ai également fait un petit peu de traitement d'image en Python, ce qui m'a permis de connaître un peu les méthodes afin de réduire le bruit d'une image ou encore la binarisation, etc.
Ce projet me permet donc de travailler en groupe avec mes camarades sur un sujet qui nous intéresse tous.

Esteban Arroyo :

Bonjour, je nomme Esteban Arroyo, j'ai 20 ans, je suis passionné d'informatique, c'est pour cela que j'ai préféré faire sti2d.
De plus, j'aime connaître les dernières news sur les nouvelles technologies qui vont sortir. J'adore prendre des photos et les partager au plus grand nombre.
Ce projet est très enrichissant, car il nous permet de connaître les subtilités du C, langage qui sera très utile pour l'année prochaine en ING1.

Romain Ludet :

Bonjour, je m'appelle Romain Ludet, j'ai 19 ans. Je me suis très tôt intéressé à l'informatique et j'ai su très vite que c'était ce que je voulais faire. J'ai ainsi eu la chance de ne pas avoir à me casser la tête pour trouver une orientation professionnel pendant le lycée.
J'ai commencé par faire des petits jeux sur Scratch lorsque j'étais au collège, puis je suis rapidement passé sur un langage de programmation car je trouvais trop de limites et de contraintes à Scratch. Je me suis donc jeté sur les bases du C avant finalement de changer et faire du Python en classe de Terminale en spécialité ISN.

3 Répartition des tâches

	Alexandre	Jules	Esteban	Romain
Application (interface et système)				
Chargement de l'image en mémoire		X	X	
Prétraitement manuel		X		X
Sauvegarde de la grille		X	X	
Affichage de la grille résolue	X	X	X	
Interface graphique	X	X	X	
Traitement de l'image				
Détection de la grille			X	
Prétraitement automatique				X
Suppression des couleurs		X	X	
Détection des cases de la grille	X	X	X	
Récupération des chiffres présents dans les cases			X	
Reconnaissance de caractères	X		X	
Reconstruction de la grille			X	
Réseau de neurones				
Réseau de neurones OCR (et test XOR)	X			
Apprentissage par propagation des corrections	X			
Sauvegarde et chargement des poids du réseau	X			
Jeu d'images pour l'apprentissage	X			X
Résolution du sudoku				
Ouverture et sauvegarde du fichier contenant la grille		X		
Résolution de la grille		X		

Légende : X = Contribution

4 Prétraitement

Le prétraitement d'une image est probablement la partie la plus importante afin d'avoir une reconnaissance de la grille la plus rapide et réussie possible. C'est pourquoi il faut procéder à plusieurs étapes successives.

4.1 Chargement de l'image en mémoire

La première étape, la plus simple, est de charger l'image dans la mémoire.
Ainsi, le programme prend le chemin relatif ou absolu d'une image en paramètre. Cette image va être

chargée en mémoire tout au long de l'exécution du programme.
Pour réaliser cela, nous avons utilisé la librairie : SDL1.2, ainsi que ses dérivées. La bibliothèque nous permet de charger les images de type ".bmp". A terme nous offrirons la possibilité de charger des images. Nous avons la possibilité via la SDL d'afficher l'image sur l'écran de l'utilisation. Pour ce premier rendu, nous avons fait le choix d'afficher la détection de la grille même s'il ne s'agit que d'une étape intermédiaire afin d'avoir un résultat concret de notre avancement.
A terme, nous afficherons l'image initiale résolue.

4.2 Conversion en niveaux de gris

La première étape pour retirer le bruit (les pixels parasites empêchent la reconnaisse de la grille) est de convertir notre image en nuances de gris.
Cette technique de conversion permet de retirer une majeure partie des détails inutiles à la reconnaissance de l'image.
Pour faire ce filtre, il suffit d'appliquer une formule mathématique 0.21 * r + 0.71f * g + 0.07 * b où r,g,b représentent la quantité de rouge, de vert et de bleu sur chaque pixel. Une fois la valeur obtenue on définit alors que la nouvelle couleur du pixel contiendra autant de rouge, de vert et de bleu, à savoir la valeur trouvée. On fait cela sur chaque pixel de l'image pour pouvoir obtenir une image en nuances de gris.

