
Rapport de soutenance

Reconnaissance Optique des Caractères (ROC) Optical Character Recognition (OCR)

Alexandra Wolter - Erwann Lesech - Guillaume Jolivalt - Valentin Gibbe

EPITA Lyon | Promo 2026



1	Introduction	2
1.1	Qu'est ce qu'un OCR ?	2
1.2	Présentation du projet	2
1.3	Présentation du groupe de travail	3
2	La répartition des tâches	4
2.1	Tableau de répartition	4
3	Etat d'avancement du projet	5
3.1	Prévisions cahier des charges	5
3.1.1	Rappel des estimations dans l'avancement des tâches	5
3.1.2	Avancement actuel	6
4	Aspects techniques	7
4.1	Traitement d'image	7
4.1.1	Chargement d'une image et suppression des couleurs	7
4.1.2	Rotation manuelle de l'image	7
4.1.3	Détection de la grille et de la position des cases	8
4.2	Découpage de l'image	10
4.3	Sauvegarde de la découpe	10
4.4	Résolution du sudoku	11
4.5	Réseau de neurones	11
4.5.1	Fonctionnement d'un réseau de neurones	11
4.5.2	Preuve de concept (porte ou-exclusive)	13
4.5.3	Sauvegarde et chargement des poids	14
4.5.4	Base d'images pour l'entraînement du réseau	14
4.5.5	Mise en place du réseau principal de neurones	15
4.6	Ressenti du projet	16
4.6.1	Erwann Lesech	16
4.6.2	Guillaume Jolivald	16
4.6.3	Alexandra Wolter	16
4.6.4	Valentin Gibbe	16
4.6.5	Avis général	16
5	Conclusion	17
	Acronyms	19

1.1 Qu'est ce qu'un OCR ?

Un Optical Character Recognition (OCR) ou Reconnaissance Optique des Caractères (ROC) correspond à la reconnaissance automatisée de textes imprimés ainsi qu'à la retranscription de ces textes écrits en fichiers numériques.

De fait, un OCR permet à un appareil numérique de "lire" le contenu d'un document papier et de le restituer informatiquement.

1.2 Présentation du projet

Dans notre cas, l'objectif de notre OCR est de scanner un sudoku imprimé, de numériser la grille ainsi que les chiffres déjà inscrits. Par la suite, notre logiciel devra être capable de résoudre ce sudoku pré-rempli pour enfin le restituer dans sa forme complétée via une interface utilisateur ergonomique et intuitive.

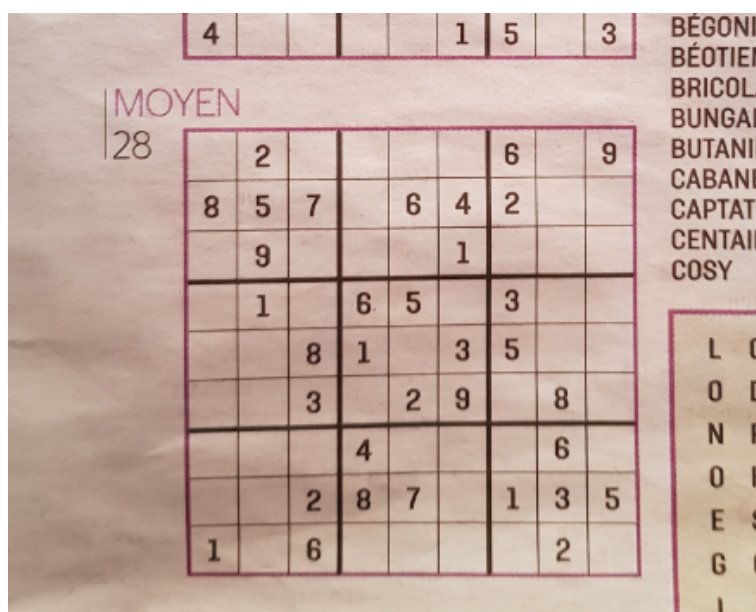


Figure 1.1: Sudoku imprimé

1.3 Présentation du groupe de travail

Erwann "R-One" LESECH (Chef de projet)

Roux et motard du groupe, Je suis fort intéressé par l'intelligence artificielle et le machine/deep learning, c'est pour quoi ce projet est pour moi la première véritable opportunité de me familiariser avec les réseaux de neurones.

Ce projet me permettra d'améliorer mes compétences de leadership afin d'assurer que chaque partie du projet progresse à la même allure afin de proposer un produit fini et de qualité dans les temps impartis.

Guillaume "Guimauve" JOLIVALT

Curieux du groupe, je m'intéresse à beaucoup de domaines de la programmation et de l'informatique depuis bien longtemps. C'est pour cela que j'ai pour but d'aider et de découvrir différentes parties.

