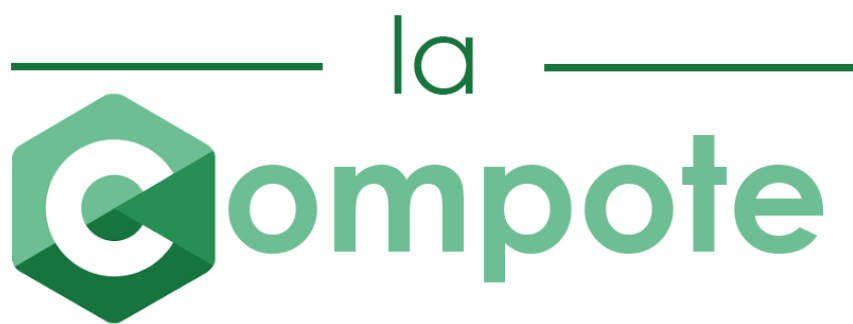


OCR

Première soutenance



Oscar MORAND
Liane DAVID
Jolan DUCLUZEAUD
Julien DELBOSC

Table des matières

1	Introduction	3
2	Présentation du groupe	3
2.1	Le groupe "La compote"	3
2.2	Les membres	3
2.2.1	Oscar Morand	3
2.2.2	Liane David	3
2.2.3	Jolan Duchuzeaud	4
2.2.4	Julien Delbosc	4
3	Avancement du projet	5
3.1	Chargement image et suppression des couleurs (Liane)	5
3.2	Rotation manuelle de l'image (Liane)	5
3.3	Détection de la grille et de la position des cases (Jolan et Julien) . . .	6
3.4	Découpage de l'image (Jolan et Julien)	7
3.5	Algorithme de résolution de sudoku (Oscar)	8
3.6	Concept de réseau de neurones (Oscar)	10
4	Récit de la réalisation	15
4.1	Oscar	15
4.2	Liane	15
4.3	Jolan	16
4.4	Julien	16
5	Avances et retards	17
6	Pour la soutenance finale	17
7	Conclusion	19

1 Introduction

Dans le cadre de notre projet de troisième semestre à Epita, nous devons réaliser un logiciel de type OCR (Optical Character Recognition) qui permet de résoudre une grille de sudoku depuis une image. Ce projet est à réaliser par groupe de quatre personnes pour une durée de 4 mois, de septembre à décembre 2021.

2 Présentation du groupe

2.1 Le groupe "La compote"

Le groupe "La Compote" s'est formé tout naturellement au début de l'année peu après la rentrée. Les membres de l'équipe se sont rassemblés surtout en fonction des affinités car nous avons appris avec le projet de S2 qu'il était important de travailler avec des personnes de confiance, avec qui la communication est facile. Bien que notre nom de groupe n'ait pas vraiment de sens et vienne à l'origine d'une blague, nous sommes quatre élèves sérieux et bien motivés à réussir ce projet.

2.2 Les membres

2.2.1 Oscar Morand

Je m'appelle Oscar. Je m'occuperai dans un premier temps de l'algorithme de résolution de sudoku puis enfin et mon plus gros travail sera de réaliser le concept de réseau de neurones pouvant apprendre l'opération binaire OU EXCLUSIF. Je suis très curieux et motivé de réaliser mon premier vrai réseau de neurones et content que ce soit dans un langage comme le C avec peu de bibliothèques disponibles pour véritablement créer ce réseau de A à Z et de comprendre véritablement tout ce que je fais, notamment les notions de mathématiques cachées derrière les propagations avant et arrière. Ce projet sera une autre occasion, à la suite du projet de S2, de travailler en groupe sur un gros projet tel que l'OCR, et, ayant de très bons souvenirs de cette dernière expérience, j'ai hâte de mener à bien ce nouveau défi avec mes camarades.

2.2.2 Liane David

Je m'appelle Liane. Je serai en charge du pré-traitement des images pour cette première phase de travail sur ce projet. Je me suis découvert l'année dernière un intérêt inattendu pour la programmation en faveur de l'image grâce à certains tp de C#. L'OCR est l'opportunité de découvrir un peu plus de cet aspect. Ce projet est également une occasion pour moi de m'exercer au langage C et d'en apprendre davantage sur la création de réseau de neurones, un domaine dans lesquelles j'ai peu de connaissances, ainsi que les techniques de reconnaissances de formes et de caractères. D'autre part, travailler en équipe, qui plus est avec ses amis, est une grande source de motivation, ce projet OCR sera sans conteste une expérience enrichissante que je suis ravie de réaliser avec mon groupe.

2.2.3 Jolan Ducluzeaud

Moi c'est Jolan ! J'adore la programmation, c'est tout un univers avec des possibilités infinies. Et parmi ces possibilités, me voilà confronté à ce projet OCR. Mon rôle dans ce projet est de repérer le sudoku et tracer les lignes de découpage afin de découper chaque case du sudoku en une nouvelle image avec l'aide du travail de Julien. Je trouve les méthodes utilisées pour détecter des formes dans une image très intéressantes, et les implémenter dans un code est un défi qui, une fois relevé, est vraiment satisfaisant ! Ce projet m'apporte des connaissances et une manière de réfléchir à des solutions de problème différentes de d'habitude, et cela alimente ma curiosité et me motive de plus en plus. Chers camarades de "La Compote", codons cet OCR!!!