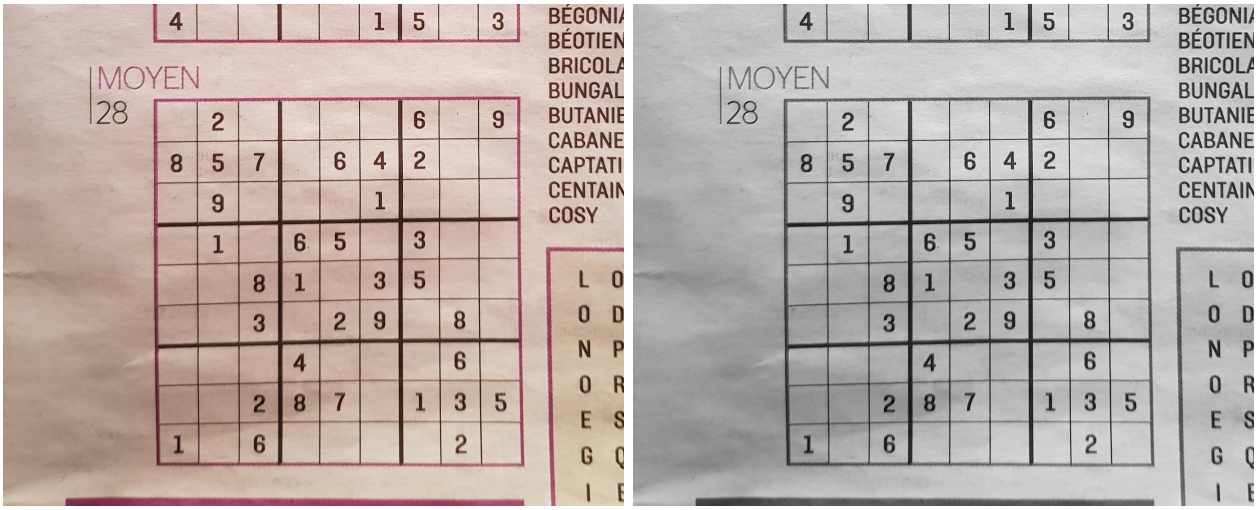

FIGURE 1 – Image convertie en nuances de gris

4.3 Conversion en noir et blanc / Binarisation

Pour convertir en noir et blanc la méthode la plus simple est d'étudier la valeur des pixels sur une image en nuance de gris, si la quantité de rouge (ou bleu ou vert car ils possèdent la même valeur) est supérieur ou égal 127 alors on peut convertir ce pixel en un pixel blanc à savoir r,g,b = 255,255,255 et si la valeur est inférieure alors r,g,b = 0,0,0, ce qui donne du noir.
Le problème majeur de cette méthode est que, pour les images avec peu de contraste, l'on perd beaucoup d'information et on se retrouve avec une image blanche. Il faut donc trouver un moyen de définir la valeur de seuil en fonction des contrastes globales de l'image.
C'est maintenant qu'entre en jeu la méthode d'Otsu.

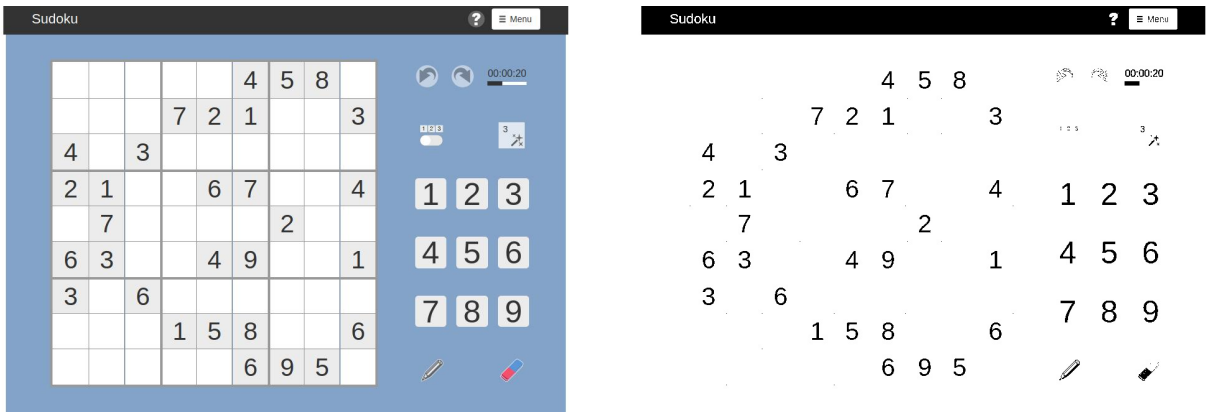

FIGURE 2 – Binarisation avec valeur de seuil = 127

4.4 Méthode de Otsu

La méthode d'Otsu est utilisée pour effectuer un seuillage automatique à partir de l'histogramme des pixels d'une image. L'algorithme suppose alors que l'image à binariser ne contient que deux types de pixels, noir ou blanc, ceux en premier plan et ceux en arrière-plan, puis calcule le seuil optimal afin d'obtenir une bonne séparation du premier plan et de l'arrière-plan.
Commme on peu le voir sur cette image, la méthode augmente grandement la performance de la binarisation.

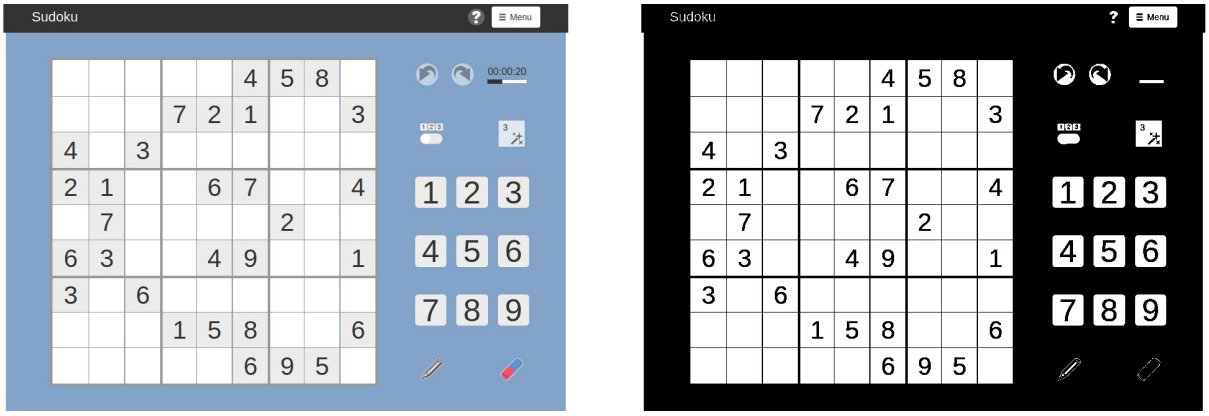

FIGURE 3 – Binarisation avec la methode d'Otsu / valeur de seuil = 195

4.5 Rotation de l'image

Pour le confort de l'utilisateur, notre solver doit pouvoir reconnaître la plus grande palette d'images possible et pas uniquement les images où la grille est parfaitement droite. C'est pourquoi il est important de prendre en compte les images où la grille est désaxée.
Plutôt que de faire une méthode de reconnaissance de la grille qui la trouve même si elle est en diagonale, il est préférable de faire une rotation avant tout traitement.
Pour l'instant nous demandons à l'utilisateur de saisir manuellement l'angle de rotation (0 modulo 360 s'il ne veut pas faire de rotation), mais à terme, nous ferons la rotation automatiquement en détectant les lignes droites de l'image.