Ce projet va me permettre de découvrir et d'explorer de nouveaux univers plus captivants les uns que les autres.

Alexandra "Alex" WOLTER

Joueuse de Wow et seule fille du groupe, j'aime apprendre de nouvelles choses et affronter des difficultés. La création d'interface est une grande première pour moi et j'espère en apprendre beaucoup.

Ce projet est un nouveau challenge à relever pour moi et j'ai hâte d'y parvenir à l'aide de mes camarades !

Valentin "Nimu" GIBBE

Alcoolique du groupe, je suis globalement motivé pour tous types de projets informatique. Le projet OCR m'est assez déroutant étant donné que je n'ai jamais fait de traitement d'image auparavant.

Ce projet est une découverte pour ma part et va me permettre de sortir de ma zone de confort en confrontant des problèmes auxquels je n'ai jamais eu à faire.

CHAPTER 2

LA RÉPARTITION DES TÂCHES

2.1 Tableau de répartition

Tâches \ Membres	Alexandra	Erwann	Guillaume	Valentin
Traitement d'image				
Chargement d'une image	S			R
Suppression des couleurs	S			R
Rotation manuelle de l'image	R			S
Detection de la grille	S			R
Decoupage de l'image	R		S	
Sauvegarde de la découpe	S		R	
Résolution du sudoku				
Programme de résolution		S	R	
Sauvegarde et présentation des résultats		S	R	
Réseau de neurones				
Recherches des différents algorithmes		R	S	
Réseau de neurones pour XOR		R	S	
Sauvegarde et chargement des poids		R	S	
Jeu d'image et importation		R	S	
Réseau de neurones complet		R	S	
UI / Présentation				
Reconstruction de la grille	R		S	
Affichage de la grille	R		S	
Sauvegarde du résultat	R	S		
Interface graphique	R	S		

Table 2.1: Tableau de la répartition des tâches, R = Responsable & S = Suppléant

CHAPTER 3

ETAT D'AVANCEMENT DU PROJET

3.1 Prévisions cahier des charges

3.1.1 Rappel des estimations dans l'avancement des tâches

Voici le tableau rappelant les estimations dans l'avancement des différentes tâches

Tâches	Soutenances	soutenance 1	soutenance 2
Traitement d'image			
Chargement d'une image		100%	100%
Suppression des couleurs		50%	100%
Rotation manuelle de l'image		100%	100%
Detection de la grille		100%	100%
Découpe de l'image		75%	100%
Sauvegarde de la découpe		100%	100%
Résolution du sudoku			
Programme de résolution		100%	100%
Sauvegarde et présentation des résultats		100%	100%
Réseau de neurones			
Recherches des différents algorithmes		80%	100%
Réseau de neurones pour XOR		100%	100%
Sauvegarde et chargement des poids		50%	100%
Jeu d'image et importation		40%	100%
Réseau de neurones complet		10%	100%
UI / Présentation			
Reconstruction de la grille		0%	100%
Affichage de la grille		0%	100%
Sauvegarde du résultat		0%	100%
Interface graphique		0%	100%

Table 3.1: Estimation des avancées des différentes tâches

3.1.2 Avancement actuel

Voici le tableau récapitulatif du respect de notre avancement :

Tâches	Prévisions	Fait
Traitement d'image		
Chargement d'une image	100%	100%
Suppression des couleurs	50%	50%
Rotation manuelle de l'image	100%	100%
Détection de la grille	100%	80%
Découpage de l'image	75%	75%
Sauvegarde de la découpe	100%	100%
Résolution du sudoku		
Programme de résolution	100%	100%
Sauvegarde et affichage des résultats	100%	100%
Réseau de neurones		
Recherches des différents algorithmes	80%	80%
Réseau de neurones pour XOR	100%	100%
Sauvegarde et chargement des poids	50%	80%
Jeu d'image et importation	40%	40%
Réseau de neurones complet	0%	10%
UI / Présentation		
Reconstruction de la grille	0%	0%
Affichage de la grille	0%	0%
Sauvegarde du résultat	0%	0%
Interface graphique	0%	5%

Table 3.2: Respect des prévisions de l'avancement du projet

Pour résumé, en l'état actuel nous avons implémenté :

- Un repérage des grilles qu'elles soient tournées ou non.
- Une rotation manuelle des images si nécessaire.
- Une numérisation des sudoku imprimés ainsi qu'une détection de grille permettant la division par 81 images de toutes les cases du sudoku présenté.
- Une résolution de sudoku fonctionnelle qui récupère une grille dans un fichier, la résout et sauvegarde les résultats dans un autre fichier.
- Un réseau de neurones apprenant la Porte **ou-exclusive** (XOR) ainsi que la sauvegarde des poids et biais du réseau, son chargement à partir d'un fichier texte et enfin une database partielle d'image pour entraîner le réseau principal de neurones.