2.2.4 Julien Delbosc

Je m'appelle Julien. Pour ma part je travaillerai d'abord avec Jolan pour trouver des méthodes de détection des lignes du sudoku, et ensuite de mon cote, je m'occuperai de la segmentation de l'image pour enregistrer une a une les cases du sudoku afin de, par la suite, pouvoir nourrir le réseau de neurone. Je suis excité de pouvoir réaliser a nouveau un projet de groupe avec mes camarades, qui plus est de l'OCR, un sujet que je trouve particulièrement intéressant en informatique et qui possède énormément d'applications concrètes. De plus ce projet sera une bonne occasion de se pencher sérieusement sur l'apprentissage du C qui est un langage incontournable de nos jours. Je suis impatient de me plonger dans la réalisation de ce projet avec mes amis.

3 Avancement du projet

3.1 Chargement image et suppression des couleurs (Liane)

Nous avons choisi d'utiliser SDL 1.2 comme bibliothèque logicielle pour l'affichage et le traitement d'images car nous avons auparavant pu l'apprivoiser lors d'un des tp de programmation. Ce dernier nous a considérablement servi pour l'étape de chargement et l'affichage de l'image, ainsi que pour comprendre plus aisément les fonctions de la documentation SDL.

La suppression des couleurs s'est fait en deux parties : les nuances de gris puis le noir et blanc. Pour modifier les couleurs de l'image, il faut quoi qu'il en soit parcourir chaque pixel de l'image et récupérer sa valeur RGB composée de trois octets correspondant respectivement aux composantes rouge, vert et bleue du pixel. Dans le cas du filtre "grayscale", on donne la même valeur à chaque composante pour obtenir la nuance de gris du pixel, avec la formule : $0.3 * R + 0.59 * G + 0.11 * B$.

Enfin, pour transformer l'image en noir et blanc, une fonction qui contraste l'image est d'abord appliqué. Cet algorithme

on détermine tout d'abord les valeurs RGB minimale (**rgb_min**) et maximale de l'image (**rgb_max**). Nous avons seulement besoin de connaître celle de la composante rouge puisque les 3 composantes sont égales suite au filtre "grayscale". Puis nous calculons une nouvelle valeur de la couleur du pixel de cette manière :

$$\text{contrasted_R} = (R - \text{rgb_min}) * 255.0f / (\text{rgb_max} - \text{rgb_min})$$

Ce qui nous donne le RGB du pixel avec **rgb_min** en borne inférieur et **rgb_max** en borne supérieur, ce calcul "remplace" la valeur dans cette échelle. Pour finir, cette valeur est comparée avec un seuil que l'on détermine pour le moment à 128, si elle est inférieur le pixel deviendra noir (RGB = (0,0,0)), sinon blanc (RGB = (255,255,255)).

3.2 Rotation manuelle de l'image (Liane)

Le principe simplifié de la rotation est d'application la formule de rotation d'un point de coordonnées (x,y) à chaque pixel pour un angle θ donné. Pour le moment, l'angle est choisi par l'utilisateur en le passant en argument de la commande lorsque qu'il exécute le programme. L'angle est donné en degrés, il est converti ensuite en radian au début de l'algorithme.

$$\begin{aligned}x' &= x * \cos(\theta) - y * \sin(\theta) \\y' &= x * \sin(\theta) + y * \cos(\theta)\end{aligned}$$

Pour ne pas "écraser" de valeurs en tournant les pixels de l'image originale, il nous faut en créer une copie. Toutefois, en effectuant une rotation de l'image, elle a de fortes chances de sortir en partie du cadre. Afin de garder le visuel de l'image entière tournée, nous devons changer les dimensions de la copie. Pour se faire, on calcule les

coordonnées des coins de l'image suite à leurs rotations, et, après plusieurs comparaisons entre les des coordonnées (x,y) de chaque coin, on obtient les coordonnées minimum et maximum de la fenêtre souhaitée. En les soustrayant, on connaît la hauteur et la largeur de l'image post-rotation que l'on peut utiliser pour créer une nouvelle surface à la bonne taille.

Finalement, on effectue un parcourt de cette image copié. Pour chaque pixel, on calcule ses coordonnées (x', y') après rotation autour du centre (x0, y0), on vérifie qu'elle sont toujours comprises dans les dimensions, puis on récupère le pixel à cette position sur la surface originale. Si les coordonnées sont en dehors de l'image alors le pixel sera noir.

$$\begin{aligned}x' &= (x - x0) * \cos(\theta) - (y - y0) * \sin(\theta) + x0 \\y' &= (x - x0) * \sin(\theta) + (y - y0) * \cos(\theta) + y0\end{aligned}$$

Le pixel est placé, cette fois-ci sur la nouvelle image, aux coordonnées (x, y) initiales. Pour finir, cette image est retournée.

3.3 Détection de la grille et de la position des cases (Jolan et Julien)

Afin de pouvoir résoudre un sudoku, il faut déjà voir les cases. Pour ce faire, nous découpons chaque case pour en faire une nouvelle image à donner au réseau de neurones. Mais pour en arriver là, il faut repérer le sudoku dans l'image et repérer le positionnement des cases. Pour ce faire, Jolan a implémenté l'algorithme de Hough qui permet par un système de vote dans une matrice de récupérer des coordonnées polaires et de les utiliser pour tracer les lignes d'une image. Tout naturellement, la Transformée de Hough va nous permettre de tracer les lignes les plus importantes de l'image et par conséquent identifier le sudoku dans l'image. Le système était le suivant : On crée une matrice de taille $\theta \times \rho$, on regarde chaque pixel et s'il est noir : on fait varier un angle θ de 0 à 180 degrés et on calcule :

$$p = x * \cos(\theta) + y * \sin(\theta)$$

FIGURE 1 – Transformée de Hough

Puis incrémente de 1 aux coordonnées θ et ρ de notre matrice. Une fois tout les pixels parcourus, on récupère les coordonnées des maximums de la matrice et on trace la fonction $y(x)$ sur l'image en colorant les pixels d'une autre couleur ou en augmentant la valeur d'une composante RGB :

$$y = \frac{-x * \cos(\theta)}{\sin(\theta)} + \frac{\rho}{\sin(\theta)}$$

FIGURE 2 –

Ainsi, nous voici avec un sudoku prêt à être découpé !

