

OCR : Rapport de projet

Alexis Cognet

Alexandre Posta

Flavien Geoffray

Quentin Gillet

Decembre 2022

Table des matières

1	Retours personnels des membres de l'équipage	3
1.1	Alexis Cognet	3
1.2	Alexandre Posta	3
1.3	Flavien Geoffray	4
1.4	Quentin Gillet	5
2	Introduction	6
3	Solver	7
3.1	Manipulation du Sudoku	7
3.2	Gestion des erreurs	7
3.3	Résolveur	7
3.4	Sauvegarde du résultat	7
4	Prétraitement	8
4.1	Sauvegarde de l'image	8
4.2	Filtre Grayscale	8
4.3	Filtre des Contrastes	9
4.4	Réduction de Bruit (Filtre Median)	9
4.5	Binarisation (Filtre à seuil Adaptatif)	11
4.6	Filtre d'inversion	12
4.7	Filtre Sobel	12
4.8	Réultat Final	13
5	Ségmentation	14
5.1	Transformation d'Hough	14
5.2	Reconnaissance des Cases	16
5.2.1	Récupération des carrés	16
5.2.2	Récupération des cotés de la grille	18
5.3	Découpage et Chargement des Cases	19
6	Réseau de neurones	21
6.1	XOR	21
6.1.1	Structure	21
6.1.2	Forward Propagation	21
6.1.3	Back Propagation	22
6.2	Digit Recognition	23

6.2.1	Structure	23
6.2.2	Forward Propagation	24
6.2.3	Back Propagation	24
6.3	Jeu de données pour l'entraînement	24
6.4	Entrainement du réseau	25
6.5	Sauvegarde des poids et des biais	26
6.6	Résultats et difficultés de la reconnaissance de chiffre	27
7	Fonctionnement global d'OCR	28
8	Interface	30
8.1	Interface globale	30
9	Ressentis et Organisation	32
9.1	Resumé des Avancements	32
9.2	Dificultés	32
9.3	Points Positifs	32
10	Conclusion	33

1 Retours personnels des membres de l'équipage

1.1 Alexis Cognet

Issu d'une terminale générale option Math/SI je suis passionné depuis longtemps par l'informatique, idée que j'ai renforcée après avoir fait mon stage de 3ème dans une entreprise d'informatique. Ce projet m'a permis de m'améliorer en programmation et de découvrir le traitement d'image ainsi que la création d'une interface ce qui m'a pas mal plu.



FIGURE 1 – Alexis Cognet

1.2 Alexandre Posta

Depuis tout petit passionné par l'informatique, j'ai décidé de rejoindre EPITA dans le but de me spécialiser dans ce domaine qui me plaît tant. L'informatique fait partie de la vie de tous les jours et son application peut se faire dans tous les domaines. J'ai trouvé très intéressant ce projet ocr dans la mesure où il nous a permis de voir plus loin que ce que nous avions jamais vu jusqu'à là. Le fait de s'intéresser au traitement d'image comme à l'intelligence artificielle nous a ouvert de nouvelles connaissances et de nouvelles compétences. J'appréhendais beaucoup l'OCR avant de le commencer, le fait de faire un projet aussi complet dans un langage que je ne connaissais pas. Mais je suis très bien entouré avec des personnes motivées ce qui facilite la tâche. Je suis très content de ce qu'on a pu fournir durant la première partie de ce projet, nous continuerons sur cette lancée.



FIGURE 2 – Alexandre Posta

1.3 Flavien Geoffray

L'informatique et les nouvelles technologies de manière général m'ont toujours beaucoup intéressé. C'est d'ailleurs pourquoi j'ai choisis l'EPITA. Le projet OCR m'a permis de faire un premier pas dans un domaine que j'ai toujours eu envie de découvrir : l'IA. Je dois dire que la tache n'a pas été facile, mais j'ai beaucoup aimé créer les réseaux de neurones. Je suis très fier de ce que notre équipe à accomplis, bien que notre projet ne soit pas parfait. Ce projet m'a également conforté dans mon envie de choisir la majeure IA.



FIGURE 3 – Flavien Geoffray

1.4 Quentin Gillet

Bonjour, Quentin, j'adore l'informatique depuis que je suis tout petit, je programme depuis des années. J'ai beaucoup aimé le projet S2, mais je dois avouer que j'ai préféré le projet OCR ! Ce projet était plus compliqué que le projet S2 mais ça m'a permis d'apprendre énormément sur la reconnaissance et le traitement d'image. Je suis très fier que ce qu'on a accomplis même si certaines parties du projet ne fonctionne pas totalement.



FIGURE 4 – Quentin Gillet

2 Introduction

Ce rapport de projet retrace tout le travail effectué sur le projet OCR de ce troisième semestre. Il détaille l'implémentation de chaque partie du projet mais aussi les problèmes rencontrés, les solutions trouvées et les décisions prises durant son développement.

Tableau de répartition				
Matière	Alexandre	Alexis	Quentin	Flavien
Solver				X
Prétraitement	X		X	
Ségmentation	X		X	
IA		X		X
GUI		X		

3 Solver

3.1 Manipulation du Sudoku

Tout d'abord, pour arriver à résoudre une grille de sudoku à l'aide d'un programme, il fallait trouver un moyen de représenter le fichier d'entrée, c'est-à-dire l'image du sudoku à résoudre, en un fichier utilisable par la machine.

Le sujet nous forçait à utiliser un format spécial de sudoku, en fichier texte. Le but était donc de résoudre le sudoku sur ce fichier texte. Pour ce faire, le moyen le plus simple était de transformer le fichier, contenant neuf lignes de neuf caractères, en un tableau à une dimension, de longueur 81.

3.2 Gestion des erreurs

Pour que le résolveur de sudoku soit complet et fonctionne correctement, il était important de mettre en place une gestion d'erreur, qui teste si le fichier d'entrée est correct et correspond à un format de sudoku solvable. Une fois que nous avions le tableau à une dimension contenant les 81 caractères du sudoku, vérifier les erreurs n'est pas compliquer.

L'algorithme teste si les éléments du tableau sont bien des chiffres ou des «cases vides». Si ce n'est pas le cas, une erreur est renvoyée.

Un teste des lignes, des colonnes et des «box» est aussi réalisé pour vérifier que le sudoku est bien correct.

3.3 Résolveur

L'algorithme de résolution est assez simple, une fois que la gestion des erreurs a été faite. L'algorithme utilise le «Backtracking» qui est une méthode dite «brute force» et qui teste toutes les possibilités case par case en récursion. Il retourne en arrière s'il rencontre une erreur, pour tester une autre valeur et ainsi trouver la solution.

3.4 Sauvegarde du résultat

Une fois le tableau complété avec les bonnes valeurs. Le résultat est sauvegardé sous la même forme que le fichier d'entrée. 9 lignes de 9 caractères. Cette fois les 9 caractères sont des chiffres.

4 Prétraitemet

Le prétraitemet de l'image va permettre deux choses.

Premièrement, le découpage de l'image pourra être effectué dans la mesure où la grille du sudoku sera facilement détectable.

Ensuite, il va permettre à l'intelligence artificielle de différencier chaque chiffre de la grille, il a été grandement amélioré par rapport à la dernière soute-nance. Notamment la transformation de Houg qui détecte les lignes beaucoup plus efficacement et qui n'a même plus besoin de réduction de ligne.