4.1 Traitement d'image

4.1.1 Chargement d'une image et suppression des couleurs

Cette partie fut au final plutôt facile étant donné que nous l'avons traité en TP (le TP 06). Pour charger l'image, nous utilisons la librairie SDL2 Image qui permet de charger tous les formats (JPEG, PNG etc).

Nous convertissons ensuite l'image au format RGB888. Une fois cela fait, nous mettons l'image en gris en calculant la nuance de gris de chaque pixel à l'aide de cette fonction:

$$average = 0.3 * red + 0.59 * green + 0.11 * blue;$$

Et nous mettons ensuite à chaque composante (bleu,vert,rouge) la valeur "average".

4.1.2 Rotation manuelle de l'image

La rotation manuelle se déroule en plusieurs grandes étapes. La première étape est de créer les différentes surfaces nécessaires à la rotation de l'image à partir du fichier donné par l'utilisateur.

Dans la seconde étape, nous utilisons l'angle en degrés donné par l'utilisateur et nous le convertissons en radian à l'aide de la formule :

$$angle_r = \text{degrés} * \pi / 180$$

Ensuite, nous opérons la rotation sur chaque pixel de la surface à l'aide des formules suivantes

$$X = x * \cos(angle_r) - y * \sin(angle_r)$$

$$Y = x * \sin(angle_r) + y * \cos(angle_r)$$

Pour finir, la quatrième et dernière étape consiste à vérifier si le pixel tourné fait bien parti de la surface, ainsi nous opérons la rotation, sinon il n'y a aucun changement effectué.

4.1.3 Détection de la grille et de la position des cases

Pour détecter la grille ainsi que la position des cases, nous avons dû reconnaître les contours de l'image afin de pouvoir détecter la grille par la suite. Voici un exemple sur le premier Sudoku:

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

Exemple du filtre de Sobel

Pour ce faire, nous avons utilisé le filtre de Sobel qui se base sur deux matrices de Convo-lution:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{A} \quad \text{et} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{A}$$

Matrice de Convolution

Nous calculons par la suite les gradients horizontaux (pour chaque pixel, nous prenons tous ses pixels voisins puis nous les multiplions par la matrice \mathbf{G}_x), et les gradients verticaux (même procédé mais avec la matrice \mathbf{G}_y). Nous calculons ensuite la norme du gradient. Si celle-ci est plus grande que 128 nous mettons un pixel blanc sinon un pixel noir.

Une fois que nous avons détecté les contours, il faut repérer les lignes du sudoku pour pou-voir faire le découpage. C'est sûrement l'une des parties les plus complexes du chargement de l'image car il faut réussir à repérer les lignes sans repérer les caractères.

Nous allons pour ce faire utiliser la Transformée de Hough. Voici ce que donne cet algorithme après l'implémentation sur le premier Sudoku:

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

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

Exemple du transformé de Hough

Comme nous pouvons le voir, les lignes violettes représentent ce que l'algorithme a réussi à "trouver" comme lignes. Nous pouvons voir qu'il arrive à repérer les lignes sans les caractères. Le transformée de Hough fonctionne de la façon suivante:

- Nous commençons par initialiser une matrice que nous nommons matrice d'accumulation. Nous allons aussi initialiser toutes les valeurs cosinus et sinus de 0 à 180 degrés (pour ne pas avoir à refaire le calcul à chaque fois).

- Une fois cela fait, si nous rencontrons un pixel blanc, nous allons calculer toutes ses droites polaires correspondantes en faisant varier une variable Θ de 0 à 180 degrés. L'équation de la droite polaire est la suivante:

$$r = (x - (largeur/2)) * \cos(\Theta) + (y - (longueur/2)) * \sin(\Theta)$$

Nous incrémentons ainsi de 1 la matrice d'accumulation: $matrice_accumulation[r][theta] + 1$. Avant cela nous ajoutons la diagonale de l'image à r pour éviter les valeurs négatives. En même temps que cela, nous cherchons la valeur maximum de cette matrice.

- Par la suite, nous allons reparcourir notre matrice d'accumulation afin de retenir les "pics" de cette dernière. La première chose à faire est de ne retenir que les valeurs de la matrice supérieures au $max * 0.42$. Ensuite, nous allons chercher le pic local maximum sur 9*9 pixels. Uniquement celui-ci sera retenu.

- Enfin, il ne nous reste plus qu'à reconvertir les rho et theta que nous avons retenus en coordonnées cartésiennes et de tracer les lignes.