Pour faire la rotation, nous avons utilisé la fonction rotozoomsurface qui appartient à une librairie qui est une extension de la librairie SDL à savoir SDL_gfx.
Le problème qu'on a pu repérer est que l'image après rotation perd de sa qualité, c'est pourquoi, l'idéal est de faire sa propre fonction de rotation d'image.

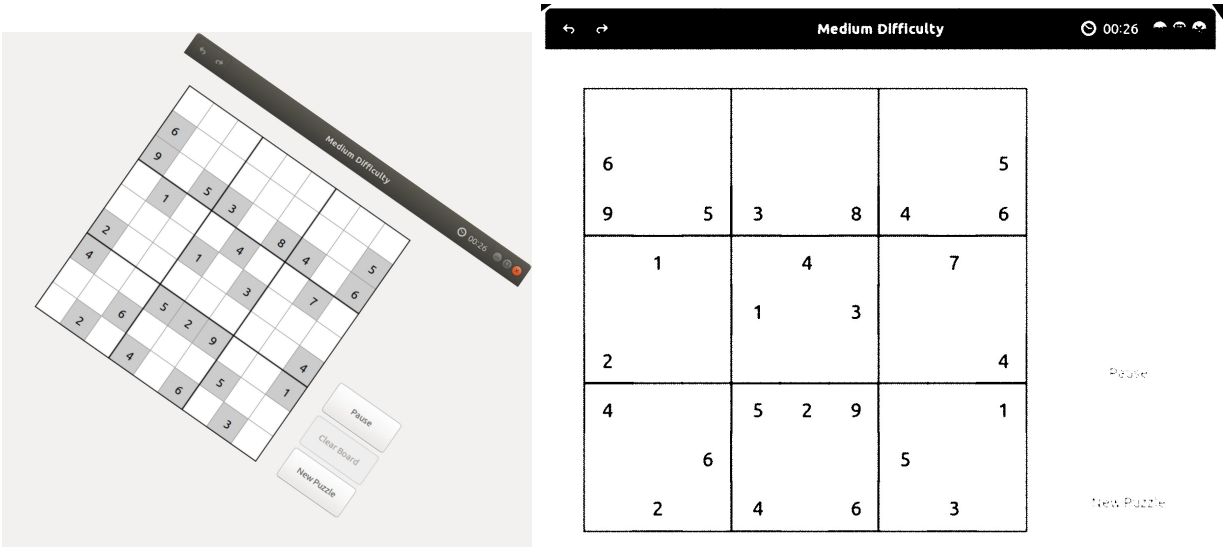

FIGURE 4 – Image après rotation et binarisation

4.6 Autres méthodes à faire

Afin d'obtenir des images avec le moins de bruit possible, nous allons également appliquer des filtres de dilatation, de flou et de renforcement des contrastes.
L'effet de flou nous permettrait d'atténuer les pixels parasites car nous avons donc un niveau de détail moindre.
La dilatation permettrait de pouvoir reconstruire les lignes effacées comme sur l'image06 que nous avons.
Enfin le reforcement des contrastes permettrait de mieux détecter les changements de couleur et donc d'avoir une fonction de binarisation plus précise.

5 Détection de la grille

La détection de la grille est la deuxième étape la plus importantes après le pré-traitement. C'est pourquoi il faut procéder à plusieurs étapes successives :

5.1 Détection des pixels noir dans l'image

Pour pouvoir détecter les lignes, nous devons préalablement effectuer un parcours de notre image en X et Y. Si le pixel parcouru est noir, nous incrémentons le tableau des X et Y correspondant à la coordonnée de ce pixel. Puis nous renouvelons ce programme jusqu'à ce que l'image totale soit parcourue et que tous les pixels noirs aient été récupérés. Ces étapes réalisent la construction de deux histogrammes représentant la densité des pixels noirs par ligne/colonne. À présent, que ces données ont été récupérées, nous pouvons passer à l'utilisation des valeurs dans ces deux tableaux.

5.2 Filtrage des valeurs et détection des lignes

Dans cette partie, nous avons dû concevoir une sous-fonction qui va nous permettre de l'utiliser sur les histogrammes du nombre de pixels noir en X et Y.
Premièrement, nous calculons la moyenne du nombre de pixels noir et son écart-type. Nous les additionnons pour obtenir une valeur de seuil.
Deuxièmement, nous effectuons un parcours du tableau de pixels en X ou Y selon les paramètres de la fonction. Lors de ce parcours, dès que la valeur du nombre de pixels est supérieure à la valeur de seuil, nous récupérons sa coordonnée. Tant que la valeur des pixels suivant est supérieur au seuil, nous continuons le parcours de l'histogramme.
Nous avons ainsi récupéré les coordonnées de chaque début le ligne et début de colonne. Nous le stockons dans un tableau qui correspond aux coordonnées des lignes. Nous recommençons ce processus 10 fois, car il y a 10 lignes au sodoku en X et Y.