4.1 Sauvegarde de l'image

Pour sauvegarder l'image dans un fichier, nous avons utilisé le format bitmap très répandue pour les fichiers d'images pixelisés. Cette sauvegarde nous a été très utile pour le débogage de nos programmes de prétraitemet puisque qu'il permet d'afficher à quoi ressemble à notre sudoku après chaque étape qu'on lui applique. Cette partie nous a été indispensable pendant le débogage de nos programmes.

4.2 Filtre Grayscale

Pour débuter le traitement de l'image, nous avons premièrement appliqué le filtre grayscale. Pour chaque pixel de l'image la fonction utilisée est :

$$f(P) = 0.3 * Pr + 0.59 * Pg + 0.11 * Pb$$

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 5 – Image 1 et 6 après application du filtre grayscale

4.3 Filtre des Contrastes

Nous avons ensuite décidé d'utiliser le filtre de contraste. Ce filtre permet tout simplement de réduire la différence entre les parties les plus contrastées et les moins contrastées de l'image.

La formule, qui est répétée 10 fois pour chaque pixel de l'image, est la suivante :

$$\forall P \in [i * (255/10), (i + 1) * (255/10)] \quad f(P) = (i + 1) * (255/10)$$

Avec i : index de 0 à 9

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 – Image 1 et 6 après application du filtre grayscale

4.4 Réduction de Bruit (Filtre Median)

Nous avons réduit le bruit de l'image grâce au filtre médian. Ce filtre permet d'éliminer les contours et les imperfections des formes présente sur l'image. Le protocole est le suivant pour chaque pixel de l'image :

- Créer une matrice 3x3 avec les pixels adjacents -
- Créer une liste triée à partir des valeurs de cette matrice -
- Définir la nouvelle valeur du pixel comme la médiane de la liste -

Exemple :

$$\begin{pmatrix} 90 & 49 & 45 \\ 90 & 68 & 68 \\ 84 & 78 & 58 \end{pmatrix}$$



$$[45, 49, 58, 68, 68, 78, 84, 90, 90]$$



$$f(P) = 68$$

Le majeur problème de ce programme est que les pixels situés sur les extrémités ne possèdent pas exactement 8 pixels adjacents, la matrice n'est donc pas complète.

Pour palier à cela, nous avons décidé de ne pas prendre en compte les pixels aux extrémités supposant que la marge d'erreur sur l'image finale sera minime.

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	

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 7 – Image 1 et 6 après application du filtre médian

4.5 Binarisation (Filtre à seuil Adaptatif)

Pour obtenir une image traitable, il faut que les chiffres et la grille du sudoku soit facilement repérable. Pour ce faire, il faudrait dans l'idéal que les parties intéressantes de l'image soient blanches et le reste noir. La binarisation de l'image intervient donc ici. Pour chaque pixel, on compare son intensité à une variable, si l'intensité est supérieure, on met la valeur du pixel à 255, sinon 0.

Cette variable a été préalablement calculée à l'aide de la méthode d'Otsu. Cette méthode va calculer la variance minimale entre les pixels de l'image supposés dans deux plans différents (premier plan/arrière-plan). Elle détermine ainsi la valeur de la variable avec la plus grande efficacité de binarisaion selon une image donnée.

La formule se présente ainsi :

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

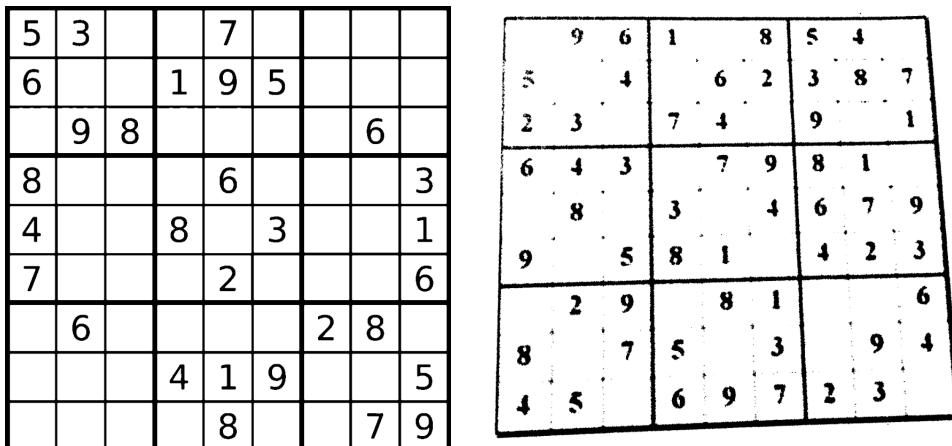


FIGURE 8 – Image 1 et 6 après application de la binarisation

4.6 Filtre d'inversion

Pour détecter les zones importantes, il est plus efficace de traiter ces zones quand elles sont blanches plutôt que noir. Comme on peut le remarquer ici, ce n'est pas le cas. On prend alors l'inverse de l'image grâce à cette formule :

$$f(P) = 255 - P$$

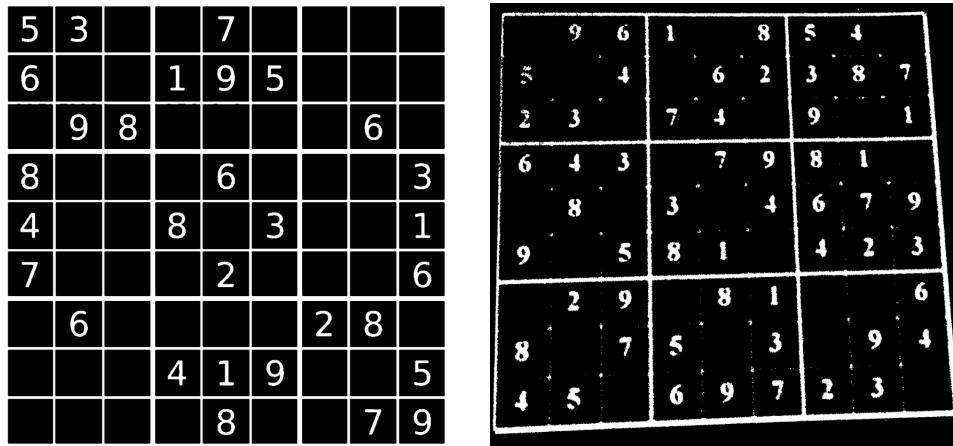


FIGURE 9 – Image 1 et 6 après application du filtre d'inversion

4.7 Filtre Sobel

Ce dernier filtre mis en place permet de détecter les contours des éléments de l'image. Il utilise une convolution entre des matrices et les axes x, y pour finalement calculer le gradient du pixel :

$$f(P) = \sqrt{{G_x}^2 + {G_y}^2}$$

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * P \quad \text{et} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * P$$

Bien sûr, les pixels avec une valeur assez élevée sont mis à 255, les autres à 0.