4.2 Découpage de l'image

Pour le découpage de l'image, nous utilisons une image prétraitée afin qu'il y ait le moins d'espace possible autour de la grille. Nous découpons en surfaces égales la surface, de façon à avoir une surface par case de la grille, pour ce faire nous utilisons les pixels de la surface.

Nous procédons de la même manière pour toutes les cases de la grille, tant que le pixel appartient à la case, c'est à dire tant que sa position est inférieure en largeur et en longueur de la case alors :

- si le pixel appartient à l'image : nous utilisons le pixel correspondant sur la surface de base
- si le pixel n'appartient pas à l'image, : celui-ci devient un pixel noir

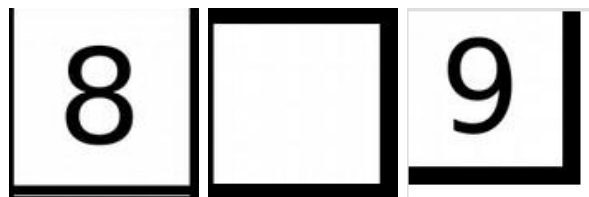


Figure 4.1: Exemples de découpes de l'image

4.3 Sauvegarde de la découpe

Lors de la découpe du sudoku, nous avons sauvegardé une nouvelle image pour chaque case découpée. Chaque image a pour nom "*son_indexage.bmp*". L'indexage étant compris entre 0 et 80 et commence en haut à gauche pour finir en bas à droite. Ces images seront récupérées et utilisées ultérieurement par l'intelligence artificielle pour détecter les caractères.

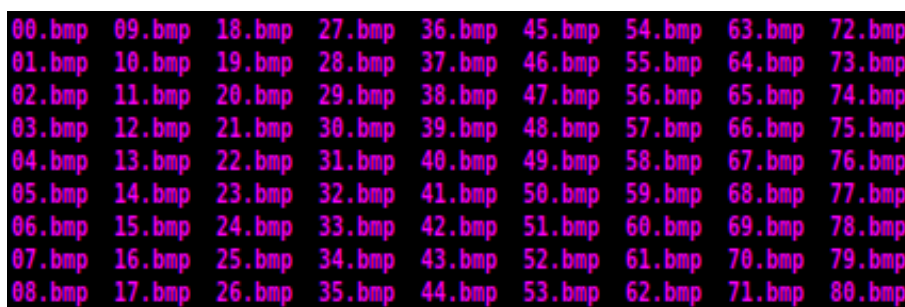
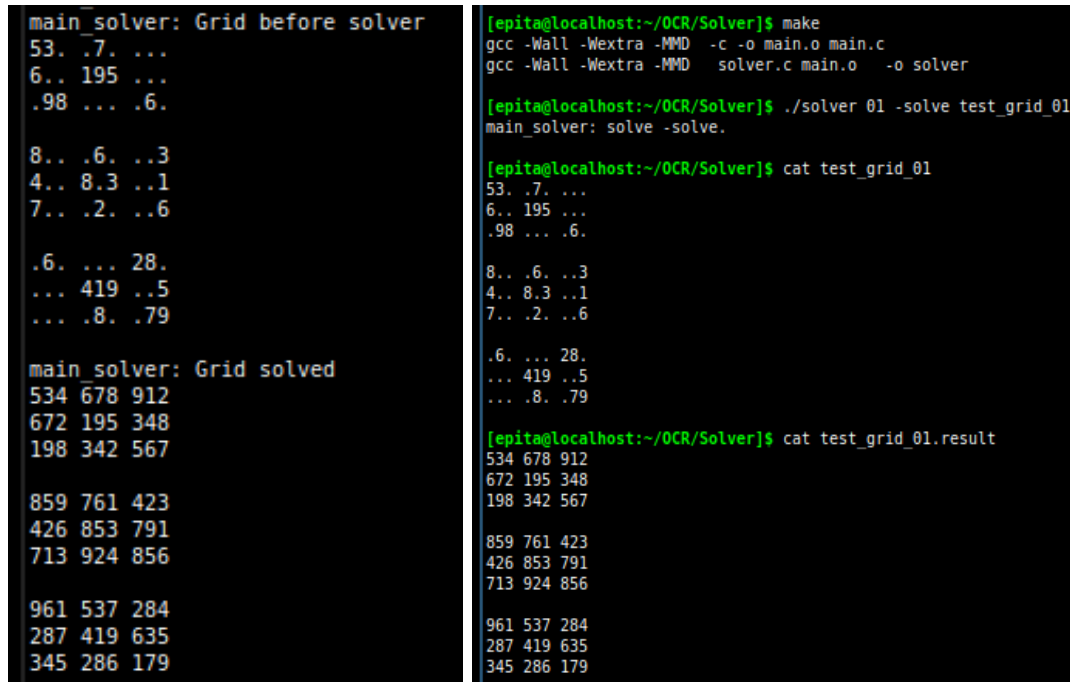