6 Découpage de la grille

Afin de pouvoir proprement découper la grille pour obtenir les images de chaque case, il a d'abord fallu supprimer en parti le bruit de l'image. Nous nous sommes cependant rendu compte que cela pouvait faire disparaître certaines lignes sur certaines images, c'est pourquoi il a d'abord fallu les épaissir.

6.1 La dilatation

La dilatation est un procédé qui consiste à épaissir les lignes déjà présentent sur la grille de sudoku. Pour ce faire, il faut parcourir chaque pixel de la grille. Pour l'instant, nous appliquons la dilatation sur les images en noir et blanc. Ainsi, nous dilatons chaque pixel noir de cinq pixels verticalement et de cinq pixels horizontalement. Ce procédé permet d'élargir et de combler certaines lignes. Cela élargit également légèrement les chiffres, mais ce n'est pas un problème grâce à la fonction suivante, celle qui permet de réduire les bruits.

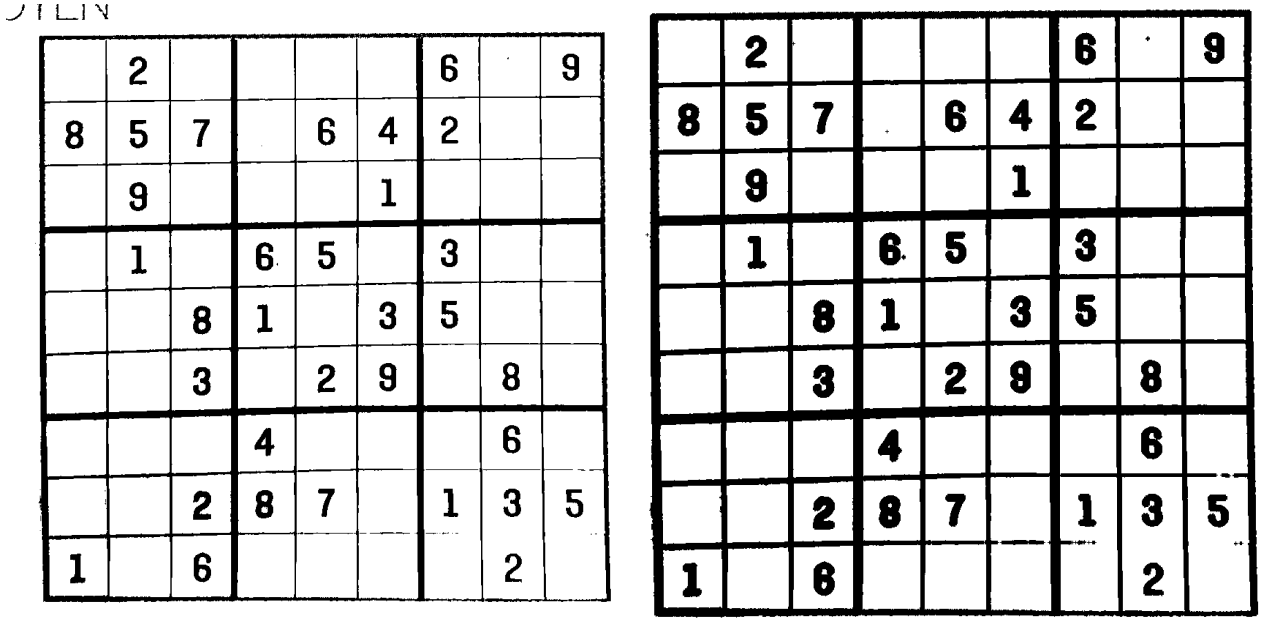

FIGURE 5 – Dilatation des lignes

6.2 Réduction de bruit

L'objectif de ce traitement est de réduire les bruits de l'image pour permettre une meilleure reconnaissance de la grille. Voici son fonctionnement. Chaque pixel possède huit pixels voisins. Sur notre image, nous ne voulons conserver que les chiffres et la grille. Or, il se trouve que les lignes qui composent la grille et que les chiffres présents sur cette dernière sont composés d'un amas de pixels noirs, à la différence des bruits qui sont généralement des imperfections de l'image qui se traduisent par des petits groupes espacés de pixels noirs.
Pour retirer le bruit, nous allons donc supprimer tous les pixels qui ne possèdent pas leurs huit voisins noirs. Les lignes et les chiffres vont donc devenir plus fins, et les bruits vont en grande partie disparaître. Notre image ainsi traitée est enfin prête pour le découpage.