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 3 –

De plus, là où plusieurs lignes se croisent les pixels sont plus intenses, et donc définissent un coin de case. Cette petite astuce va nous être très utile dans le découpage de l'image !

3.4 Découpage de l'image (Jolan et Julien)

Par la suite, il nous faudra nourrir notre réseau de neurones avec des cases de sudoku pour, dans un premier temps l'entraîner, et ensuite pour reconnaître les chiffres. Afin de récupérer ces cases, nous avons d'abord détecté la grille de sudoku tout en mettant en surbrillance les droites et particulièrement les croisements des droites.

Après avoir parcouru la documentation du C99, nous avons conclu que la méthode 'SDL_BlitSurface()' était la plus intéressante pour faire la segmentation des cases. Cette fonction prends en paramètre la surface de départ qui contient les données a copier, la surface d'arrivée ou on va coller les données copiées ainsi que 2 'SDL_Rect()' qui vont délimiter les zones de copie et de collage respectivement dans les surfaces de départ et d'arrivée. Un 'SDL_Rect()' est défini par :

- x, y : les coordonnées de départ du Rectangle
- w, h : les dimensions du Rectangle

C'est pour renseigner les coordonnées x et y du 'SDL_Rect()' que les coins des cases surlignés lors de la détection de la grille vont être utiles, en effet, c'est un moyen sûr de reconnaître les coins d'une case. Pour ce qui est des dimensions du Rectangle, étant donné que nous travaillerons exclusivement avec des sudokus de 9x9, nous pouvons tout simplement récupérer la taille en pixels du sudoku et la diviser par 9. Et voilà, nous avons pu récupérer les cases du sudoku :

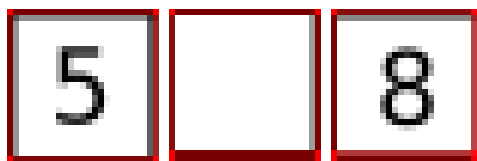


FIGURE 4 – Output de la segmentation

3.5 Algorithme de résolution de sudoku (Oscar)

La résolution d'un sudoku depuis une photo passe bien évidemment un algorithme de résolution de sudoku. Pour tester cet algorithme, nous devons concevoir un programme nommé Solver qui s'exécute en ligne de commande avec comme paramètre le nom du fichier contenant le sudoku à résoudre.

Parseur La première étape a été la réalisation d'un parseur permettant de lire des fichiers et d'en extraire le contenu, afin de créer un tableau correspondant au sudoku contenu dans le fichier. Pour l'instant, tous les sudokus qui nous seront proposés sont de format 9x9. Il a donc été plutôt simple de réaliser la lecture du fichier dans ces conditions, car, sauf erreur, les fichiers que l'on devra parser auront toujours la même structure : une structure imposée par le sujet, composé de 11 lignes dont 9 lignes de 11 caractères et 2 lignes vides. Les cases vides sont représentées par le caractère '.' et il y a des espaces entre chaque groupe de 9 cases. Il est nécessaire tout d'abord d'initialiser un tableau avec des valeurs nulles. Nous utiliserons ici un tableau à deux dimensions de taille 9 par 9. On peut ensuite commencer le parsing en appelant la fonction Load, appelée avec comme paramètres le nom du fichier à ouvrir (l'argument de la commande rentrée en ligne de commande) et le tableau précédemment créé. Cette fonction de remplissage ne renvoie pas de nouveau tableau mais modifie directement celui passé en paramètre et n'a donc pas de valeur de retour. On commence par ouvrir un fichier en mode lecture en faisant attention aux cas d'erreur ou le fichier ne peut pas être ouvert, puis on remplit la grille en bouclant à travers notre fichier et en lisant les valeurs (ici les cases vides représentées par des points sont remplacées par le caractère '0'). On finit par fermer le fichier tout en vérifiant s'il n'y a pas eu d'erreur lors de la fermeture.

Algorithme de résolution Viens ensuite l'étape la plus importante puisque il faut remplir et résoudre cette grille grâce à un algorithme de résolution de sudoku. Pour cela, on appelle une fonction chapeau Solve qui va appeler la fonction récursive principale. Dans cette fonction récursive, si la case sur laquelle nous sommes actuellement est vide, on va tester à la suite les différentes valeurs que peut prendre la case et qui ne rentrent pas en conflit avec les autres valeurs déjà mises (d'après les règles du sudoku), pour chaque valeur on va ré-appeler la fonction sur la case d'après et ainsi de suite. Si aucune valeur ne marche pour une case, alors les valeurs mises précédemment dans les cases d'avant sont incorrectes et on remonte les appels récursifs. On continue ce processus jusqu'à arriver à la fin du tableau. Dans ce dernier cas, on sait que le sudoku est rempli et que toutes les valeurs sont cohérentes, on peut donc mettre fin à la récursion en la remontant jusqu'à l'appel récursif initial. Il est