Figure 4.2: Fichiers résultant de la découpe

4.4 Résolution du sudoku

Pour résoudre le sudoku, nous récupérons une grille contenue dans un fichier. Après avoir récupéré celle-ci, nous parcourons chaque case de la grille en essayant de la remplir tout en vérifiant qu'il n'y a pas plusieurs fois le même nombre dans la même ligne, colonne ou diagonale. Si la grille n'est pas valide, nous retournons sur la dernière grille valide et testons avec d'autres chiffres jusqu'à en trouver une. Après avoir résolu la grille, nous l'affichons pour bien montrer qu'elle a été résolue et nous la sauvegardons dans un fichier.



```
main_solver: Grid before solver
53. .7. ...
6.. 195 ...
.98 ... .6.

8.. .6. ..3
4.. 8.3 ..1
7.. .2. ..6

.6. ... 28.
... 419 ..5
... .8. .79

main_solver: Grid solved
534 678 912
672 195 348
198 342 567

859 761 423
426 853 791
713 924 856

961 537 284
287 419 635
345 286 179

[epita@localhost:~/OCR/Solver]$ make
gcc -Wall -Wextra -MMD -c -o main.o main.c
gcc -Wall -Wextra -MMD solver.c main.o -o solver

[epita@localhost:~/OCR/Solver]$ ./solver 01 -solve test_grid_01
main_solver: solve -solve.

[epita@localhost:~/OCR/Solver]$ cat test_grid_01
53. .7. ...
6.. 195 ...
.98 ... .6.

8.. .6. ..3
4.. 8.3 ..1
7.. .2. ..6

.6. ... 28.
... 419 ..5
... .8. .79

[epita@localhost:~/OCR/Solver]$ cat test_grid_01.result
534 678 912
672 195 348
198 342 567

859 761 423
426 853 791
713 924 856

961 537 284
287 419 635
345 286 179
```

Figure 4.3: Affichage et sauvegarde de la grille résolue

4.5 Réseau de neurones

Dans l'objectif de reconnaître les caractères écrits dans chacune des cases, nous utilisons un Réseau de neurones (RN) qui a pour but de nous dire via l'image d'une case et du chiffre écrit à l'intérieur de quel chiffre il s'agit afin de numériser le sudoku présenté pour le résoudre.

4.5.1 Fonctionnement d'un réseau de neurones

Afin de détecter de quel chiffre il s'agit, le RN doit apprendre en amont, via une base de données plus ou moins, à faire la différence entre chaque chiffre écrit. Pour ce faire, un réseau de neurones imite les réseaux cérébraux humains.

Forward Propagation

Il s'agit de la propagation des calculs partant des neurones d'entrée (input layer) de gauche à droite à travers chaque poids et neurones intermédiaire jusqu'aux différents neurones de sortie (output layer).

Chaque neurone utilise une fonction d'activation. La plus standard et celle que nous utilisons est la fonction sigmoïd.

Fonction d'activation : Sigmoid

Elle se défini ainsi :

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

Back Propagation

La back propagation a pour but de calculer le taux d'erreur du réseau afin d'essayer de réduire ce nombre d'erreur en ajustant les poids et biais du réseau.

Pour ce faire, nous utilisons l'algorithme de la Gradient Descent (descente de gradient (GD) qui permet de trouver le minimum dans toutes fonctions convexes et nous permet ici de trouver le taux minimum d'erreurs que le réseau puisse faire.

Deux paramètres entre en jeu. Le "learning rate" qui multiplie l'action de la GD et permet théoriquement d'atteindre le minimum plus rapidement mais un learning rate trop grand peut avoir l'effet inverse :