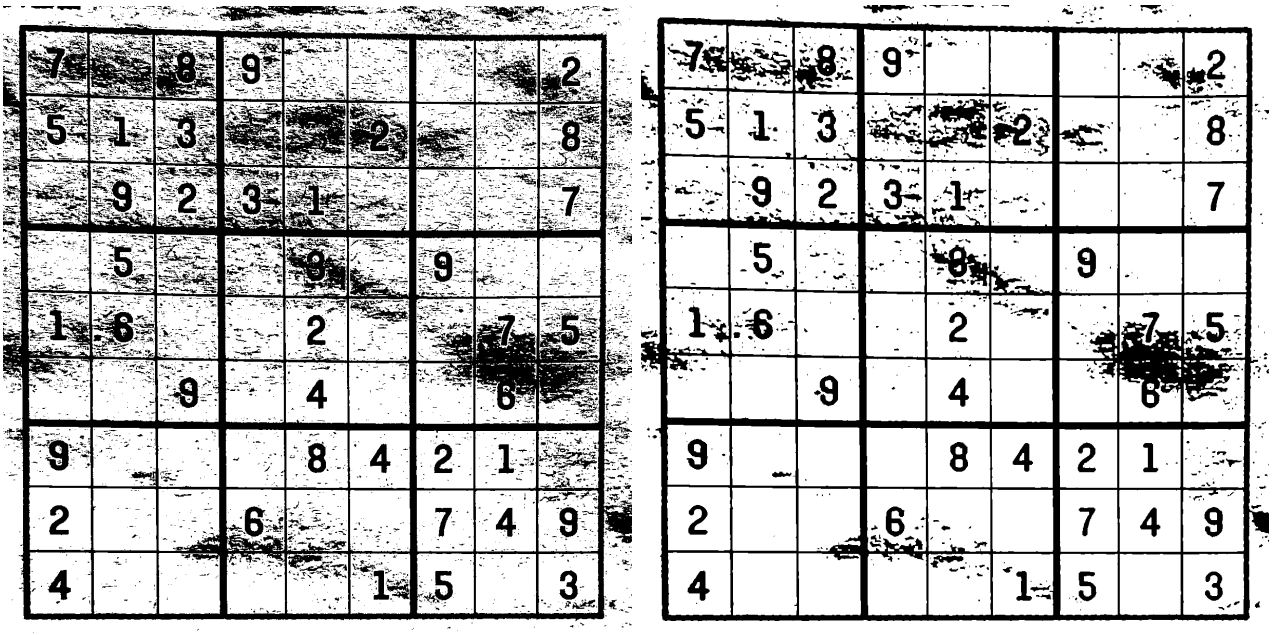

FIGURE 6 – Réduction de bruit

6.3 Le découpage de la grille

Après avoir récupérer les coordonnées des dix lignes présentes en X et des dix lignes présentes en Y, le découpage de la grille peut commencer. Pour cela, il a fallu parcourir la grille en fonction des coordonnées. Il faut ainsi parcourir la grille à partir du premier point X jusqu'au suivant, et du premier point Y jusqu'au suivant, et ceci ainsi de suite pour toutes les coordonnées (X,Y) des lignes.
Pour chaque parcours, il faut créer une nouvelle image ayant pour dimensions celles qui séparent les lignes, puis la remplir en copiant les pixels de l'image de référence dans cette nouvelle image. Une fois cela fait, avant de passer aux coordonnées suivantes, il faut sauvegarder l'image, dans un dossier spécial pour éviter d'encombrer le dossier actuel. Chacune des 81 images est sauvegardée avec comme nom les coordonnées du positionnement de sa case dans la grille.

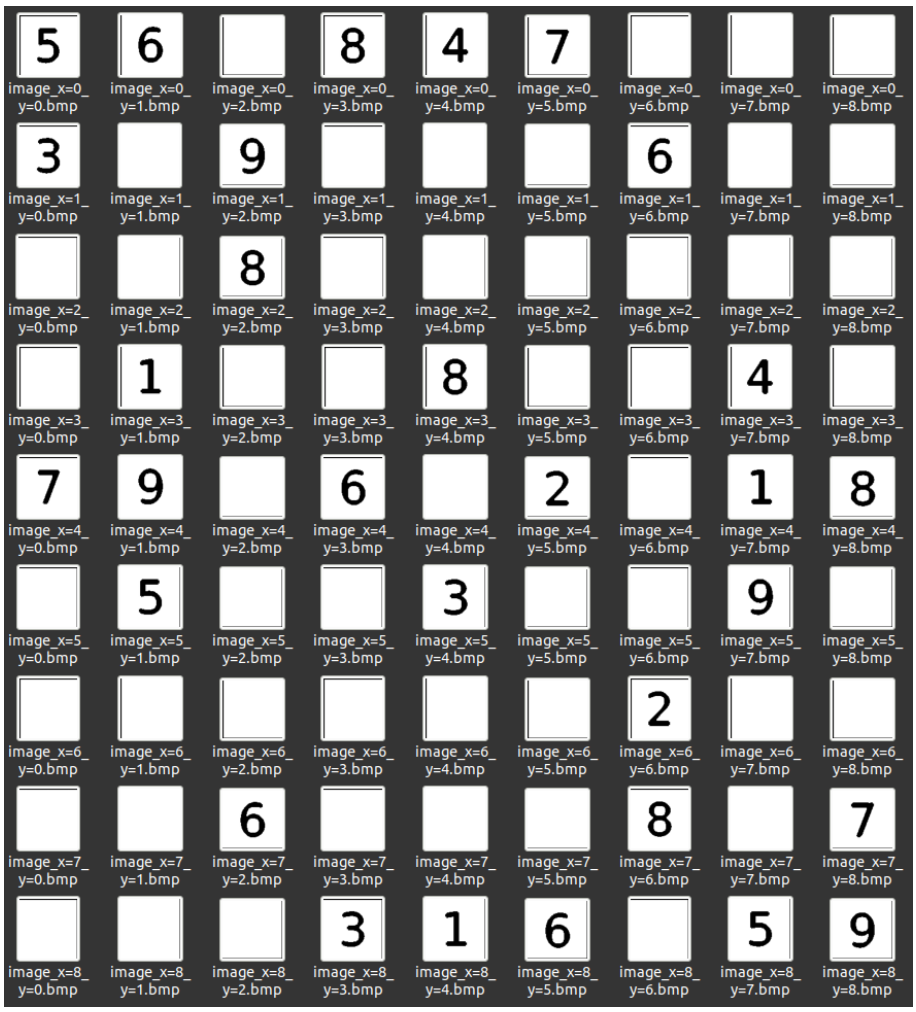

FIGURE 7 – Découpage des images

7 Réseau de neurones

7.1 Qu'est-ce qu'un réseau de neurones ?

Un neurone est une représentation mathématique et informatique du neurone biologique. Il reproduit certaines de ses caractéristiques biologiques, en particulier les dendrites, axones et synapses, au moyen de fonctions et de valeurs numériques. Un neurone élémentaire peut manipuler des valeurs binaires, représentées par 0 et 1, ou réelles.