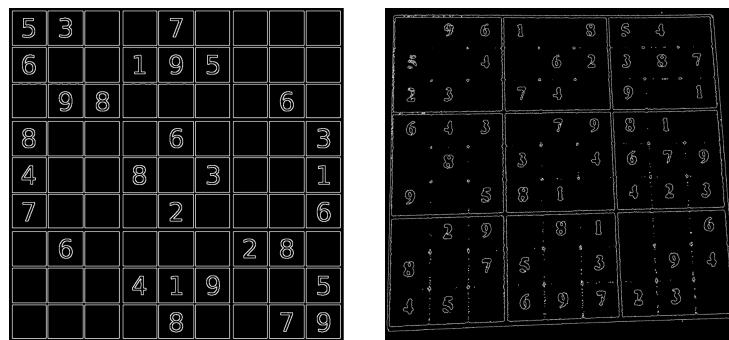


FIGURE 10 – Image 1 et 6 APRES application du filtre sobel

4.8 Resultat Final

Voici finalement l'image après le traitement en entier :

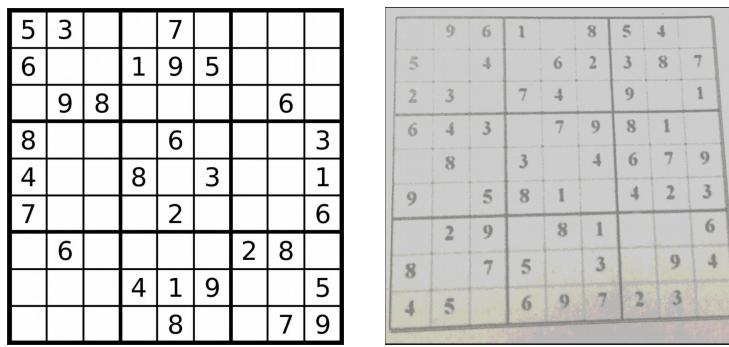


FIGURE 11 – Image 1 et 6 AVANT application de l'ensemble des prétraitements

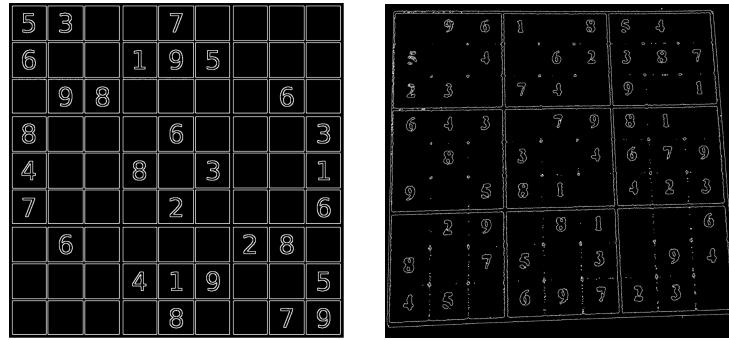


FIGURE 12 – Image 1 et 6 APRES application de l'ensemble des prétraitements

5 Ségmentation

Après avoir fini le traitement d'image, il ne reste plus qu'à récupérer chaque case de la grille pour les donner à l'intelligence artificielle.

La segmentation de l'image a été une partie assez spéciale de ce projet. En effet, les fonctions ou les formules utilisées pour la réaliser ne sont pas forcément complexes et facilement relisible. Néanmoins, c'est dans la logique et dans l'efficacité que cette partie a été le plus travaillé.

Nous avons ainsi dû la reprendre à plusieurs fois pour avoir une fonction qui s'exécute le plus rapidement possible.

5.1 Transformation d'Hough

Premièrement, nous avons implémenté l'algorithme de la transforme de Houg. Cet algorithme a pu être réalisé grâce aux différents prétraitements appliqués sur la grille au préalable. Ces traitements ont rendu la grille de sudoku facilement détectable par l'algorithme.

Ce dernier pourra alors nous donner sans difficulté les lignes du sudoku que nous pourrons traiter par la suite.

C'est la technique que nous avons utilisée pour permettre à notre programme de faire de la reconnaissance de lignes. C'est une étape indispensable pour résoudre un sudoku, car si nous n'avons pas les cases de ce sudoku, il devient impossible de le résoudre.

Donc le principe de la transformée de Hough est de passé d'une représentation scalaire a une représentation polaire. Cette méthode va permettre de pouvoir détecter des lignes même si elles sont incomplètes ou partiellement obstruée.

Pour que l'algorithme puisse être efficace il nous faut un moyen de garder seulement le contour des éléments sinon le nombres de lignes serait énorme et l'algorithme très lent pour des résultats difficilement exploitable. On a donc utilisé pour cela, le filtre de Sobel. Une fois que nous avons des bords bien défini. Nous parcourons l'image et à chaque pixel d'un bord (pixel blanc), on calcule « rho » pour chaque « theta » allant de 90 dégrée a -90 dégrée.

$$\text{Rho} = x * \cos(\theta) + y * \sin(\theta)$$

Après, nous allons incrémenter un accumulateur de « rho » « theta » de 1.

Nous avons donc un accumulateur qui contient le nombre de fois qu'un pixel a été parcouru en coordonnée polaire. Ce qui va nous permettre à l'aide d'un seuil définie de récupérer les coordonnées d'une ligne qui nous retransformons en coordonnée scalaire.

Nous obtenons donc un x et y de fin et de début qui va nous permettre de tracer des lignes sur notre images en fonctions des lignes de l'image. Ces lignes nous permettront ensuite de faire le découpage de cette même image.

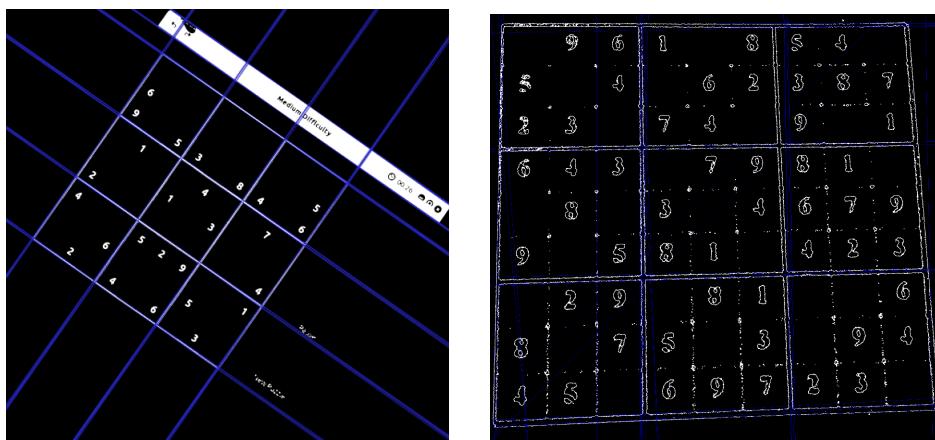


FIGURE 13 – Transformation de Houg

Pour cette soutenance, nous sommes repartis de 0 pour la transformée de Hough. Celle-ci ne marchait que trop peu et était long et mal optimisée. Nous avons tout de même gardé la même « base », la même idée pour l'algorithme mais nous avons changé notre implémentation, pour un gain d'efficacité et fiabilité.

Désormais, avec cette nouvelle méthode, nous avons même réussi à nous passer de l'algorithme de réduction de ligne introduit à la première soutenance, nous obtenons aussi de bien meilleurs résultats 3x plus rapidement que précédemment, et c'est comme ça que nous pouvons sereinement découper les images avec la plus grande fiabilité possible.

Hélas tout cela n'est pas arrivée comme par magie, c'est un effort de plusieurs semaines mais pourvoir détecter les lignes efficacement et avec une grande fiabilité était pour nous un point capital. C'est maintenant chose faites.