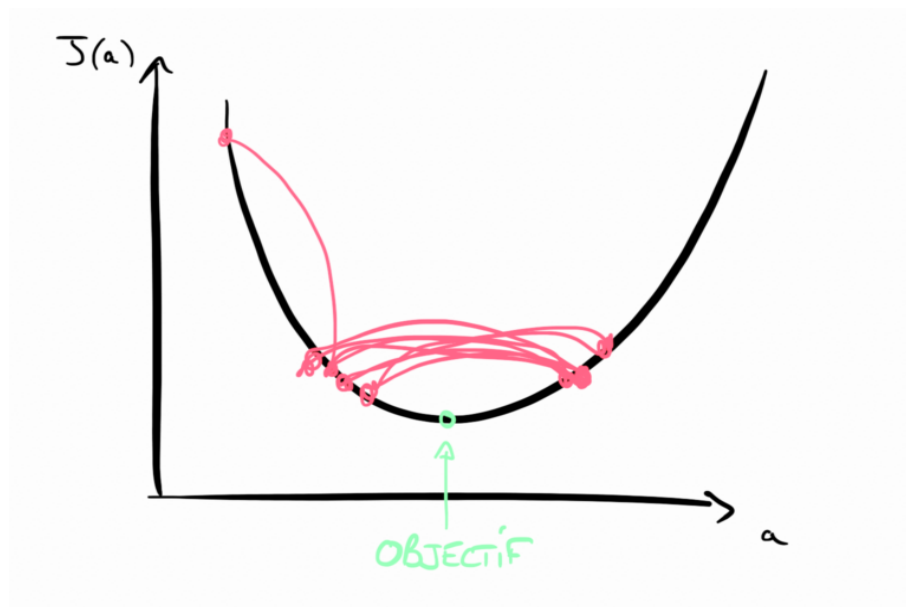


Figure 4.4: Tableau GD avec learning rate trop grand

A contrario, un learning rate trop faible nous demandera davantage d'itérations pour trouver l'objectif optimal. C'est alors qu'intervient le second paramètre qui est le nombre d'epochs (d'itération) d'apprentissage. Ainsi, il est de notre ressort de trouver les paramètres idéaux pour un apprentissage efficace et rapide.

Pour user de la GD, nous utilisons la dérivé de la fonction d'activation : la sigmoïd dérivée.

Dérivé de la fonction d'activation : dSigmoid

Elle se défini comme suit :

$$x * (1 - x)$$

Mise à jour des poids et biais

A chaque fin d'itération, il est primordial de mettre à jour tous les poids et tous les biais du réseau dans le but d'optimiser les prochaines forward propagations. Pour ce faire, nous utilisons les résultats de la GD qui nous permet de diminuer le taux d'erreurs.

4.5.2 Preuve de concept (porte ou-exclusive)

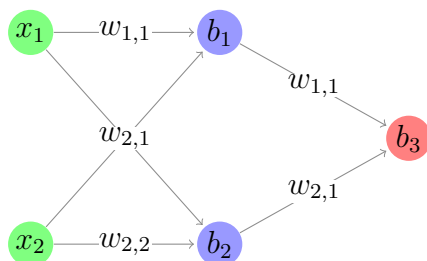
Pour cette soutenance intermédiaire, il nous a été demandé une preuve de concept d'un réseau de neurones basique permettant d'apprendre le comportement de la XOR qui se définit comme suit :

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4.5: Tableau XOR

Paramètres du réseau

Pour l'apprentissage, nous avons utilisé un réseau de neurones basique dont les paramètres peuvent être représentés ainsi :



2 inputs(entrées) notées (x1, x2)

1 couche cachée avec 2 neurones (en bleu)

1 output(sortie) en rouge.

Chaque liaison est constituée d'un poids($W_{i,i}$) et chaque (B_i) est un biais.

Ainsi pour le neurone 1 de la couche cachée, la propagation se fait via la fonction :

$$f(x) = x2 * W_{2,1} + x1 * W_{1,1} + B_1$$

Et ce pour chacun des neurones du réseau. Grâce à la backpropagation qui permet de calculer le taux d'erreur du réseau ainsi que la fonction mettant à jour les poids et biais pour les ajuster. Nous avons réussi à obtenir des résultats convaincants.

```
iteration : 0
m_dimensions: 4 rows - 1 cols
[ 0.676848 ]
[ 0.681624 ]
[ 0.707081 ]
[ 0.643640 ]
```

Figure 4.6: Première itération

```
iteration : 1000000
m_dimensions: 4 rows - 1 cols
[ 0.998292 ]
[ 0.998153 ]
[ 0.001901 ]
[ 0.001641 ]
```

Figure 4.7: Dernière itération

Les 2 premières lignes correspondent aux inputs $[1, 0]$ et $[0, 1]$ qui obtiennent ainsi une probabilité moyenne de 99.8% de sorties égales à 1.

Les 2 dernières lignes correspondent aux inputs $[0, 0]$ et $[1, 1]$ qui obtiennent ainsi une probabilité moyenne de 0.001% de sorties égales à 1 soit près de 99.9% de sorties égales à 0.

Ce sont des résultats tout à fait acceptables pour un réseau de neurones à 2 neurones cachés.

4.5.3 Sauvegarde et chargement des poids

Après que le réseau de neurones soit entraîné, il nous faut sauvegarder tous les poids et biais du réseau afin de faire une prédiction en fonction des entrées que nous voulons via les paramètres entraînés au préalable.

