

BY SUDOKUCRAFTERS

EPITA 2023-2024

Projet OCR

Anthony LAWANDOS
Lucile PELOU
Lucas DEMULIERE
Lucil FINKELSTEIN

6 novembre 2023

Table des matières

1	Introduction	2
2	Présentation du groupe	3
3	Répartition des tâches	4
4	Prétraitement	5
4.1	Rotation manuelle	5
4.2	Suppression des couleurs	6
5	Détection et découpage de la grille	7
5.1	L'histogramme	7
5.2	La transformée de Hough	8
5.3	Détection de la grille et des cases	10
6	Réseau neuronal	12
6.1	Mini réseau	12
6.2	Le réseau principal	12
6.2.1	Structure globale	12
6.2.2	Initialisation du réseau	13
6.2.3	Propagation avant :	13
6.2.4	Rétropropagation	13
6.2.5	Descente de gradient	14
6.2.6	XOR	14
6.3	Conclusion	15
7	Résolveur	16
7.1	Solubilité du Sudoku	16
7.2	Résolveur	16
7.3	Sauvegarder des Résultats	17
8	Avancées générales	18
8.1	Première soutenance	18
8.2	Avancés bonus	19
8.3	Deuxième soutenance	19
9	Conclusion	20
10	Annexes	21

1 Introduction

Ce rapport vous présentera les premières avancées et implémentations de ce projet. Vous y trouverez aussi la présentation des membres de *SUDOKUCRAFTERS*. Pour rappel pour ce premier rapport, voici les différents éléments qui doivent être implémentés :

- 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

Tous ces éléments constituent ce premier avancement. De plus, nous avons pu commencer à implémenter la suite de ce projet (hors prérequis de cette première soutenance). 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 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.

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.

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.

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 représente une opportunité pour approfondir mes compétences en programmation en langage C, ainsi que pour en acquérir de nouvelles.

3 Répartition des tâches

Afin de mieux réaliser les différentes tâches demandées pour cette première soutenance, nous nous sommes répartis les tâches de cette manière :

Tâches Membres	Lucas	Anthony	Lucil	Lucile
Prétraitement			Principal	
Rotation manuelle			Principal	
Détection de la grille et cases	Secondaire			Principal
Découpage de l’image				Principal
Résolution de la grille	Principal			
Réseau de neurones	Secondaire	Principal		

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

4 Prétraitement

Afin d'avoir une propre pour la détection de la grille et son analyse suivante, il faut réaliser différentes étapes sur les images.

4.1 Rotation manuelle

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 :

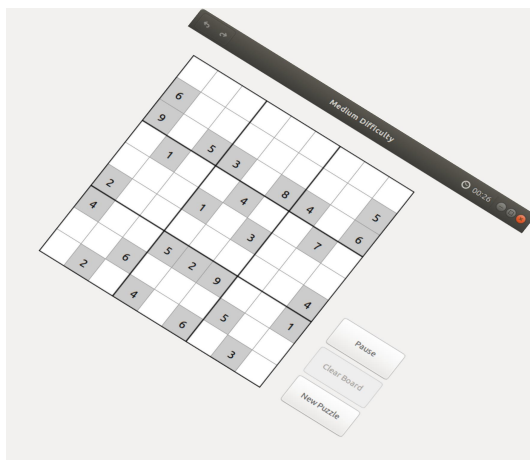


FIGURE 2 – Avant rotation

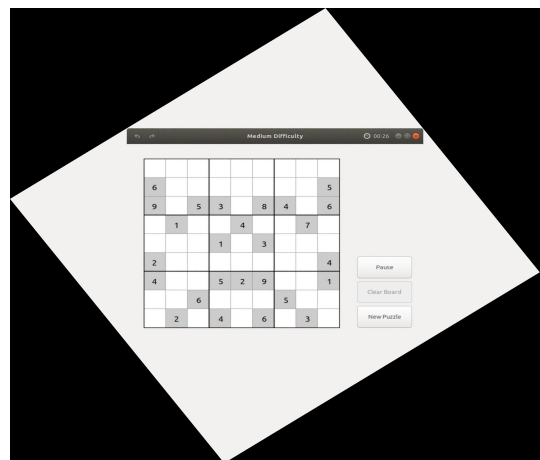


FIGURE 3 – Après rotation de -35 degré

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.