Voici l'accumulateur obtenu après l'exécution du programme :

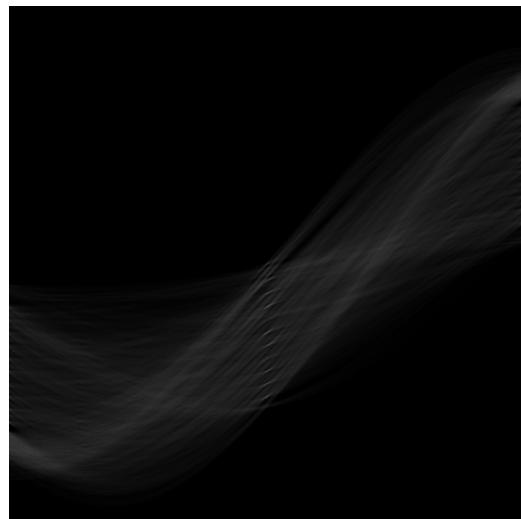


FIGURE 14 – Accumulateur en coordonnée polaire

5.2 Reconnaissance des Cases

Une fois le nombre de ligne réduit, il fallait réussir à détecter les cases du sudoku. Pour ce faire, nous avons implémenté une méthode simple dans son fonctionnement :

- Récupérer le carré avec l'aire la plus grande de la grille
- Le découper en 9×9 pour récupérer chaque case

5.2.1 Récupération des carrés

Dans un premier temps, la récupération des carrés. Cette méthode était la plus importante à optimiser dans la mesure où pour n lignes détectées, l'algorithme aura environ une complexité de n puissance 4. Il se décrit comme suivant :

- Pour chaque ligne de la grille, on rentre dans la fonction
- On réitère la fonction dans chaque intersection de cette ligne
- On répète ce processus 4x pour déterminer les 4 cotés du carré
- On vérifie finalement si le carré obtenu est fermé

- On le stock dans une liste

Voici comment la détection d'une intersection entre deux lignes dont on ne connaît seulement le point d'arriver et de départ fonctionne :

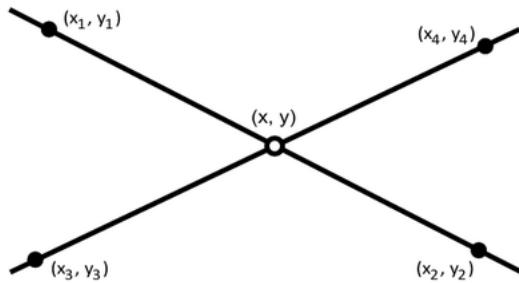


FIGURE 15 – intersection de deux droites

$$(x_1 - x_2) * (y_3 - y_4) - (y_1 - y_2) * (x_3 - x_4) \neq 0 \Rightarrow \exists \text{intersection}$$

On calcule ainsi le (x,y) de l'intersection des deux lignes.

De plus, puisque que ce ne sont pas des carrés parfaits, une marge d'erreur est prise en compte sur les valeurs des côté et des angles.

Voici l'ensemble des carré détectés :

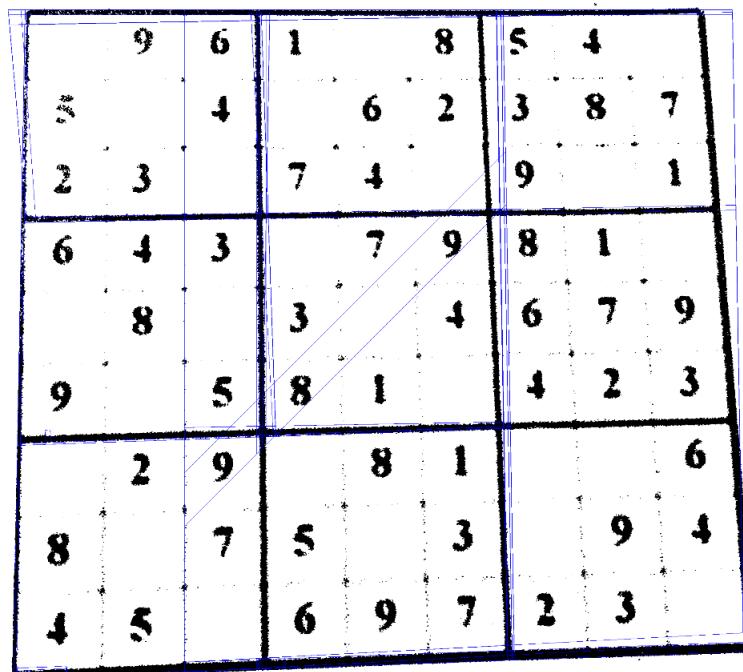


FIGURE 16 – tout les carrés détectés

5.2.2 Récupération des cotés de la grille

Finalement, après avoir obtenu la liste de tous les carrés de la grille, on récupère celui avec la plus grande aire.

Comme on peut le remarquer, l'algorithme de réduction de ligne joue ici une très haute importance puisque pour chaque ligne en moins, c'est des dizaines itérations du programme de détection qui n'ont pas besoin de s'exécuté.

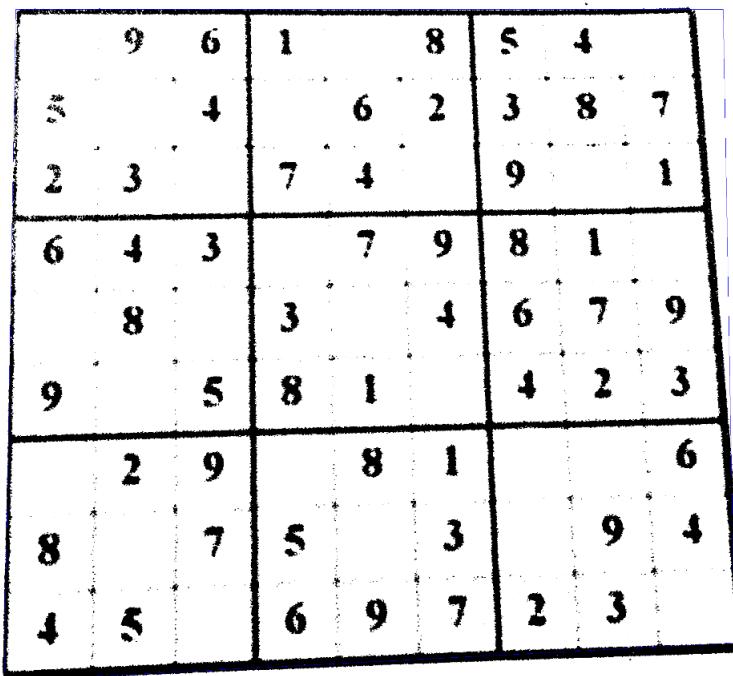


FIGURE 17 – tout les carrés détectés

5.3 Découpage et Chargement des Cases

Dans un second temps, nous devions découper la grille en 81 cases. Pour se faire, nous sommes partis du coin en haut à gauche du plus grand carré puis nous l'avons parcouru comme si nous parcourions une matrice mais en ajoutant à chaque fois les constantes x et y égales au ratio de la longueur des coté par 9.

On redimensionne aussi les cases pour s'assurer que le réseau de neurones pourra les traiter. Les images finales sont donc en format 28 x 28.