nécessaire de réaliser des fonctions auxiliaires qui ont pour but de déterminer, pour une valeur donnée dans une case, si la ligne, la colonne et le carré correspondant possèdent déjà cette valeur. Si c'est le cas, alors on ne peut pas placer la valeur dans la case et on essaie avec la prochaine. Le tableau passé en paramètre de la fonction Solve va également être modifié directement et on obtient ainsi un sudoku complété que l'on peut visualiser à l'aide d'une fonction PrintGrid.

```

oscar morand@LAPTOP-F7KQM9V8:~/OCR_LaCompote/solver$ ./solver test
Avant remplissage:
 0 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 0

 0 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 0

 0 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 0

Après remplissage:
 0 0 0  0 0 4  5 8 0
 0 0 0  7 2 1  0 0 3
 4 0 3  0 0 0  0 0 0

 2 1 0  0 6 7  0 0 4
 0 7 0  0 0 0  2 0 0
 6 3 0  0 4 9  0 0 1

 3 0 6  0 0 0  0 0 0
 0 0 0  1 5 8  0 0 6
 0 0 0  0 0 6  9 5 0

Après résolution
 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 5 – Impression de l'état du tableau après les différentes étapes

Enregistrement Finalement, on peut enregistrer notre sudoku complété dans un nouveau fichier. Ce fichier, d'après les consignes, ce nouveau fichier devra avoir le même nom de que le fichier d'origine à l'exception d'un .result ajouté à la fin du nom du fichier. Pour cela, on appelle la fonction Save qui prend en argument,

comme la fonction Load le nom du fichier et le tableau correspondant au sudoku complété. Dans cette fonction, on concatène au nom du fichier la chaîne de caractères ".result" et on ouvre un fichier avec ce nouveau nom en mode écriture. Il suffit ensuite d'écrire chaque valeur du tableau, en ajoutant des espaces et retours à la ligne lorsque nécessaire pour respecter la convention d'écriture et le format imposé. On peut terminer la fonction sur la fermeture du fichier.

```

oscar morand@LAPTOP-F7KQM9V8:~/OCR_LaCompote/solver$ cat test
... ..4 58.
... 721 ..3
4.3 ... ...

21. .67 ..4
.7. ... 2..
63. .49 ..1

3.6 ... ...
... 158 ..6
... ..6 95.
oscar morand@LAPTOP-F7KQM9V8:~/OCR_LaCompote/solver$ cat test.result
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