Pour ce faire, nous utilisons un simple fichier texte dans lequel seront représentées les matrices des poids et biais de chaque couche. Dans le cas du réseau de neurones pour le XOR, nous obtenons ainsi 4 parties, les 2 premières correspondent à la première couche (hidden layer) et les 2 suivantes respectivement aux poids et biais de la deuxième couche (output layer).

Du fait de l'architecture simpliste des fichiers de sauvegardes des poids et biais, il nous est facile de les importer pour des prédictions futures.

4.5.4 Base d'images pour l'entraînement du réseau

Pour entrainer notre futur réseau de neurones qui a pour objectif de repérer quel chiffre est écrit sur une case du sudoku, nous avons préparé un dataset(base de données) de plusieurs images de chacun des chiffres allant de 0 à 9. Il nous sera possible dans le futur d'importer les lettres A, B, C, D, E et F dans le cas où nous travaillerons sur des sudoku hexadécimaux.

Pour ce faire, nous utilisons la base **MNIST** et un programme nous permettant d'exporter des fichiers .ubyte les différentes images de 28x28 pixels distribués dans 10 dossiers correspondant à chaque chiffre en fonction de leur label respectif.



Figure 4.8: Dossiers de la base de donnée



Figure 4.9: Exemple du dossier 6

4.5.5 Mise en place du réseau principal de neurones

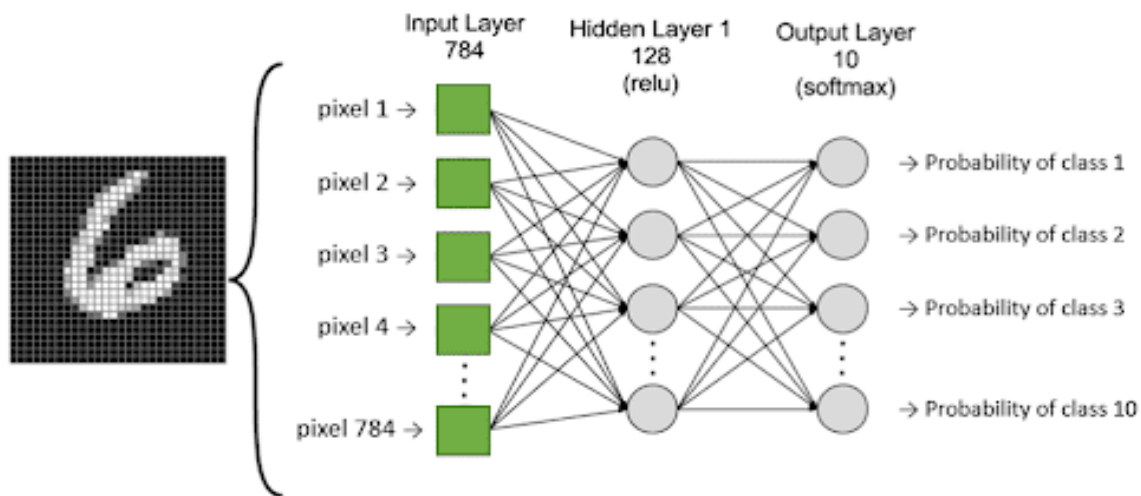


Figure 4.10: Exemple d'architecture viable

Nous avons décidé d'insérer comme entrée à notre réseau de neurones des images de 28 pixels par 28 pixels soit

$$28 * 28 = 784$$

neurones pour la couche 1.

Ensuite, nous n'utiliserons qu'une seule couche cachée avec un nombre de neurones inférieur à 784 puis 10 sorties indiquant chacune d'elles la probabilité qu'il s'agisse du nombre dont elle réfère.

Ainsi, la décision sera prise en fonction du neurone de sortie qui délivrera la probabilité la plus forte (proche de 1).

Fonction d'activation : Softmax

Défini comme suit :

$$y_i = \frac{\exp(x_i)}{\sum_{p \geq 0} \exp(x_p)}$$

La fonction d'activation **Softmax** sera utilisée lors de la dernière couche afin de forcer le réseau de neurones à renvoyer des vecteurs de probabilité, ce qui semblerait plus adéquat pour notre problème de classification à multiples réseau de sorties.

4.6 Ressenti du projet

4.6.1 Erwann Lesech

Au premier abord, j'étais transané par ce projet. Savoir implémenter son premier réseau de neurones me passionnait et ce fut définitivement le cas. Après plusieurs difficultés et de nombreux essais dans d'autres langages plus haut niveau, nous avons trouvé une configuration satisfaisante. C'est suite à cette initiation que je suis encore plus pressé de mettre en place notre véritable réseau de neurones pour la reconnaissance des caractères dans le mois qui suit.