Grâce à cela, chaque case est extraite en image traitable par la suite par le réseau de neurones. Cette partie est aussi extrêmement utile dans la mesure où les cases ressorties peuvent aussi servir à entraîner le réseau de neurones.

Plus le réseau sera entraîné plus ses résultats selon fiable, il est donc indispensable d'avoir une base de données la plus complète et la plus diversifiée. Les cases ressorties peuvent alors servir à remplir cette base de données avec nombre de sudokus différent.

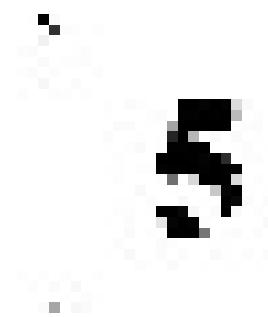


FIGURE 18 – Case contenant un 5

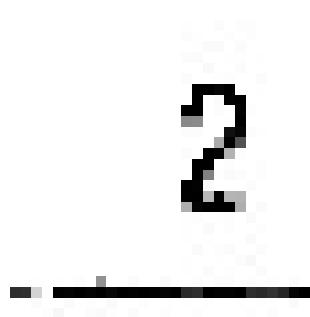


FIGURE 19 – Case contenant un 2

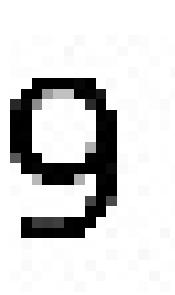


FIGURE 20 – Case contenant un 9

6 Réseau de neurones

Inspirés par le cerveau humain, les réseaux de neurones artificiels constituent un sous ensemble de l'apprentissage machine. Ils sont au cœur des algorithmes de deep learning. Pour mener à bien le projet, nous avons implémenter plusieurs réseaux de neurones, un qui puisse résoudre le XOR et un qui puisse reconnaître un chiffre depuis une image.

Les réseaux de neurones artificiels sont constitués de différentes couches de noeud (ou neurone artificiel), contenant une couche en entrée, une ou plusieurs couches cachées et une couche en sortie.

Chaque noeud, ou neurone artificiel, se connecte à un autre et possède un poids et un biais. Le but est donc de calculer la valeur des nœuds de sortie à partir des nœuds d'entrée à l'aide des poids et des biais. Ces poids et ces biais étant ajusté au fur et à mesure de l'entraînement du réseau.

Le principe est donc le même dans les deux cas que nous avons implémenté, seulement le nombre de neurones par couches, la manière de calculer les nœuds des couches, et les valeurs en entrée changent.

6.1 XOR

Le réseau du XOR a été le premier à être implémenté. Il nous a permis de comprendre plus facilement le fonctionnement général et de poser les bases de l'entraînement, avant de créer le réseau de neurones plus complexe qui permet de reconnaître des images de chiffres.

6.1.1 Structure

Le réseau du XOR comprend 2 neurones en entrée (0 ou 1 pour les deux), une couche cachée de 2 neurones, et une couche de 1 neurone en sortie (0 ou 1).

6.1.2 Forward Propagation

Pour déterminer une valeur de sortie en fonction des valeurs d'entrées, il fallait implémenter l'algorithme de Forward propagation. Cet algorithme permet de trouver la valeur des nœuds de la prochaine couche en fonction de la

couche précédente, des poids et des biais de la couche à calculer.

La Forward propagation pour le XOR détermine les valeurs sur la couche caché avec la couche d'entrée puis de la couche de sortie avec la couche cachée. Elle utilise les poids et les biais de chaque noeud.

Pour déterminer ces valeurs, l'algorithme utilise aussi une fonction d'activation. Dans le cas du XOR chaque noeud, de la couche cachée et de la couche de sortie, est activé avec la fonction sigmoïde :

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

6.1.3 Back Propagation

La Forward Propagation renvoie une valeur en sortie qui ne correspondent pas à la véritable valeur engendrée par les entrées. Pour le XOR, si nous avons en entrée 0 et 1, nous devrions avoir 0 en sortie.

La Back Propagation va donc se baser sur la valeur de sortie, qui sera aléatoire au lancement de l'entraînement, ainsi que de la valeur attendue, et va ajuster à l'aide d'un coefficient d'apprentissage, les poids et les biais de chaque couche à partir de la sortie jusqu'à l'entrée.

Dans le cas du XOR, la Back Propagation fait appel à la dérivée de la fonction sigmoïde.

6.2 Digit Recognition

Une fois que le XOR était implémenté et fonctionnel, il était plus simple d'implémenter la reconnaissance de chiffres.

6.2.1 Structure

Pour le cas de la reconnaissance de chiffre, le nombre de noeuds dans la couche d'entrée est de 784 ce qui correspond au nombre de pixels d'une image standardisé (28×28 pixels). Pour les couches intérieures, nous avons fait le choix de placer 2 couches cachées ce qui permet d'obtenir plus de précisions. La première couche cachée possède 128 neurones, la deuxième en possède 32.

Pour finir, la couche de sortie possède 10 neurones, représentant les 10 chiffres.