356 492 718
792 158 436
841 376 952

```

FIGURE 6 – Affichage du contenu des fichiers contenant le sudoku initial et le sudoku complété

3.6 Concept de réseau de neurones (Oscar)

Le réseau de neurones est une voire la partie la plus importante de ce projet. Pour sa réalisation et son bon fonctionnement, il est nécessaire de commencer avec un prototype plus simple qu'un réseau de neurones capable de reconnaître des chiffres et il a donc été demandé de réaliser un réseau capable d'apprendre la fonction OU EXCLUSIF (ou XOR).

N'ayant pas encore d'interface graphique, il a été nécessaire de créer un programme en ligne de commande nommé ici "testNN" (NN = neural network). Depuis ce programme, nous pouvons réaliser plusieurs actions différentes représentées par les paramètres de la commande. On commence tout d'abord par choisir le mode de création du réseau de neurones. En passant le paramètre "-create", on va initialiser un tout nouveau réseau de neurones avec des poids et biais aléatoires. Avec le paramètre "-load [chemin]", on va ouvrir le fichier désigné par le chemin [chemin] et récupérer

ses données pour créer le réseau de neurones correspondant. Une fois le réseau créé, on choisit l'action souhaitée. Nous pouvons tout d'abord entraîner notre réseau avec le paramètre "-train [nombre d'entraînements] [taux d'apprentissage] [chemin]", on peut donc modifier le nombre d'entraînements et le taux d'apprentissage et passer un chemin d'enregistrement du nouveau réseau. On peut également tester notre réseau avec le paramètre "-test [entrée 1] [entrée 2]" et le résultat de la prédiction du réseau de neurones avec ces deux valeurs en input sera comparé avec le résultat de l'opération XOR sur ces deux valeurs. Si l'utilisateur avait choisi le mode -create, on peut également simplement enregistrer ce réseau initialisé aléatoirement dans un fichier ayant pour chemin l'argument passé après -create.

Structure du réseau Le XOR n'étant pas une tâche très compliqué à apprendre pour un réseau de neurones, la structure du réseau choisit est très simple.

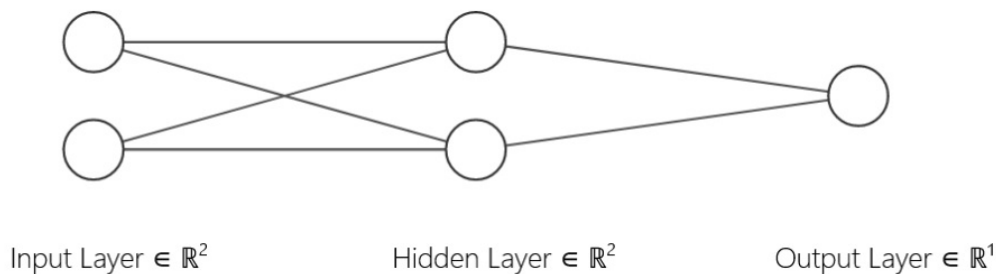


FIGURE 7 – Structure du réseau de neurones choisit

Comme représenté ci-dessus, notre réseau de neurones est composé de trois couches : une couche d'inputs de deux neurones donc les valeurs peuvent être 0 ou 1, une couche cachée de deux neurones et une couche d'output de un neurone, contenant le résultat de la propagation avant du réseau. Le résultat de la prédiction du réseau peut être obtenue en arrondissant la valeur de l'output. Afin d'augmenter la lisibilité et compréhension du code, des structures personnalisés ont été utilisés pour représenter le réseau. On a tout d'abord la structure réseau de neurones (struct NN) qui contient le nombre de couches, un tableau avec le nombre de neurones par couche et un pointeur de type (struct Lay) qui est la structure correspondant à nos couches. On retrouve dans chaque couche le nombre de neurones de cette couche et un pointeur de type (struct Neu) : la structure représentant un neurone. Chaque neurone contient l'input (v), l'output (actv), le biais du neurone (bias) et les poids sortant du neurone (outWeights) mais également les variations de ces valeurs (respectivement dv, dactv, dbias, dw), ce qui nous servira pour la partie propagation arrière.

Propagation avant Afin de calculer les sorties d'un réseau de neurones à partir des entrées, nous devons doter notre programme d'un algorithme de propagation avant (ou feed-forward). Cette notion est assez simple est ne consiste qu'en un calcul d'une valeur pour chaque neurone.

$$\text{actv}_j^i = \sigma \left(\sum_k \text{out_weights}_{jk}^i * \text{actv}_k^{i-1} + \text{bias}_j^i \right)$$

FIGURE 8 – Formule de la valeur de sortie d'un neurone

D'après la formule ci-dessus, on peut voir que la valeur d'entrée du neurone est la somme des poids des connexions entrantes depuis les neurones de la couche précédente multiplié par les valeurs de sortie de ces neurones, on additionne finalement le poids du neurone pour obtenir la valeur d'entrée. Pour passer de l'entrée à la sortie du neurone, on applique une fonction d'activation à la valeur d'entrée.

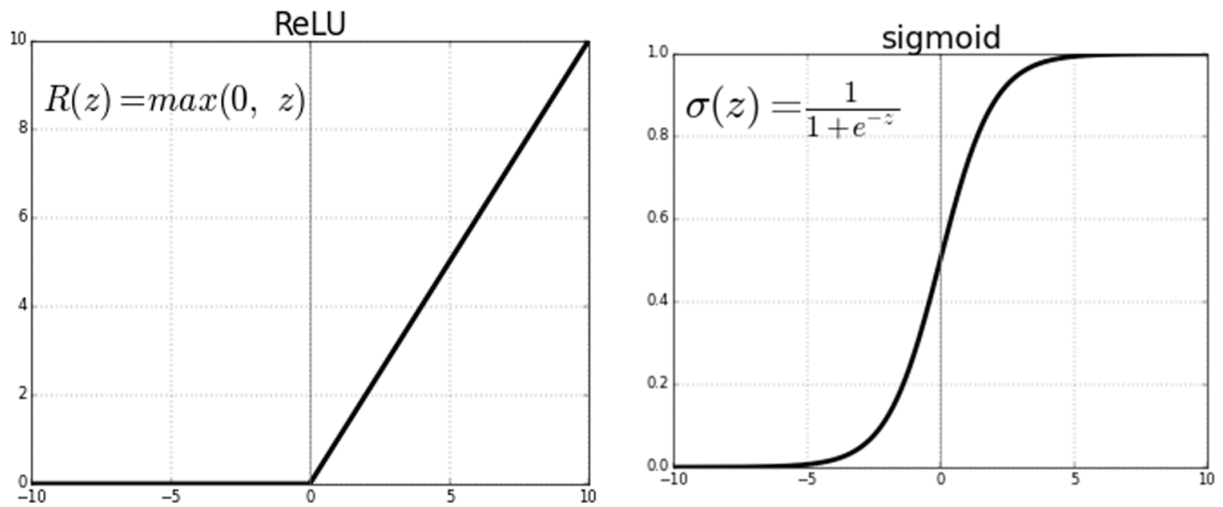


FIGURE 9 – Fonctions d'activation Relu et Sigmoid

Beaucoup de fonctions d'activations existent mais on utilisera dans le cas du XOR la fonction Relu pour les couches cachées et la fonction Sigmoid pour la couche d'output (dont les formules et courbes sont affichées ci-dessus). On effectue ainsi ce calcul à la chaîne, en utilisant les valeurs des calculs précédents, couche par couche, jusqu'à arriver à la couche d'output, obtenant ainsi le résultat de la prédiction du réseau de neurone.