4.2 Suppression des couleurs

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.

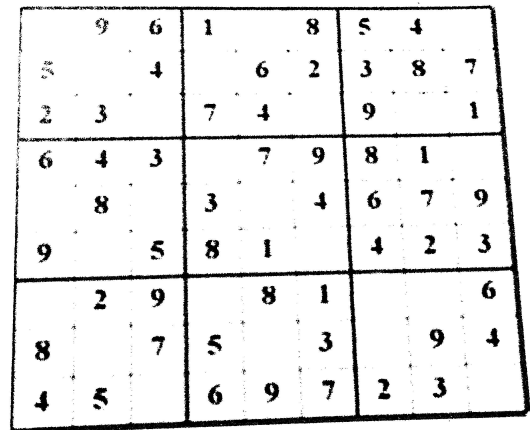
En résultante, nous obtenons une image contrastée (en noir et blanc).

Voici un exemple de ce programme :



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

FIGURE 4 – Avant application des contrastes



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

FIGURE 5 – Après application des contrastes

De plus, d'autres modifications seront réalisés sur le images afin de retirer les tâches et les bruits parasites avec notamment les filtres Médian (utilisé avec le filtre de flou pour harmoniser l'image) ainsi que la binarisation de l'image pour son utilisation dans le découpage de la grille.

5 Détection et découpage de la grille

5.1 L'histogramme

Pour détecter la grille nous avons en premier lieu voulu appliquer la méthode de la transformée de Hough, seulement nous n'arrivions pas à l'appliquer comme nous le souhaitons. Nous avons donc mis en place un algorithme permettant de détecter une potentielle ligne 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 détectée. 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.

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

FIGURE 6 – Lignes sur une image parfaite

FACILE									Rayez dans la grille restantes, trouvez la
7	8	9						2	ABOYER
5	1	3			2			8	ABRICOTIER
9	2	3	1					7	AGRESSION
									ALPIN
	5			3		9			AMANT
1	6			2			7	5	AMNISTIE
									AOÛTAT
		9		4			6		APOLOGIE
									ASSORTI
9				8	4	2	1		ATTRAIT
2			6			7	4	9	AUTEL
4					1	5		3	AUTRUI
									BÉGONIA
									BÉOTIEN
									BRICOLAGE

FIGURE 7 – 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 minimums 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.

FACILE										Rayez dans la grille restantes, trouvez la
27	7		8	9					2	ABOYER
	5	1	3			2			8	ABRICOTIER
		9	2	3	1				7	AGRESSION
										ALPIN
		5			3		9			AMANT
	1	6			2			7	5	AMNISTIE
			9		4			6		AOUTAT
										APOLOGIE
	9				8	4	2	1		ASSORTI
	2			6			7	4	9	ATTRAIS
										AUTEL
	4					1	5		3	AUTRUI
										BÉGONIA
										BÉOTIEN
										BRICOLAGE

FIGURE 8 – Nouvelles lignes trouvées

Les lignes vertes symbolisent la simplification des lignes déjà trouvées auparavant. Les lignes bleues ont été ajoutées avec le second algorithme.

5.2 La transformée de Hough

En parallèle de notre méthode d'histogramme nous avons continué à travailler sur l'implémentation de la transformée de hough qui contrairement à la première méthode nous permet de détecter les lignes avec beaucoup plus de précisions, y compris celles non droites.