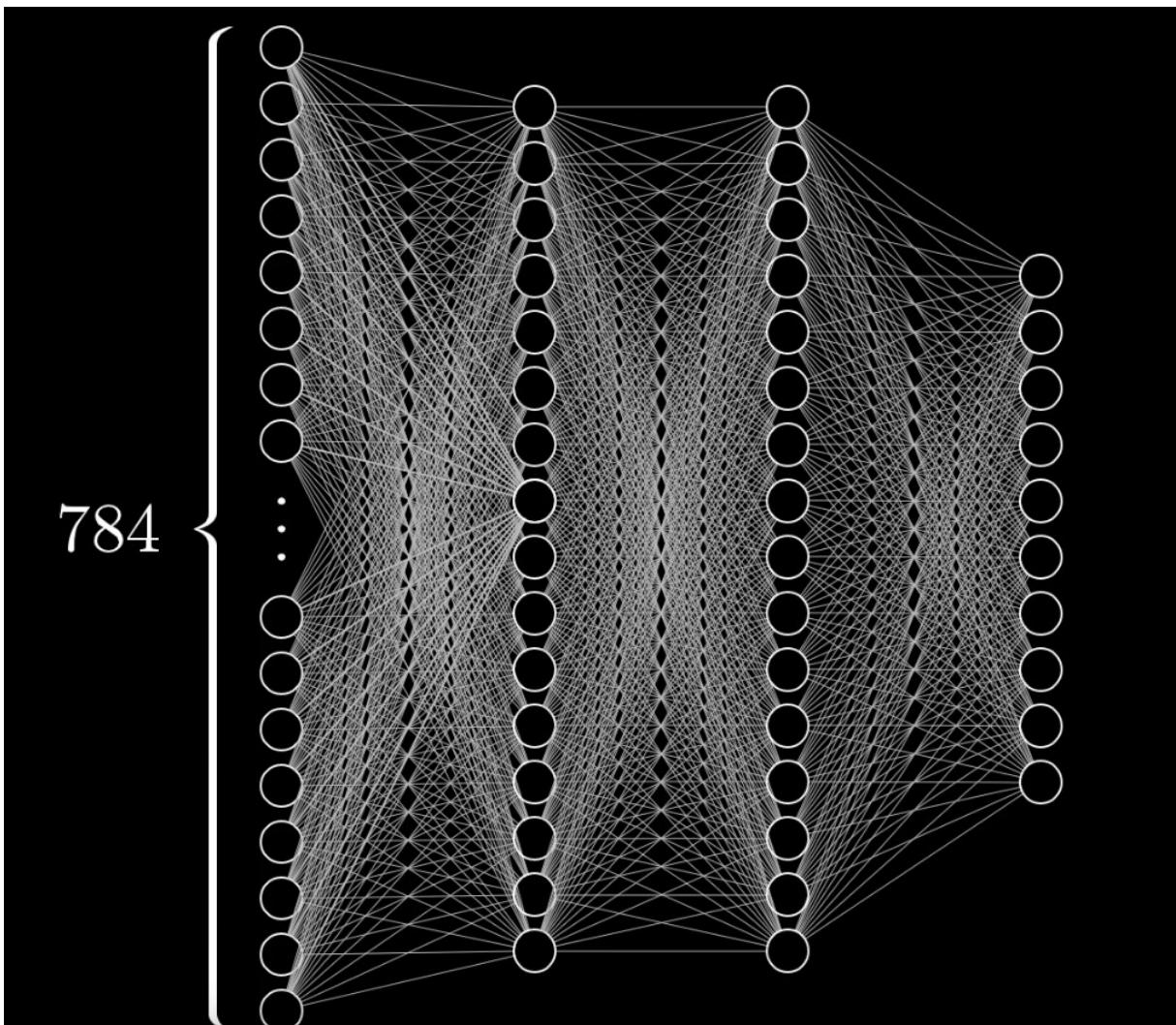


FIGURE 21 – Structure du réseau

6.2.2 Forward Propagation

La Forward propagation pour le réseau de digit recognition est légèrement différente de celle du XOR puisque cette fois la couche de sortie comporte 10 neurones. Pour déterminer ses valeurs, l'algorithme utilise deux fonctions d'activation. Chaque nœud des couches cachées est activé avec la fonction sigmoïde, tout comme le XOR.

Mais cette fois les neurones de la couche de sortie sont activés avec une autre fonction d'activation : softmax

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Cette fonction permet de sortir un tableau de probabilité en sortie. Dès lors, la valeur max de ce tableau correspond au nombre le plus probable d'être sur l'image d'entrée.

6.2.3 Back Propagation

Pour ajuster les poids et les biais de chaque couche, la Back Propagation dans le cas de la reconnaissance de chiffre suit le même principe que celle du XOR, seulement elle ajuste la couche de sortie avec la dérivée de softmax, et les couches cachées avec la dérivée de sigmoïde.

6.3 Jeu de données pour l'entraînement

Dans le cas du XOR, le jeu de données correspond à la table d'entrée d'un XOR, ce qui fait 4 valeur possible pour la couche d'entrée (0 0, 0 1, 1 0, 1 1).

Dans le cas de la reconnaissance de chiffres, la création d'un jeu de données à fait l'objet d'un travail plus important. A l'aide d'un script python et d'un jeu de polices non manuscrites, nous avons créer un jeu de données de 5000 images de 28*28 pixels pour entraîner le réseau. Chaque image étant un chiffre aléatoire entre 0 et 9 écrits dans une police aléatoire.

Pour le cas du 0, un sudoku ne possédant pas le chiffre 0, il fallait que le réseau reconnaisse les cases vides. Donc au moment de la création d'une image de 0, le script python crée une case vide.

Pour tester le réseau de façon correcte, nous avons créer un deuxième jeu de données différent de celui pour entraîner. Les images sont créées avec des polices différentes de celles du jeu d'entraînement.

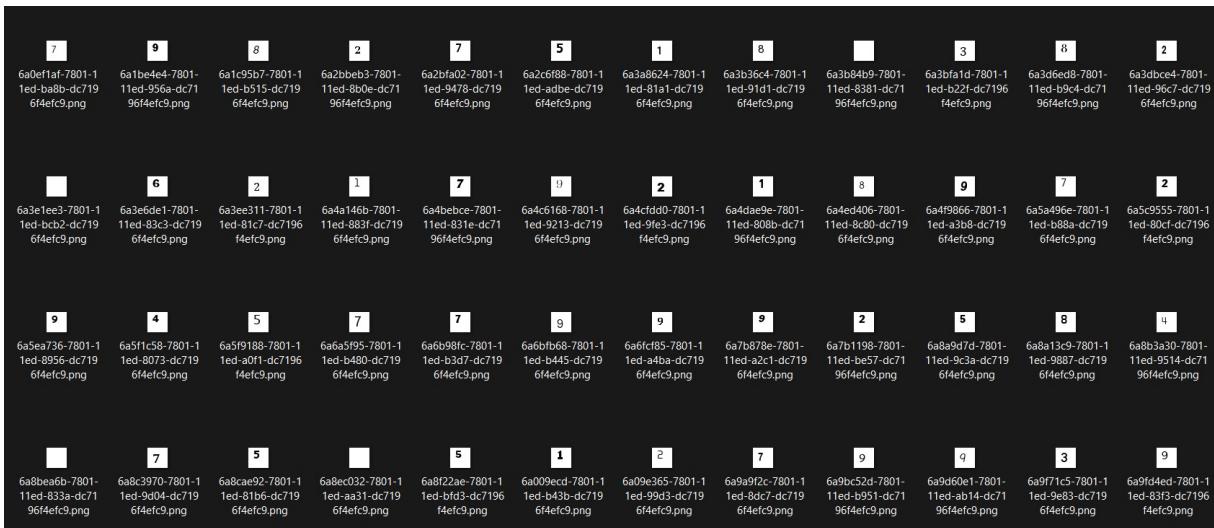


FIGURE 22 – Jeu de données

6.4 Entrainement du réseau

Pour entraîner les différents réseaux il faut enchaîner plusieurs fois la Forward Propagation et la Back Propagation avec l'ensemble du jeu de données mélanger de façon aléatoire.

Pour la première étape, les poids et les biais sont générés aléatoirement.

Ensuite pour les étapes d'après les poids et les biais sont ajuster pour s'approcher de la véritable valeur de sortie. Plus le réseau est entraîné, plus le résultat en sortie est précis.

Dans le cas de la reconnaissance des chiffres, le jeu de données était des images, il fallait donc faire un prétraitement sur les images afin de les transformer en tableau de pixels de 784. Heureusement, des fonctions déjà implémenter pour le prétraitement du sudoku ont permis de le faire facilement.