4.6.2 Guillaume Jolival

Ce projet fut très enrichissant pour moi, j'ai pu découvrir et approfondir des domaines très intéressants, comme l'implémentation d'un réseau de neurones ou encore le découpage d'une image. De plus, le solver était une partie assez sympathique à réaliser. Avancer et finir ce projet va être encore plus instructif, et j'en suis fort satisfait.

4.6.3 Alexandra Wolter

J'ai beaucoup aimé travailler sur cette première partie du projet. Les deux parties sur lesquelles j'ai travaillé sont fonctionnelles bien que j'ai parfois rencontré quelques difficultés avec l'utilisation de SDL. Néanmoins, le projet avance avec un très bon rythme et le travail de groupe au sein de notre équipe fonctionne très bien ! J'ai hâte de continuer le projet sur le reste du semestre et de réellement commencer le travail sur l'interface graphique qui ne va pas être si simple !

4.6.4 Valentin Gibbe

Pour ma part, cette première partie de projet s'est bien passé. Malgré de nombreux problèmes techniques (algorithme de Hough qui ne fonctionne pas), j'ai réussi à les surpasser. Je suis très content de ce que j'ai réussi à faire et de ce que j'ai appris durant ce début de projet. Jamais auparavant je ne me serais cru capable de repérer les cases d'un Sudoku ni de traiter une image.

Je trouve qu'il y a une bonne cohésion d'équipe et nous avançons tous très bien sur nos parties. Nous avons du quelques fois réajuster la répartition des tâches afin que tout le monde est la même quantité de travail dans le groupe.

4.6.5 Avis général

Nous sommes satisfaits de la manière avec laquelle notre groupe avance et accomplit tous nos objectifs fixés. Notre gestion via git est très performante et suite à nos récentes difficultés, l'utilisation adéquate de cet outil afin de revenir à de précédents envois nous a été d'une grande utilité.

Concernant le groupe en lui même, chacun est conscient de la possibilité de se reposer sur les compétences des autres pour assurer le déroulement infaillible de notre projet malgré les difficultés individuelles ce qui nous permet par ce rapport de vous démontrer notre respect des attentes en temps voulu, c'est à dire cette première soutenance.

Nous sommes en capacité lors de cette première soutenance de présenter la totalité des éléments demandés et plus encore.

En effet, notre traitement d'image permet, à partir d'une grille de sudoku, de supprimer les couleurs, de détecter les contours de l'image ainsi que la grille avec le contour de chacune des cases.

Il est aussi possible d'effectuer une rotation manuelle de l'image afin de faciliter le traitement de cette dernière et la détection des chiffres.

Après la détection des cases, nous pouvons toutes les séparer pour les enregistrer en 81 images afin d'être présenté au réseau de neurones pour reconnaissance optique des chiffres.

Par la suite, nous pouvons démontrer via la preuve de concept que notre réseau de neurones basique est en capacité d'apprendre le comportement d'une porte logique XOR ainsi que de sauvegarder ses poids et biais. Ces derniers peuvent être chargés ultérieurement pour effectuer des prédictions après entraînement préalable du réseau.

Il est possible évidemment de résoudre le sudoku et de sauvegarder le sudoku complété dans un fichier.

Notre groupe est complètement soudé et prêt à poursuivre sur sa lancée pour implémenter le réseau principal de neurones, la rotation automatique et enfin une interface utilisateur simple, intuitive et ergonomique afin de présenter en Décembre un logiciel fini et de qualité.

LIST OF FIGURES

1.1	Sudoku imprimé	2
4.1	Exemples de découpes de l'image	10
4.2	Fichiers résultant de la découpe	10
4.3	Affichage et sauvegarde de la grille résolue	11
4.4	Tableau GD avec learning rate trop grand	12
4.5	Tableau XOR	13
4.6	Première itération	14
4.7	Dernière itération	14
4.8	Dossiers de la base de donnée	14
4.9	Exemple du dossier 6	15
4.10	Exemple d'architecture viable	15

LIST OF TABLES

2.1	Tableau de la répartition des tâches, R = Responsable & S = Suppléant . .	4
3.1	Estimation des avancées des différentes tâches	5
3.2	Respect des prévisions de l'avancement du projet	6

ACRONYMS

GD Gradient Descent (descente de gradient 12, 13, 18

OCR Optical Character Recognization 2

RN Réseau de neurones 11

ROC Reconnaissance Optique des Caractères 2

XOR Porte **ou-exclusive** 6, 13, 14, 17, 18

SOURCES

Réseau de neurones

Machine learnia

Hough Transform

Wikipedia HIPR

Sobel filter

Wikipedia

Découpage de l'image

WikiSDL