Propagation arrière Nous en venons à la partie la plus compliqué mais primordiale pour l'apprentissage du réseau de neurones. Pour que le réseau puisse l'améliorer jusqu'à ce qu'il ne se trompe (presque) jamais, il est nécessaire de l'entraîner et de lui faire apprendre de ses erreurs. Pour mettre une valeur sur cette erreur, on utilise une fonction appelée "fonction coût".

$$C = \frac{1}{2n} \sum_x \|\text{actv}^i(x) - y(x)\|^2$$

Ici, la fonction coût utilisée est la fonction de coût quadratique, dont nous allons surtout utiliser la dérivée. Le but ici dans la propagation arrière est de minimiser

la fonction coût, c'est à dire trouver les valeurs des poids et biais pour lesquelles le résultat de la fonction coût sera minimale. On va donc chercher comment varie cette fonction coût par rapport aux différentes valeurs de notre réseau de neurones. On va par exemple chercher comment varie la fonction par rapport à chaque poids de la dernière couche. L'expression de ce changement est en fait la dérivée partielle de la fonction coût par rapport au poids d'un neurone :

$$\frac{\delta Cost}{\delta weight} = \frac{\delta input}{\delta weight} \frac{\delta output}{\delta input} \frac{\delta Cost}{\delta output}$$

En utilisant la règle des dérivations en chaîne, on peut obtenir cette dérivée partielle comme composition de trois dérivées partielles connues. Premièrement la dérivée partielle de l'input sur le poids correspond à la dérivée de la fonction qui nous avait servi à calculer l'input d'un neurone en faisant seulement varier le poids, cette dérivée est égale à l'output du neurone de la couche précédente. On a ensuite la dérivée partielle de l'output en fonction de l'input, ce qui correspond à la dérivée de la fonction d'activation (sigmoid ou Relu). La troisième est la dérivée partielle de la fonction coût en fonction de l'output, ce qui correspond à la dérivée de la fonction quadratique de coût. On peut procéder de la même manière avec les biais dont la formule sera légèrement différente. Pour les couches cachées, le calcul sera légèrement plus compliqué car repose sur les valeurs des couches suivantes. Cependant, réalisant la propagation de la fin du réseau vers le début, nous pouvons réutiliser les valeurs précédemment calculées, simplifiant grandement les calculs. On peut stocker les variations de chaque valeur grâce à notre structure neurone.

Entraînement Une fois toutes les variations calculées, on peut passer véritablement à l'apprentissage du réseau de neurones en actualisant les nouvelles valeurs. On ne veut pas directement actualiser les valeurs en faisant la différence entre l'ancienne et sa variation mais plutôt calculer la moyenne des variations sur un groupe de tests et appliquer la variation résultante en suivant la formule suivante :

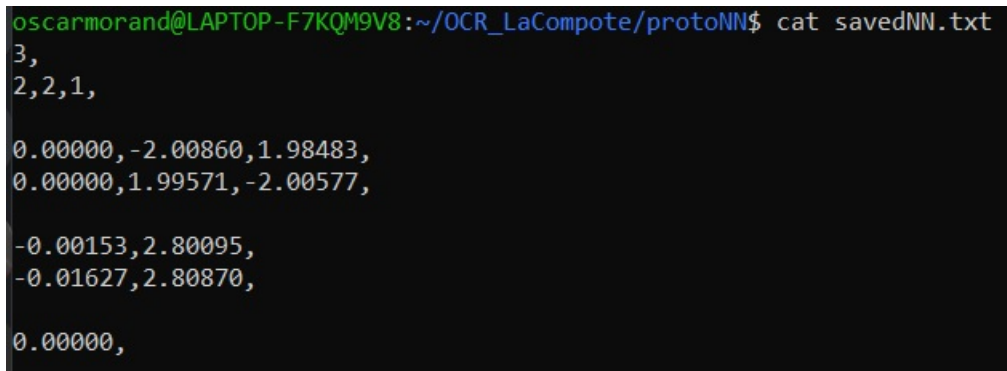
$$w' = \frac{\eta}{m} * \sum_k^m w^k - dw^k$$

On actualise ainsi tous les poids et biais de notre réseau de neurones. Le symbole η représente le taux d'apprentissage, la variation des valeurs du réseau se feront donc de façon proportionnelle à ce coefficient, il est cependant important de ne mettre un taux ni trop grand car cela pourrait entraîner un phénomène de rebond autour du minimum de la fonction coût, ni trop petit car il serait trop long d'atteindre ce minimum. La majorité de nos tests ont été effectués avec une valeur de 0,1. La valeur m de la formule ci-dessus représente le nombre de tests que l'on veut rassembler dans ce que l'on appelle un "batch". Ici chaque batch est composé de 4 tests, correspondant aux quatre valeurs d'entrées que peut prendre l'opération XOR ((0,0) (0,1) (1,0) ou (1,1)). On va effectuer un certain nombre de tests, nombre que

va rentrer l'utilisateur. Ici dans notre cas le réseau est assez petit et donc plutôt rapide à entraîner, on peut donc ainsi calculer plusieurs dizaines voire centaines de milliers de tests en quelques secondes et obtenir un réseau de neurones fonctionnel.

Enregistrement et chargement des poids et biais du réseau de neurones

A ce jour, n'importe quel réseau de neurone créé peut être enregistré dans un fichier il est possible de créer un tout nouveau réseau avec le chargement d'un fichier. Le réseau est sauvegardé dans un fichier au format texte (.txt) et suit une structure inventée par nos soins, cherchant à rendre le travail de parsing et d'enregistrement le plus simple possible en restant compréhensible par un être humain lors de la lecture du fichier texte.