L'ensemble que forme l'agencement de ces neurones inter-connectés est un réseau neuronal capable d'apprendre et de se modifier en fonction de cet apprentissage. Il est à sens unique et est constitué de couches de neurones dans lesquelles la sortie d'un neurone ne peut seulement alimenter l'entrée d'un neurone situé dans la couche postérieure.

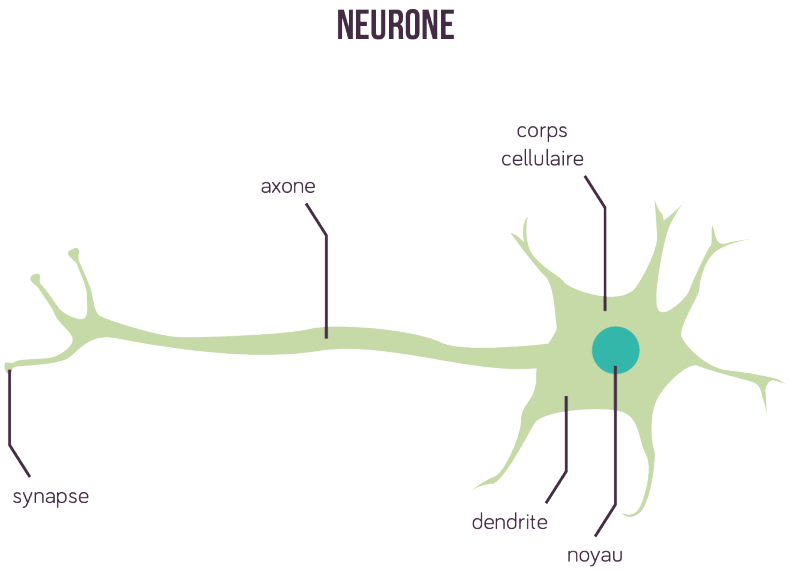

FIGURE 8 – Schéma d'un neurone en biologie

7.2 L'architecture des réseaux de neurones

Les réseaux de neurones sont constitués de plusieurs couches (au minimum 2), elles même construites par le rassemblement d'un ou plusieurs neurones. Ces couchent portent un nom en fonction de leur place dans le réseau.

- **Couche d'entrée :** les neurones de cette couche contiendront les informations / données que l'on souhaite tester

- **Une ou plusieurs couches cachées :** les neurones contenus dans cette couche entrent dans un processus de calcul complexe. Leurs valeurs sont seulement exploitées par le réseau de neurone et participe ainsi au bon apprentissage de celui-ci.

- **Couche de sortie :** les neurones de sortie nous donneront le résultat au problème présenté en couche d'entrée.

Les neurones qui composent une couche sont reliés à l'ensemble des neurones de la couche qui suit, sauf dans le cas de la couche de sortie qui marque la fin du réseau. Il peut arriver que les neurones d'entré soient connectés à la couche de sortie bien qu'il y ait des couches cachées entre celles-ci, dans le but d'améliorer l'apprentissage.

Chaque connexion entre deux neurones possède un poids qui permet de faire les différents calculs de propagation qui interviennent dans les réseaux neuronaux.

De plus, chaque neurone possède une valeur de sortie (comprise entre 0 et 1) qui est la valeur à tester pour la couche d'entrée et le résultat d'un calcul pour les autres couches, cette valeur correspond à l'information que chaque neurone transmet aux neurones suivants.

7.3 Implémentation de l'architecture

Pour la création de l'architecture complète du réseau neuronal, nous avons tout d'abord créé la structure du neurone dans un fichier à part puis de même pour celle correspondante aux couches du réseau. L'ensemble de ces nouvelles structures sont ensuite rassemblées dans un nouveau fichier en fonction des quantités et proportions demandées par l'utilisateur du réseau avec notamment l'initialisation des poids entre chaque neurone de couches adjacentes.

7.4 Apprentissage

Le but de l'apprentissage est de permettre au réseau de neurones de s'améliorer en apprenant littéralement de ses erreurs. Cet apprentissage se réalise à travers plusieurs étapes afin d'obtenir de nouvelles valeurs de poids que l'on pourra mettre à jour dans l'ensemble du réseau. En somme, cela consiste simplement à modifier les poids de chaque connexion.

Plus les cycles d'apprentissage sont nombreux, plus les poids des connexions entre les neurones convergent vers leur valeur optimale lors du travail du réseau neuronal. Cela le mène donc à un améliorer ses taux de réussite face à la résolution de problèmes auxquels il se forme.

Le pas d'apprentissage est une valeur qui permet de régler la vitesse à laquelle le réseau apprend mais aussi la précision de cet apprentissage. Plus le pas sera grand, plus l'apprentissage sera rapide, moins il sera précis, et inversement.

