SUDOKU SOLVER

KAZUKI
Io Lart
Anatole Gonnon
Jacques Remy
Axel Cochepin

27 octobre 2021

Table des matières

1 KAZUKI **3**
- 1.1 Anatole Gonnon . . . 3
- 1.2 Axel Cochepin . . . 3
- 1.3 Jacques Remy . . . 3
- 1.4 Io Lart . . . 4

2 Traitement de l'image (Io) **5**
- 2.1 Binarisation de l'image . . . 5
- 2.2 Rotation manuelle de l'image . . . 6

3 Détection et découpage (Axel) **8**
- 3.1 Découpage grille . . . 8
- 3.2 Détection grille . . . 8

4 Solver et affichage (Anatole) **11**

5 Réseaux de neurones (Jacques) **13**
- 5.1 Qu'est ce qu'un réseau de neuronne ? . . . 13
- 5.2 Stockage des valeurs . . . 13
- 5.3 Les différentes étapes de l'apprentissage . . . 14
  - 5.3.1 Forward Propagation (entrée vers la sortie) . . . 14
  - 5.3.2 Backpropagation (sortie vers l'entrée) . . . 15

6 Répartitions et avancements des tâches **16**
- 6.1 Répartition des tâches . . . 16
- 6.2 Avancement . . . 16

7 Ressentis **17**
- 7.1 Io . . . 17
- 7.2 Axel . . . 17
- 7.3 Anatole . . . 17
- 7.4 Jacques . . . 18

8 Conclusion **19**

‘

1 KAZUKI

Notre groupe **KAZUKAKI** est inspiré de la prononciation en japonais de nos initiales. KA pour le A, ZU pour le J et KI pour le I. Il faut savoir que le nom sudoku est né de l'abréviation japonaise de la règle du jeu « *Suji wa dokushin ni kagiru* » (chiffre limité à un seul). Su (chiffre) doku(unique), nous avons donc décidé de reprendre l'étymologie du mot pour créé un nom de groupe.

1.1 Anatole Gonnon

Je suis un passionné de jeux vidéo, en particulier car ils représentent un potentiel de créativité élevée : comme en dessin, on peut créer ce qui nous vient a l'esprit et la seule limite est l'imagination. Très a l'aise en mathématiques, avec un esprit cartésien, le monde de l'informatique me permet de m'exprimer a travers la recherche d'algorithmes efficaces. Les jeux vidéo rétro sont pour moi une parfaite représentation informatique grâce au mélange d'efficacités et d'astuces dans un espace limit e et avec un rendu parfait. En Terminale, j'ai choisi la spécialité "Science de l'informatique et du numérique". Cela m'a permis de découvrir la programmation (en particulier Python) et de savoir que je souhaitais faire des études supérieures dans ce domaine. Le projet ‘a réaliser en groupe consiste en l'élaboration d'un jeu de poker. Ce projet m'a donné l'expérience du travail en groupe et de la répartition des tâches en fonction des compétences de chacun.

1.2 Axel Cochepin

J'ai toujours été un grand passionné d'informatique ce qui a fait que je me suis rapidement dirigé vers des études scientifiques et ainsi finalement vers EPITA. Pour commencer, j'ai débuté dans le monde de la programmation avec la découverte de Python au collège et avec la création de petits sites en HTML/CSS. J'ai toujours été plutôt attirer par l'utilisation d'outils informatique dans tout ce qui touche au traitement d'image avec Photoshop ou la modélisation avec Solidworks et Blender. C'est pour cela que la partie détection et segmentation de la grille, qui fait partie du traitement d'image, a été très passionnante a travailler pour cette première soutenance.

1.3 Jacques Remy

Depuis mon jeune âge, j'ai l'ambition de consacrer ma carrière professionnelle aux métiers du numérique et j'ai rapidement compris que cela passerait par la validation d'un diplôme d'ingénieur. Très tôt sensibilisé à l'informatique, j'ai commencé à apprendre le python et le C++. Par simple curiosité au début, puis naissant un réel engouement et surtout une envie d'en apprendre davantage.
Le projet de l'année dernière fut une excellente expérience en laquelle je suis véritablement fier. Nous avons su proposer un jeu au dessus de la hauteur de mes attentes, et je suis sûr que je ne serai pas déçu de ce projet OCR non plus.
Bien que celui-ci soit moins libre, je suis tout autant intéressé, et notamment par le réseau de neurone qui sera ma partie lors de cette première soutenance.