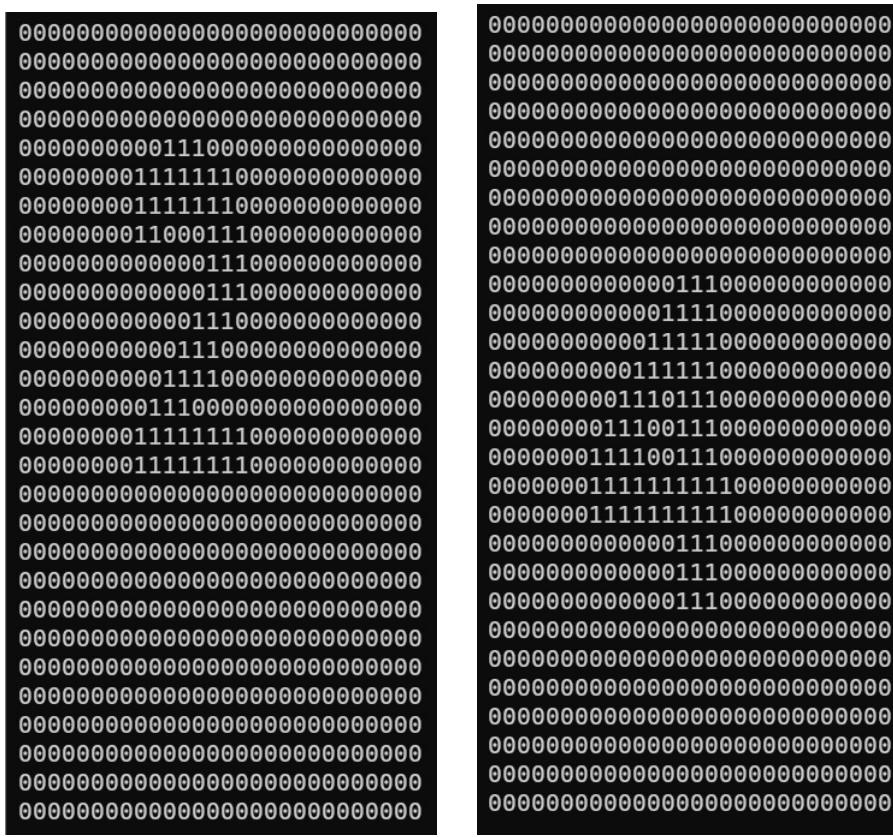


FIGURE 23 – APRES le prétraitement d'une image 2 et d'une image 4

6.5 Sauvegarde des poids et des biais

Une fois que le modèle a été entrainé, les poids et les biais de chaque neurone permettent de trouver un résultat précis. Il faut donc sauvegarder ces poids et ces biais pour que la machine soit capable de trouver le bon résultat en fonction des entrées.

Dans notre cas, après avoir entraîné le réseau, un dossier dans lequel un fichier par couche sauvegarde les poids et les biais de la couche.

6.6 Résultats et difficultés de la reconnaissance de chiffre

L'entraînement a été l'une des plus grandes difficultés sur le projet. Nous avons réalisé beaucoup d'entraînements différents. Comme dit précédemment, pour entraîner et tester de façon correcte notre réseaux, nous avons créé 2 dataset différents, un pour entraîner et un pour tester (avec des images différentes).

Nous avons d'abord entraîné avec 5000 images sur plus de 10 000 epochs (c'est-à-dire 10 000 fois forward et backward propagation) et tester avec 3000 images différentes. Le tout tester plusieurs fois avec plusieurs coefficients d'apprentissage. Puis avec 3000 tester avec 5000.

Ensuite avec 2500 images sur 2000 epochs et 2000 images de test.

Nous avons aussi essayé de faire diminuer le coefficient d'apprentissage au fur et à mesure de l'avancée des epochs, pour faire en sorte de préciser l'apprentissage.

Et pour finir nous avons essayé d'entraîner 9000 images sur 1000 epochs.

Beaucoup d'essais qui prennent du temps.

Finalement, le réseau ne présente malheureusement pas plus de 80% de résultats corrects. Cela reste un bon résultat mais pas encore assez bon pour atteindre l'état de l'art.

```

image 4970/5000 : 1 -- OK
image 4971/5000 : 2 -- OK
image 4972/5000 : 1 -- OK
image 4973/5000 : 3 (expected 7)
image 4974/5000 : 9 (expected 4)
image 4975/5000 : 2 -- OK
image 4976/5000 : 2 -- OK
image 4977/5000 : 4 -- OK
image 4978/5000 : 5 -- OK
image 4979/5000 : 5 -- OK
image 4980/5000 : 5 -- OK
image 4981/5000 : 8 -- OK
image 4982/5000 : 2 -- OK
image 4983/5000 : 2 (expected 9)
image 4984/5000 : 4 -- OK
image 4985/5000 : 7 -- OK
image 4986/5000 : 7 -- OK
image 4987/5000 : 4 -- OK
image 4988/5000 : 8 -- OK
image 4989/5000 : 9 -- OK
image 4990/5000 : 0 -- OK
image 4991/5000 : 3 -- OK
image 4992/5000 : 3 (expected 2)
image 4993/5000 : 6 -- OK
image 4994/5000 : 8 (expected 6)
image 4995/5000 : 8 -- OK
image 4996/5000 : 9 -- OK
image 4997/5000 : 5 -- OK
image 4998/5000 : 4 -- OK
image 4999/5000 : 1 -- OK
image 5000/5000 : 1 (expected 2)
winrate : 78.084383

image 2967/3000 : 2 -- OK
image 2968/3000 : 2 (expected 1)
image 2969/3000 : 1 -- OK
image 2970/3000 : 6 -- OK
image 2971/3000 : 7 -- OK
image 2972/3000 : 0 -- OK
image 2973/3000 : 4 -- OK
image 2974/3000 : 3 -- OK
image 2975/3000 : 9 -- OK
image 2976/3000 : 0 -- OK
image 2977/3000 : 3 -- OK
image 2978/3000 : 2 (expected 7)
image 2979/3000 : 4 -- OK
image 2980/3000 : 4 (expected 5)
image 2981/3000 : 4 -- OK
image 2982/3000 : 6 -- OK
image 2983/3000 : 6 -- OK
image 2984/3000 : 8 (expected 6)
image 2985/3000 : 6 (expected 1)
image 2986/3000 : 9 -- OK
image 2987/3000 : 3 (expected 1)
image 2988/3000 : 4 -- OK
image 2989/3000 : 6 -- OK
image 2990/3000 : 0 -- OK
image 2991/3000 : 5 -- OK
image 2992/3000 : 5 -- OK
image 2993/3000 : 5 -- OK
image 2994/3000 : 2 -- OK
image 2995/3000 : 2 (expected 1)
image 2996/3000 : 0 -- OK
image 2997/3000 : 0 -- OK
image 2998/3000 : 4 (expected 9)
image 2999/3000 : 6 (expected 8)
image 3000/3000 : 5 -- OK
winrate : 78.273909

```

FIGURE 24 – Résultats de deux différents entraînements

7 Fonctionnement global d'OCR

Après de nombreuses recherches, voici la méthode que nous avons utilisé et qui nous parrait la plus efficace :

1. Prétraitement
 - (a) Filtre Grayscale
 - (b) Filtre des Contrastes
 - (c) Filtre Médian
 - (d) Filtre à seuil Adaptatif
 - (e) Filtre d'inversion
 - (f) Filtre Sobel

2. Segmentation de la grille
 - (a) Transformation de Houg

- (b) Rotation automatique
 - (c) Détection de tout les carrés
 - (d) Détection des bords de la grille
 - (e) Découpage des 81 cases
3. Reconnaissance des chiffres
 - (a)
 4. Résolution du sudoku
 - (a) Algorithme de back tracking
 5. Reconstruction du sudoku
 - (a) Affichage du sudoku complété

8 Interface

8.1 Interface globale

Pour l'interface nous avons utilisé GTK3, une librairie graphique très utilisé en C et en python et plutôt simple d'utilisation pour les possibilités d'interface qu'elle offre. Pour faire l'interface on peut placer chaque élément à l'aide de lignes de codes mais cela est très fastidieux et long.