Avant de pouvoir utiliser notre algorithme chargé de faire la transformée de Hough nous devons appliquer un filtre pour détourner les contours de l'image. Pour cela nous avons décidé d'utiliser l'algorithme de Sobel. Cette méthode repose sur la convolution 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 : $\sqrt{Gradient_{horizontal}^2 + Gradient_{vertical}^2}$ 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.

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.

La transformée de Hough, en se concentrant sur 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 :

$$\rho = x \times \cos(\theta) + y \times \sin(\theta)$$

ρ est la distance perpendiculaire de l'origine à la ligne, et θ 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 ρ et θ . 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'incrémentations indiquent la présence probable d'une ligne dans l'image, avec les paramètres (ρ, θ) correspondant à la position de ces cellules dans l'accumulateur.

Voici un exemple de ce qu'on l'on a pu obtenir :

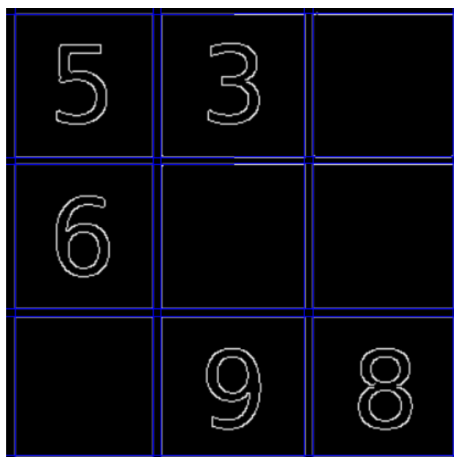


FIGURE 9 – Exemple avec la transformée de Hough

Nous ne sommes pas encore parvenu à l'implémenter de manière 100% opérationnelles, c'est pourquoi nous utilisons la méthode de l'histogramme pour l'instant.

5.3 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.

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

FIGURE 10 – Grille trouvée pour le sudoku de la figure 4

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 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.

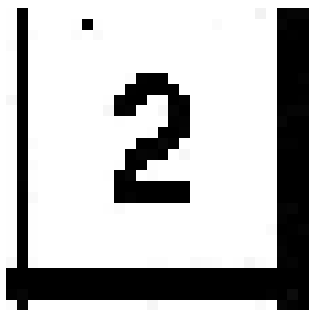


FIGURE 11 – Case coupée avec résidus



FIGURE 12 – Suppression des résidus

6 Réseau neuronal

6.1 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.

6.2 Le réseau principal

6.2.1 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.

6.2.2 Initialisation du réseau

Cette partie est découpée en deux.

Création du réseau qui crée notre réseau (couches et neurones) selon les valeurs mises en paramètre.

Initialisation du réseau qui attribue des valeurs aléatoires aux poids et aux biais des neurones, ou cette initialisation peut se faire avec des valeurs déjà sauvegardées.

6.2.3 Propagation avant :

La fonction de Propagation avant (forward propagation) est essentielle dans le fonctionnement de notre réseau de neurones. Son rôle est de calculer les valeurs de chaque neurone dans le réseau en utilisant la fonction d'activation sigmoïde. La formule de la fonction sigmoïde est la suivante :

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

où x est une valeur calculée à partir du biais, des poids et des valeurs des neurones de la couche précédente. 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 la formule de la sigmoïde. 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.

6.2.4 Rétropropagation

La rétropropagation est une autre 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.

6.2.5 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.

6.2.6 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.