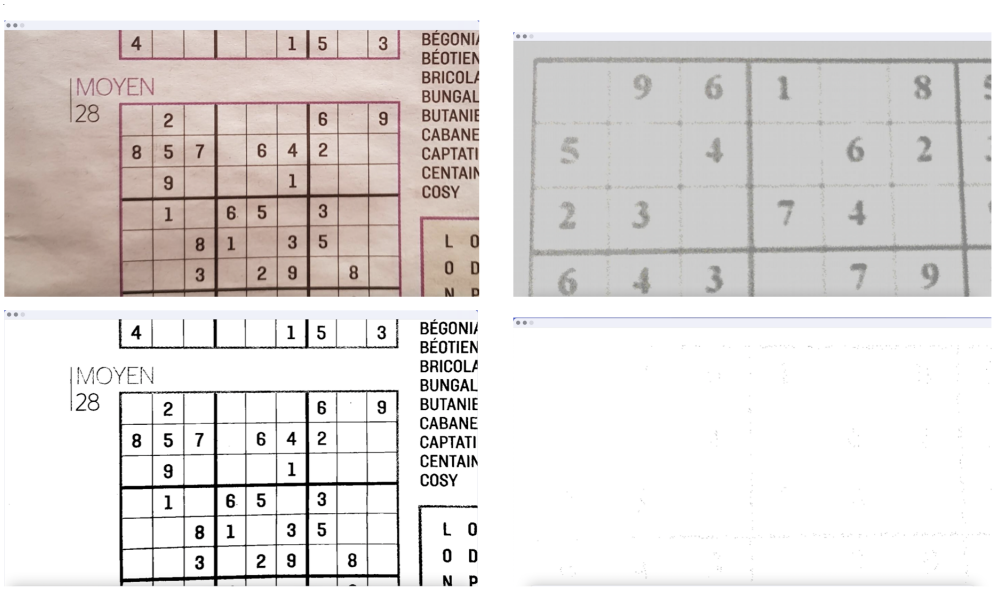
1.4 Io Lart

J'ai toujours voulu recevoir un apprentissage scientifique et technique, qui me permettra de participer à la conception, l'innovation et au progrès de notre environnement. La technologie, l'informatique sont des milieux d'avenir. Ils sont attrayants et ne cessent de se développer. Ils permettent de construire le monde de demain et je souhaite y participer. C'est pour cela que l'EPITA est une école qui correspond à mes attentes. De plus, la faible part de fille dans l'ingénierie informatique, m'a davantage poussé à choisir ces études. Je suis curieuse de ce qui m'entoure et l'informatique à ouvert en moi beaucoup de questionnement ainsi que beaucoup d'intérêts. Pour cette première soutenance je me suis occupée du traitement d'image, qui se réalise avant la résolution de la grille. Pour cette soutenance, j'ai réalisé un grand travail de documentation.

2 Traitement de l'image (Io)

2.1 Binarisation de l'image

Pour réaliser la binarisation en noir et blanc de l'image, j'ai commencé par modifier la fonction de notre tp 04. J'ai donc choisi un seuil moyen à 128 (256 valeurs, 256/2 = 128), toutes les valeurs de pixel supérieurs au seuil se transforment en blanc et celle en dessous, en noir. Après plusieurs tests, je me suis rendue compte que sur pas mal d'images, le rendu n'était pas de bonne qualité.
Comme on peut le voir par exemple sur cette image avec un exemple qui marche, et l'autre non...



C'est là, que le travail de documentation a commencé.

J'ai donc fais des recherches pour binariser efficacement une image en noir et blanc. Je suis tombée dans le domaine du traitement d'image et l'opération correspondante s'appelle la segmentation. J'ai vu qu'il existait différentes méthodes ; le feuillage, la classification, le clustering...
D'après la première réunion sur le projet, j'avais le souvenir d'avoir entendu parler de la méthode d'otsu, méthode, qui après renseignement, m'a parût très efficace, mais assez complexe à comprendre.
Il existe donc deux méthodes deux seuillages différentes (passer d'une image en niveau de gris en image binaire (noir et blanc)), le feuillage manuel et celui automatique qui correspond à la méthode d'otsu.

La méthode d'otsu consiste à séparer les pixels en deux catégories ; celle des pixels noirs et celle des pixels blancs, avec un k qui sépare les deux classes différentes. Le but de cette méthode est de trouver le k qui sépare au mieux le fond et les objets de l'image à traiter.
Pour calculer k, on va trouver la variance interclasse entre la catégories des pixels noirs et celle des pixels blancs, pour tous k possible entre 0 et 255 (les valeurs de pixels).
K sera donc le seuil et il représente, la plus haute variance interclasse.