7.5 La propagation vers l'avant (de l'entrée vers la sortie)

L'algorithme de calcul en avant permet de trouver la(les) valeur(s) de sortie du réseau de neurones en fonction de la(les) valeur(s) d'entrée en parcourant le réseau du début vers sa fin et en appliquant les différents calculs que nous allons définir.

La première étape est de calculer une valeur d'activation pour chacun des neurones autrement appelé le biais. Ce biais est une fonction de combinaison elle-même jointe à une fonction d'activation. La fonction de combinaison retourne une valeur calculée à partir des différentes entrées et des poids qui y sont associés.

Cette valeur est ensuite passée en argument à la fonction d'activation dont le résultat sera la sortie finale du neurone. Nous avons choisi d'utiliser la fonction sigmoïde, une fonction qui a la particularité de donner des images entre 0 et 1 avec une progression à partir de petits débuts qui s'accélère et se rapproche d'un point culminant à la fin de l'intervalle.

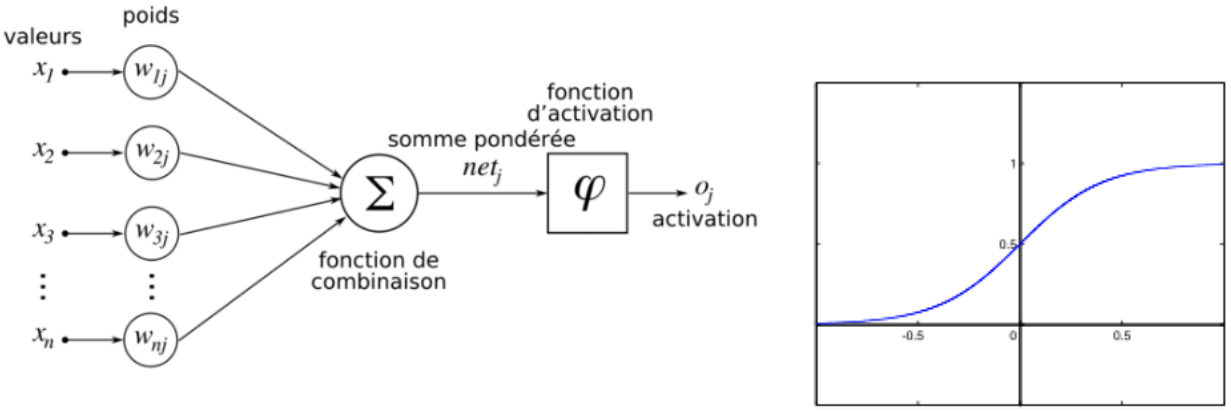

FIGURE 9 – Propagation avant et allure de la fonction d'activation (sigmoïde) g(x)= 1/(1+e(-x))

7.6 La rétropropagation du gradient (de la sortie vers l'entrée)

Cet algorithme en particulier qui intervient dans l'apprentissage. Il permet de modifier les poids des neurones pour devenir plus précis. Appliquer cet algorithme nécessite la connaissance de la valeur de sortie que l'on récupère après l'essai de certaine valeur en entrée. Cette valeur récupérée est ensuite comparée au résultat que l'on devrait obtenir pour les mêmes valeurs en entrée. Que les valeurs comparées soient égales ou non, le processus s'opère donc en quatre étapes distinctes.

- Tout d'abord il faut effectuer une propagation en avant dans le réseau neuronal pour les entrées de chaque essai et stocker ce que renvoi le réseau à chacun des tests.

- Ensuite par un calcul de comparaison, on récupère la valeur d'erreur représentée par la différence entre les résultats collectés et les résultats que l'on devrait normalement obtenir (valeurs théoriques)

- L'algorithme met à jour le poids de chaque connexion en partant de la dernière couche de neurones vers la première. Cet algorithme remonte l'ensemble du réseau d'où le nom rétropropagation. Les poids synaptiques qui contribuent plus à une erreur seront modifiés de manière plus importante que les poids qui provoquent une erreur marginale.

- Afin que l'apprentissage soit efficace, il faut répéter cet algorithme un grand nombre de fois afin que le réseau finisse par toujours avoir une bonne réponse (pour un pas peu élevé, il faut compter entre 1000 et 10000 générations pour une simple porte logique).

7.7 Réseau de neurones : Porte XOR et portes logiques

Pour notre réseau de neurones capable d'apprendre la porte XOR, nous utilisons un réseau de la forme 2-2-1 c'est-à-dire 2 neurones d'entrée dans la première couche, 2 neurones dans la seule couche cachée que comporte notre réseau et un seul neurone pour la couche de sortie. Pour faire fonctionner ce réseau correctement, il est ensuite possible de choisir les entrées possibles avec les sorties correspondantes (ici 4 voir la table de vérité).