```
oscar morand@LAPTOP-F7KQM9V8:~/OCR_LaCompote/protoNN$ cat savedNN.txt
3,
2,2,1,

0.00000,-2.00860,1.98483,
0.00000,1.99571,-2.00577,

-0.00153,2.80095,
-0.01627,2.80870,

0.00000,
```

FIGURE 10 – Affichage d'un exemple de réseau de neurones sauvegardé

Comme nous pouvons l'observer ci-dessus, on enregistre tout d'abord le nombre de couches en incluant la couche d'inputs et la couche d'outputs (ici il n'y a donc que 1 une couche cachée). On enregistre ensuite le nombre de neurones pour chaque couche (ici deux neurones d'input, deux neurones sur la couche cachée et un neurone d'output). On peut passer ensuite à la sauvegarde des poids et biais du réseau. Chaque couche est séparée par un saut de ligne pour mieux les différencier entre elles et les neurones d'une couche sont séparés par un retour à la ligne. Sur chaque ligne (pour chaque neurone), le premier élément à être enregistré est le biais du neurone, viennent ensuite les poids de chaque liaison sortante du neurone. On peut donc remarquer que chaque neurone possède un nombre de poids égal au nombre de neurones de la couche suivante et que la dernière couche, soit la couche d'output, est représentée uniquement par ses biais, car elle ne comporte pas de liaison avec une couche suivante. Le nombre de chiffres après la virgule a été décidé et est modifiable très simplement en changeant la valeur d'une constante dans le programme. L'extraction d'un réseau de neurones à partir d'un fichier, elle, renvoie un nouveau réseau de neurones.

4 Récit de la réalisation

4.1 Oscar

Algorithme de résolution de sudoku : La première tâche que j'ai eu à réaliser est l'algorithme de résolution de sudoku. En effet, j'ai estimé cette partie plus simple comparée à la confection d'un réseau de neurones et que ce que j'apprendrais lors de cette partie me servira pour la suite. Nous avions déjà réalisé un programme similaire en C# l'année dernière. Cependant, étant une étape réalisée au tout début du projet, ma maîtrise du langage C était encore très limitée surtout sur certains sujets comme les tableaux, pointeurs, allocation dynamique... J'ai donc préféré ne pas simplement réadapter mon ancien algorithme mais plutôt recommencer depuis le début et refaire chaque étape avec les outils assez limités qu'offre le C. Cette décision, bien que m'ayant coûté du temps, m'a permis d'être plus à l'aise avec le langage et de commencer à manipuler des fichiers, ce qui m'aura été bien utile dans la partie chargement et enregistrement des poids et biais du réseau de neurone. Finalement la partie de sauvegarde du sudoku n'a pas posé de problème étant plus facile et ressemblant à la fonction de lecture.

Concept de réseau de neurones : Ma tâche principale pour cette soutenance a été la conception du réseau de neurones. Cela a été un véritable défi pour moi, ma seule expérience des réseaux de neurones étant un tp de programmation l'année dernière donc en C#. J'ai cependant cherché à réaliser le plus gros du travail pour cette soutenance pour que toutes mes fonctions puissent s'adapter à n'importe quel réseau de neurones, de n'importe quelle taille. L'utilisation des structures à été au début d'une grande aide puisque j'avais l'impression de retrouver les regrettes classes de l'orienté objet. J'ai découvert et approfondis de nombreuses notions avec cette tâche comme l'utilisation plus poussée des pointeurs et de l'allocation dynamique. J'ai donc pris du temps à faire des recherches, réaliser des fonctions simples... pour découvrir ces outils qui me seront indispensables pour ce réseau. Je comprend désormais beaucoup mieux le problème de mémoire que le C apporte et arrive à éviter la majorité des erreurs que je rencontrais au début du projet. J'ai également rencontré des problèmes dans la compréhension des mathématiques derrière le fonctionnement du réseau, et plus particulièrement pour la propagation arrière. Mais après beaucoup de recherches et de documentation, j'ai réussi à complètement assimiler le fonctionnement de cet algorithme et de ses notions mathématiques.

4.2 Liane

Suppression des couleurs : On dit que le plus dur est souvent de commencer. Pour ma part, j'ai eu la chance que mon travail soit en lien avec le tp3 de programmation dont une partie consistant à du traitement d'image dont le chargement de l'image et le filtre "grayscale" (nuance de gris). De ce fait, j'ai pu récupérer la plupart de mes fonctions pour commencer mon travail sur ce projet.

Rotation de l'image : Pour la rotation, il m'a fallu me replonger dans certaines formules de mathématiques et dans des cours de géométries (certains étant des cours

de collègues...) car mes connaissances n'étaient pas très solides. Néanmoins, après être bien à l'aise avec le principe de rotation, l'algorithme n'a pas fait rencontré de difficultés particulières. Cela s'est un peu corsé, pour la copie de l'image car, la documentation de SDL 1.2 étant peu fournie, la fonction de création d'une surface avec ses multiples paramètres m'a posé quelques problèmes. Trouver la manière de calculer les dimensions de la nouvelle fenêtre a également été un peu compliqué, j'ai du passer par plusieurs schémas et tests pour avoir l'idée correcte de l'algorithme. Finalement, le programme de rotation a vite fonctionné, je n'ai rencontré que très peu d'erreurs le testant. J'ai pu ensuite mettre le paramètre de l'angle en argument de la ligne de commande, ce qui permet une meilleure utilisation.

4.3 Jolan

Recherche du sudoku et détection des lignes : Mon rôle était de trouver le sudoku, ses lignes et les coins de cases afin de permettre une découpe optimale des cases en image. La manière de procéder était en soi simple et facile à implémenter une fois avoir compris le sens de la Transformée de Hough. Le premier défi était de contrôler la matrice mère de cet algorithme, lui réserver les bonnes places mémoire, les bonnes valeurs et les récupérer correctement. Après quelques problèmes de compilation ici et là, ma matrice était remplie de toutes les bonnes choses dont La Compote avait besoin ! Cependant, le prochain défi était de tracer ces fameuses lignes, et donc inéluctablement avoir des dépassement de l'image quand il s'agissait de poser un nouveau pixel... Le combat fût rude mais j'ai finalement réussi à vaincre le bug. Tout semblait parfait, les lignes horizontales étaient tracées, mais celles verticales se révoltèrent ! Effectivement, un angle de 90° impliquait un coefficient infini pour tracer la droite... la solution : passer d'une fonction $y(x)$ à une fonction $x(y)$, et le tour était joué ! Toute cette aventure ne fût que le début d'un long codage, et la continuité de l'apprentissage du langage C.