Nous avons donc utilisé Glade qui permet de placer tous les éléments et de leur paramétriser quelques options (le texte qui apparaît dessus, s'ils sont cliquables ou pas...) facilement et avec une possibilité de voir ce que sera le rendu final. Une fois les éléments placé Glade nous renvoie un fichier XML (voir Figure 14) qui permet de faire le lien avec GTK pour qu'il sache où est chaque élément et comment ils sont paramétrés.

Comme on peut le voir sur la figure 13, l'interface nous propose plusieurs options. Celle de faire les étapes du processus les uns après les autres, celle qui permet de toutes les passer et d'avoir juste le résultat final. Il y a aussi un bouton “Reset” qui permet de revenir sur l'image de base.

Une barre de chargement permet de voir où nous en sommes dans le processus et un texte en dessous nous indique quel processus nous appliquons sur l'image.

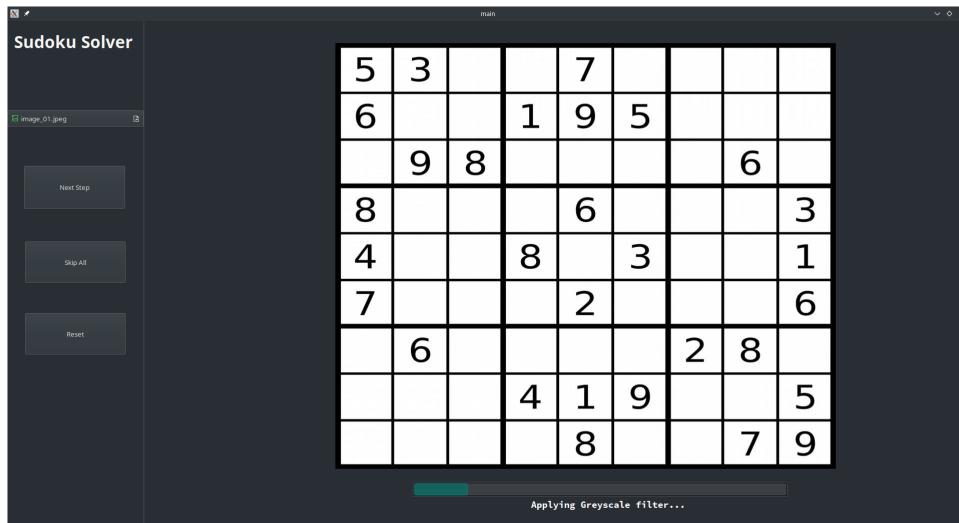


FIGURE 25 – Exemple de l’interface avec l’image 1

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated with glade 3.40.8 -->
<object class="GtkWindow">
  <property name="title" translatable="yes">Sudoku Solver</property>
  <property name="visible" type="boolean">true</property>
  <property name="can-focus" type="boolean">true</property>
  <property name="default-height" type="int">1000</property>
  <child>
    <object class="GtkPaned">
      <property name="visible" type="boolean">true</property>
      <property name="can-focus" type="boolean">true</property>
      <child>
        <object class="GtkImage">
          <property name="visible" type="boolean">true</property>
          <property name="can-focus" type="boolean">false</property>
          <child>
            <object class="GtkButton" id="NextButton">
              <property name="label" translatable="yes"><input type="button" value="Next Step Process"/></property>
              <property name="width-request" type="int">100</property>
              <property name="height-request" type="int">30</property>
              <property name="receives-default" type="boolean">true</property>
              <property name="tooltip-text" translatable="yes">>To go to next step of the process</property>
            </object>
          </child>
        </object>
      </child>
      <object class="GtkFileChooserButton" id="file_chooser">
        <property name="width-request" type="int">222</property>
        <property name="height-request" type="int">30</property>
        <property name="visible" type="boolean">true</property>
        <property name="can-focus" type="boolean">false</property>
        <property name="title" translatable="yes">>Select a Sudoku Image please</property>
        <child>
          <packing>
            <property name="y">178</property>
          </packing>
        </child>
      </object>
      <object class="GtkLabel">
        <property name="width-request" type="int">254</property>
        <property name="height-request" type="int">72</property>
        <property name="visible" type="boolean">true</property>
        <property name="can-focus" type="boolean">false</property>
        <property name="text" translatable="yes">>Sudoku Solver</property>
      </object>
    </child>
  </object>

```

FIGURE 26 – Début du fichier XML généré par Glade

9 Ressentis et Organisation

9.1 Resumé des Avancements

Pour résumer l'avancé que nous put atteindre en ces 2 mois de travail sur ocr, nous pensons avoir remplis les demandes concernant la première soutenance. Nous avons fini le Solver, le prétraitement et la détection des cases de la grille. Nous avons aussi créé le XOR qui nous permet d'avoir les bases de l'intelligence artificielle. Néanmoins, certaines optimisations vont sûrement être nécessaires pour pouvoir atteindre le meilleur taux de réussite possible.

9.2 Dificultés

Les difficultés rencontrées jusqu'à présent sur ce projet ont été principalement liées à deux sujets.

La première, bien entendu, étant le code. Nous sommes confrontés à un tout nouveau langage de programmation qui nous était, pour la majeure partie, inconnu. La difficulté était ici principalement à cause de l'introduction des pointers qui sont réelle découverte pour nous.

Ensuite, la deuxième difficulté a été l'organisation générale, étant intrinsèquement quelque chose que tout le monde redoute. Il nous a fallu faire des efforts pour arriver à un espace de travail convenant à tout le monde.

9.3 Points Positifs

Malgré toutes les difficultés citées précédemment, nous trouvons que ce début de projet c'est très bien passé en général. Les difficultés rencontrées sur le code nous ont permis de progresser significativement. Et même si nous n'avions pas l'habitude de travailler ensemble chacun a réussi à s'adapter au mieux pour réaliser ce projet.

10 Conclusion

Pour conclure, le bilan de ce projet est plutot positif, chaque membre de l'équipe à pu apporter ses compétences pour avancer et mettre en oeuvre les étapes clés. Le planning et les différentes parties a réaliser ont été respecter pour cette deuxième soutenance. Bien que nous ayons rencontré quelques difficultés, le projet à bien avancé et nous sommes fier du travail accompli.

Table des figures

1	Alexis Cognet	3
2	Alexandre Posta	4
3	Flavien Geoffray	4
4	Quentin Gillet	5
5	Image 1 et 6 après application du filtre grayscale	8
6	Image 1 et 6 après application du filtre grayscale	9
7	Image 1 et 6 après application du filtre médian	10
8	Image 1 et 6 après application de la binarisation	11
9	Image 1 et 6 après application du filtre d'inversion	12
10	Image 1 et 6 APRES application du filtre sobel	13
11	Image 1 et 6 AVANT application de l'ensemble des prétraitements	13
12	Image 1 et 6 APRES application de l'ensemble des prétraitements	13
13	Transformation de Houg	15
14	Accumulateur en coordonnée polaire	16
15	intersection de deux droites	17
16	tout les carrés détectés	18
17	tout les carrés détectés	19
18	Case contenant un 5	20
19	Case contenant un 2	20
20	Case contenant un 9	20
21	Structure du réseau	23
22	Jeu de données	25
23	APRES le prétraitement d'une image 2 et d'une image 4 . . .	26
24	Résultats de deux différents entraînements	28
25	Exemple de l'interface avec l'image 1	31
26	Début du fichier XML généré par Glade	31