Table de vérité XOR (OU exclusif)		
A	B	S
0	0	0
1	0	1
0	1	1
1	1	0

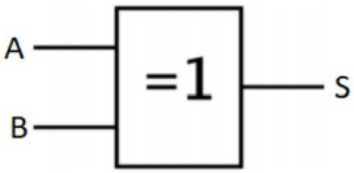

FIGURE 10 – Table de vérité du XOR et sa représentation

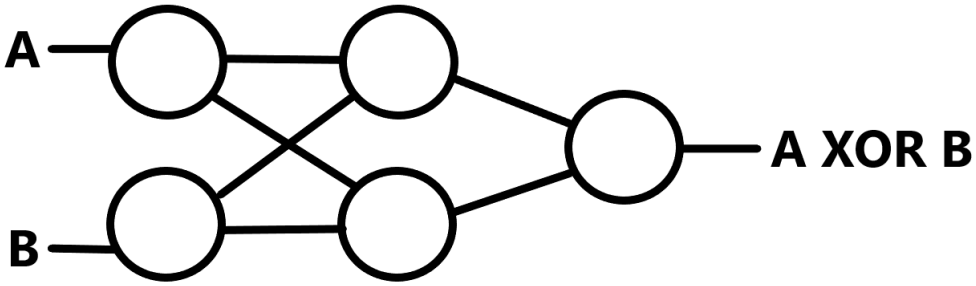

FIGURE 11 – Représentation de la construction neuronale de la porte XOR

Nous avons choisi un petit pas (pas = 0.15) pour l'apprentissage de notre réseau neuronal, ce qui nécessite un grand nombre de générations pour arriver à de bons résultats, environ 10000 pour une efficacité complète. Comme il est possible de le remarquer aux premières générations, le résultat renvoyé pour les 4 possibilités sont très souvent faut.

Les poids du réseau après apprentissage sera sauvegardé dans un fichier nommé « weights.txt ».

Il est ensuite possible de demander à tester des valeurs, cette fois non pas dans un but d'apprentissage, pour vérifier que la sortie renvoie le bon résultat.

7.8 La porte XOR

8 Résolution du sudoku

Contrairement à l'être humain, avec un ordinateur on peut essayer toutes les combinaisons possibles pour un Sudoku et voir si cette dernière est valide. Aussi appelé méthode par force brute où méthode de "backtracking", l'algorithme que nous avons décidé d'implémenter repose sur 4 étapes.
Premièrement on vérifie la validité du fichier grid_00, on le parse pour le convertir en tableau de caractère que l'on peut facilement manipuler en C.
Deuxièmement, on essaye sur chaque case en partant de la plus en haut à gauche puis de la droite vers la gauche de mettre la valeur 1. Si la valeur est déjà présente dans la ligne/colonne/carré alors on essaye avec 2 puis 3 etc, si on ne peut mettre aucune valeur alors notre grille est impossible à résoudre, il faut donc retirer les chiffres que nous avons entrée précédemment et recommencer avec le chiffre suivant, c'est pour cela que caractérisons l'algorithme de "backtracking", c'est la troisième étape.
Comme vous l'avez compris, nous allons faire des appels récursifs tant que nous n'avons pas trouver la combinaison qui nous fait remplir entièrement le tableau.
La complexité d'un tel algorithme est O(9^m) avec m le nombre de casses vides or ce n'est pas un problème car la pire des situations ne se produit presque jamais.

9 Sauvegarde de la grille

C'est la sauvegarde qui vient cloturer la méthode de résolution du Sudoku, nous l'enregistrons sous la forme la forme du cahier des charges dans le fichier grid_00.result, qui est identique au format du fichier initial grid_00.
Une dernière étape à faire pour la prochaine soutenance est l'ajout des chiffres sur l'image pour avoir un retour visuel de notre grille.

10 Affichage de la grille résolue

11 Ressenti personnel

Alexandre Devaux-Riviere :

La première moitié de notre projet est déjà entièrement réalisée, mais il reste encore beaucoup de travail surtout au niveau de la documentation pour les procédés de traitement et de séparation de l'image, et au niveau l'optimisation.
Il existe une forte cohésion au sein de notre groupe ainsi qu'une bonne organisation qui soulage, surtout lorsque les périodes sont chargées. L'OCR est une expérience très enrichissante et je vais donner mon maximum pour que la dernière partie qu'il reste à faire se déroule dans une bonne atmosphère et soit une totale réussite.

Jules GIROD :

On est déjà à la moitié du projet et il nous reste encore beaucoup de travail que nous avons hâte de faire car nous ne sommes pas là pour procrastiner ! Mes bases en C et sur la SDL m'ont permis de me sentir à l'aise rapidement et ne pas me sentir perdu face à un nouveau

langage.
Avoir un groupe qui travaille avec une forte cohésion est un soulagement et une source de motivation car je n'ai pas envie de revivre ma très mauvaise expérience avec le projet S2.

Esteban Arroyo :

Nous sommes déjà à la moitié du semestre et il nous manque énormément de choses à faire, mais vu ce que nous avons effectué en détection, conversion et sur l'IA, nous pouvons être fier même si la détection doit être améliorée pour éviter certains cas erreur. De plus malgré mes faibles expériences en C et en SDL, le travail de groupe nous permet de s'entraider, de mieux comprendre les subtilités du C et ainsi avancer plus vite. Pour finir, je suis heureux de la bonne entente du groupe car cela nous permet d'avancer sur de bonnes bases de travail.

Romain Ludet :

Ayant vu la présentation de ce projet par un étudiant lors d'une journée portes ouvertes de l'Epita, j'avais très envie de le réaliser à mon tour ! Le moins que l'on puisse dire, c'est que je ne suis pas déçu. C'est un super projet, très intéressant et il nous pousse à réfléchir et à nous documenter (la base en programmation). Je n'avais jamais utilisé auparavant la librairie SDL, mais j'avais de bonnes bases en C, donc je me suis rapidement adapté. D'autant plus en étant en vocal sur Discord avec un super groupe, notamment le soir !

12 Nous contacter

— Alexandre Devaux-Rivière : alexandre.devaux-riviere@epita.fr
— Jules GIROD : jules.girod@epita.fr
— Esteban Arroyo: esteban.arroyo@epita.fr
— Romain ludet : romain.ludet@epita.fr