4.4 Julien

Segmentation du sudoku : De mon cote, j'ai réalisé la segmentation du sudoku. Ce n'était pas une tâche très difficile mais grâce à celle-ci j'ai bien assimilé l'utilisation et le fonctionnement de la SDL. La seule difficulté que j'ai rencontrée a été que les lignes du sudoku n'étaient pas nécessairement d'un pixel de largeur. Cela pouvait par conséquent engendrer la génération d'un nombre trop important d'images d'un cote, et il nous est totalement inutile de posséder des doublons d'images d'un autre cote. Pour éviter cela, lors de mon parcours de la matrice des pixels, lors de la rencontre avec un pixel rouge vif aux coordonnées (x, y) , je rentre dans une boucle qui, incrémente x et/ou y en fonction de la couleur de ses voisins ; lorsqu'il n'y a plus de voisins rouge vif, je sors de la boucle et applique la segmentation sur ce pixel avant de continuer à parcourir ma matrice.

5 Avances et retards

Concept de réseau de neurones (Oscar) : Une des tâches devant être commencée pour la première soutenance était le chargement et l'enregistrement des poids et biais du réseau de neurones. Cependant, estimant que c'était une partie importante lors de la réalisation d'un réseau de neurones et que cette fonctionnalité nous servira très vite dans la suite du projet, cette tâche a été entièrement réalisée pour cette soutenance. Ainsi, une avance notable a été réalisée dans cette partie ce qui permettra de concentrer le travail pour la prochaine soutenance sur l'adaptation du réseau de neurones pour la détection de chiffres. Nous pouvons également considérer le fait d'avoir réalisé l'intégralité des fonctions de manière dynamique pour qu'elles s'adaptent à un réseau de neurones de n'importe quelle taille, nous donne une avance considérable car tout ce qui a été réalisé pour OU EXCLUSIF pourra être utilisé pour la détection de chiffres sans ou avec seulement très peu de modifications.

Pré-traitement (Liane) : L'objectif était d'accomplir la majorité des étapes du pré-traitement d'image pour cette première soutenance, cette tâche a été amplement réalisée. Le chargement de l'image et son affichage fonctionnent parfaitement. La suppression des couleurs a été réalisée avec la mise en nuances de gris puis en noir et blanc de l'image. Nous avons également une première fonction de renforcement des contrastes. La rotation manuelle se comporte très bien et l'utilisateur peut saisir l'angle qu'il souhaite en exécutant le programme. De plus, il y a pour le moment une sauvegarde simple de l'image en format BMP. Le nom du nouveau fichier reprend celui du fichier idéal en enlevant l'extension qui définit le format (.png, .jpeg, ...) et en ajoutant "_recup.bmp".

6 Pour la soutenance finale

Pré-traitement complet : Même si déjà bien avancé, le pré-traitement n'est pas complètement fini et il reste encore l'implémentation d'un algorithme de détection et d'élimination du bruit, bruit qu'il est important de réduire au maximum car il pourrait avoir un impact sur le réseau de neurones.

Segmentation du Sudoku : Pour la prochaine soutenance il nous faudra effectuer un peu de post-processing sur notre dataset de case de sudoku afin de retirer les bordures, harmoniser les tailles des images dans un format que le réseau de neurones pourra accepter ainsi que retirer les cases qui risqueraient de perturber l'apprentissage du réseau de neurones.



FIGURE 11 – Output de la segmentation à jeter

Ci-dessus nous avons un échantillon de cases sorties par la segmentation qui ont été détectées par les coins inférieurs du sudoku :



FIGURE 12 – Coins détectés menant aux trash output

Adaptation du réseau de neurone pour l'apprentissage des chiffres : Même si tout a été fait pour que notre code s'adapte à n'importe quel réseau de neurone, nous sommes conscient qu'il y aura de nombreuses modifications à effectuer pour réussir à faire apprendre à reconnaître des chiffres au réseau, ce qui représente une tâche bien plus compliquée que l'apprentissage du XOR. L'apprentissage sera également différent car ici le jeu de données n'est pas seulement constitué de 4 combinaisons possible d'entrées mais de milliers d'image de chiffres et le réseau de neurones étant beaucoup plus grand, le temps de calcul et donc l'apprentissage sera bien plus long.

Interface graphique : Pour la soutenance finale, il sera nécessaire de concevoir une interface graphique permettant d'utiliser tous nos programmes. En effet, pour l'instant, chaque partie fonctionne séparément et il est nécessaire de tout rassembler autour d'un seul logiciel, avec une interface graphique claire et intuitive pour que l'utilisateur comprenne facilement comment l'utiliser.

Sauvegarde du résultat : Une fois le sudoku résolu, il sera nécessaire d'enregistrer le résultat dans une nouvelle image, avec les réponses affichées dans une couleur différente pour pouvoir les distinguer de ce qui était déjà présent dans le sudoku avant résolution.

Tâches facultatives : Si nous parvenons à finir ces tâches à temps, nous comptons implémenter des éléments facultatifs proposés dans le sujet comme la reconnaissance de chiffres manuscrit ou la résolution d'hexadoku ainsi que la création d'un site internet.

7 Conclusion

Nous pouvons conclure que la première partie de la réalisation de ce projet est une réussite, avec presque aucun retard à déplorer et de l'avance sur certains points, le groupe La Compote a déjà réalisé un grand pas vers l'atteinte de notre objectif. La soutenance finale étant début décembre, nous avons devant nous plus d'un mois pour finir notre logiciel et nous sommes confiants quand à notre réussite. Notre motivation est toujours intacte malgré les problèmes rencontrés et l'ambiance et la coordination du groupe est toujours restée aussi bonne qu'à ses débuts.