J'ai pu déchiffrer des étapes nécessaires à cette méthode qui correspondent plus au moins aux différentes fonctions :

- On met l'image en niveau de gris, et on construit un histogramme de ses niveaux de gris.
- Normaliser l'histogramme (obtenir un histogramme dont toutes les valeurs sont comprise entre 1 et 0.). On obtient donc une distribution de probabilités en fonction de chaque valeur de pixel.
- Calcul de la probabilité qu'un pixel de l'image appartienne à la catégorie des pixels noirs. Cela se fait en calculant la somme de toutes les colonnes de l'histogramme comprises entre 0 et k.
- Calcul de la probabilité qu'un pixel de l'image appartienne à la catégorie des pixels blancs. En faisant simplement 1- proba(des pixels noirs) calculée dans l'étape précédente.
- Calcul de la variance interclasses entre la catégorie des pixels blancs et celle des pixels noirs
- Toutes les étapes sont répétés pour toutes les valeurs de k possibles.

Aillant su un peu tard l'existence de cette méthode, il a été très difficile de la comprendre et de finir les fonctions associées à ma compréhension. J'ai commencé, mais pas terminé.

2.2 Rotation manuelle de l'image

En ce qui concerne la rotation de l'image, j'ai utilisé plusieurs formules de trigonométrie.
Il faut bien distinguer l'image d'entrée et celle de sortie, ce qui m'a un peu pris de temps avec les librairies SDL.
La taille du fond de l'image de sortie sera supérieure à celle d'entrée, pour la calculer, j'ai utilisé cette formule :

X = x * cos(angle) + y*sin(angle) Y = y * cos(angle) - x * sin(angle)

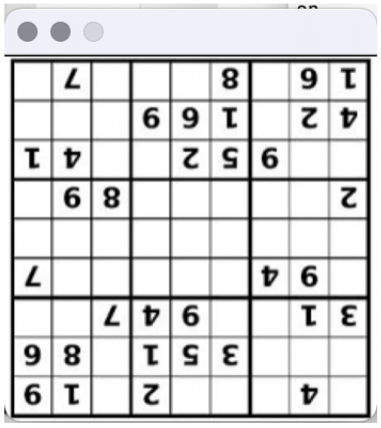
J'ai utilisé cette formule pour convertir les degrès en radians :
radian = 2*Pi*degree)/360

Et j'ai ensuite pris chaque pixel de l'image de sortie et on calcule le pixel source correspondant de l'image d'entrée.

Voici quelques exemples de la fonction rotation :

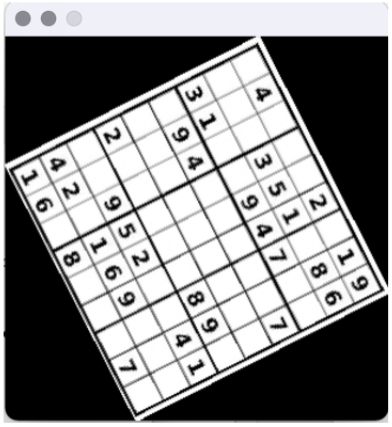
rotation de -180°C

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



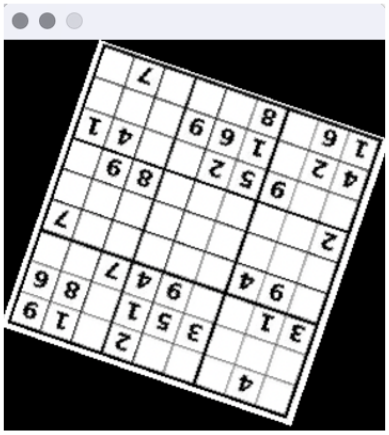
rotation de 63°C

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



rotation de 559°C

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



3 Détection et découpage (Axel)

3.1 Découpage grille

Pour commencer, nous avons commencé par le découpage de la grille. Tout d'abord, l'implémentation de ce découpage peut se faire avec une image droite et cadrer sur la grille de Sudoku. Le programme de résolution de la grille prend donc une image pour la découper en 81 cases de Sudoku qui pourront ensuite être traité par un programme de machine learning permettant de reconnaître les chiffres qui y sont présents. Pour cela, le programme récupère la largeur et la longueur de l'image pour la diviser en 9, et ainsi avoir les dimensions pour chaque case. À l'aide de ces dimensions, on peut trouver les coordonnées de chaque case et ainsi les découper. Ainsi une fois à la bonne coordonnée, nous avons juste à récupérer pixel par pixel une case et à la sauvegarder dans une nouvelle image que l'on enregistre dans un dossier à part répertoriant la totalité des cases à traiter. Les cases sont sauvegardées avec un nom sous la forme "57" où 5 et 7 représentent les positions dans la grille de sudoku.