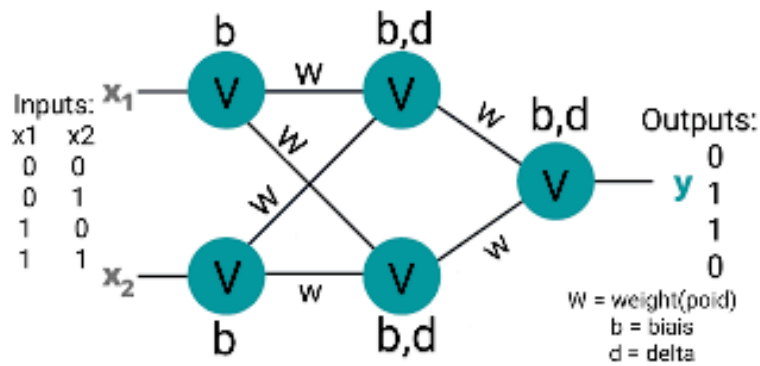


FIGURE 13 – Représentation du réseau de neurones pour la fonction XOR

6.3 Conclusion

Avec le nouveau réseau, nous avons pu entraîner notre programme à comprendre la fonction XOR, et maintenant nous avons tout ce dont nous avons besoin pour l'entraîner à détecter les chiffres cachés derrière les images. Cet objectif est pour la dernière soutenance.

7 Résolveur

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

7.1 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.

7.2 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.

7.3 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 14 – Rendu final

8 Avancées générales

Voici un tableau représentatif de nos avancées et de nos réalisations pour la première soutenance.

Tâches	Réalisé	Prévu
Prétraitement	50%	50%
Rotation manuelle	100%	100%
Détection de la grille et des cases	75%	100%
Découpage de l'image	100%	100%
Résolution de la grille	100%	100%
Réseau de neurones	70%	65%

FIGURE 15 – Tableau des avancées pour la première soutenance

8.1 Première soutenance

Si nous comparons notre avancement par rapport aux attentes demandées pour la première soutenance, nous sommes plus ou moins en phase avec celles-ci. 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.

La détection de la grille nous a posée quelques problèmes, nous avons du mal à implémenter la méthode de la transformée de Hough. Pour palier à cela nous avons mis en place une deuxième méthode tout en continuant de travailler sur la transformée en parallèle. Pour le découpage des cases, tout est fonctionnel et elles sont également récupérées 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.

Pour ce qui est du réseau de neurones, il est bien capable d'apprendre la fonction XOR. Toutes les bases du réseau sont implémentées, il ne nous reste plus qu'à l'entraîner et ajuster les fonctions déjà implémentées.

8.2 Avancés bonus

Nous avons également avancé sur les objectifs de la deuxième soutenance comme l'affichage de la grille après sa résolution ainsi que sa sauvegarde. Le réseau de neurones a aussi prit un peu d'avance sur l'objectif initial, nous sommes plutôt sur la suite de son développement car toutes les bases sont déjà bien en place.

8.3 Deuxième soutenance

Voici un tableau sur les objectifs prévus pour la deuxième soutenance et l'avancement déjà fourni. À noter que nous avons ajouté la détection de la grille et des cases sur lequel nous n'avons pas réussi à avancer autant que nous l'aurions voulu lors de ce premier temps de travail.

Tâches	Réalisé	Prévu
Détection de la grille et des cases	75%	100%
Prétraitement	50%	100%
Affichage et sauvegarde de la grille	100%	100%
Réseau de neurones	70%	100%
Interface graphique	0%	100%

FIGURE 16 – Tableau des avancées pour la deuxième soutenance

9 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 de nos avancées et 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.

10 Annexes

Table des figures

1	Tableau de la répartition des tâches au sein du groupe	4
2	Avant rotation	5
3	Après rotation de -35 degré	5
4	Avant application des contrastes	6
5	Après application des contrastes	6
6	Lignes sur une image parfaite	7
7	Lignes sur une image déformée	7
8	Nouvelles lignes trouvées	8
9	Exemple avec la transformée de Hough	10
10	Grille trouvée pour le sudoku de la figure 4	11
11	Case coupée avec résidus	11
12	Suppression des résidus	11
13	Représentation du réseau de neurones pour la fonction XOR	15
14	Rendu final	17
15	Tableau des avancées pour la première soutenance	18
16	Tableau des avancées pour la deuxième soutenance	19