3.2 Détection grille

Dans un premier temps, nous nous sommes dirigés vers la méthode "Hough Transform" pour réaliser la détection de la grille. Nous avons ainsi commencé à réaliser la détection par cette méthode, et avons donc programmé la possibilité d'appliquer un flou Gaussien à l'image.

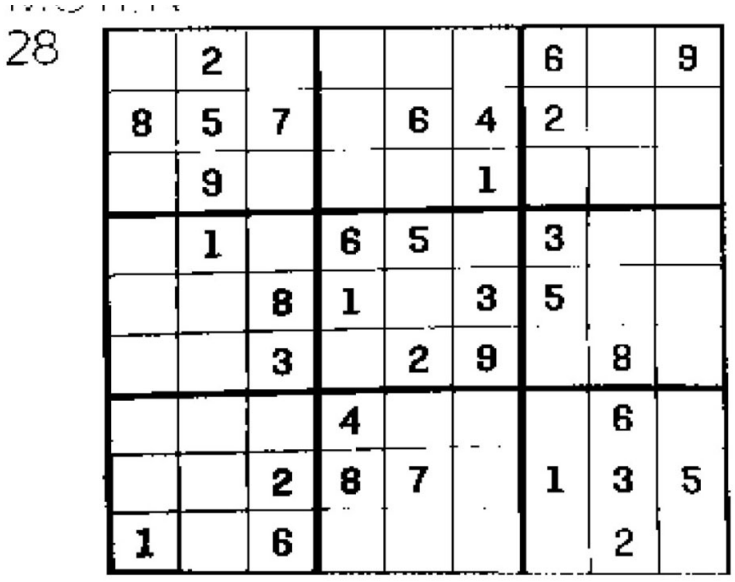
Image d'origine

Filtre Flou gaussien appliqué

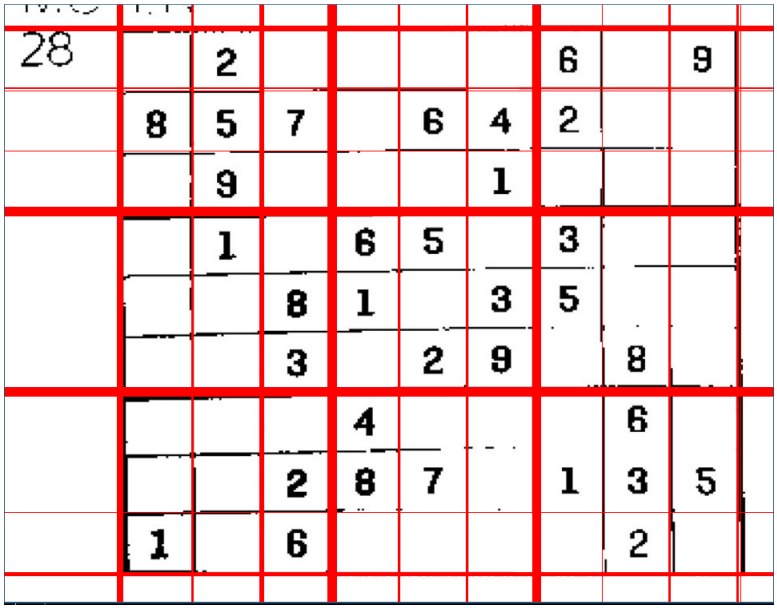
Mais après réflexions, nous avons réalisé que cette méthode utilisée de nombreuses étapes de traitement d'image qui ne sont pas forcément nécessaires à la détection de la grille, vu qu'une phase de traitement de l'image est déjà faite en amont. Nous avons ainsi décidé de partir sur une méthode par "histogramme", vu que cette méthode convient mieux au traitement de l'image réaliser en amont. Ainsi, nous allons pouvoir utiliser la propriété de l'image qui est d'être binarisé pour réaliser d'abord deux histogrammes. Nous aurons donc un histogramme pour la détection des colonnes et un autre pour les lignes. Chaque histogramme sera créé de la manière suivante :

- On parcourt une ligne/colonne
- Pour chaque pixel, on regarde s'il est noir ou blanc
- S'il est noir, on l'ajoute à la somme des pixels noirs de la ligne/colonne
- Pour chaque ligne/colonne, si la somme dépasse la longueur/largeur divisée par 2.5 alors on ajoute la somme à l'histogramme, sinon on met 0

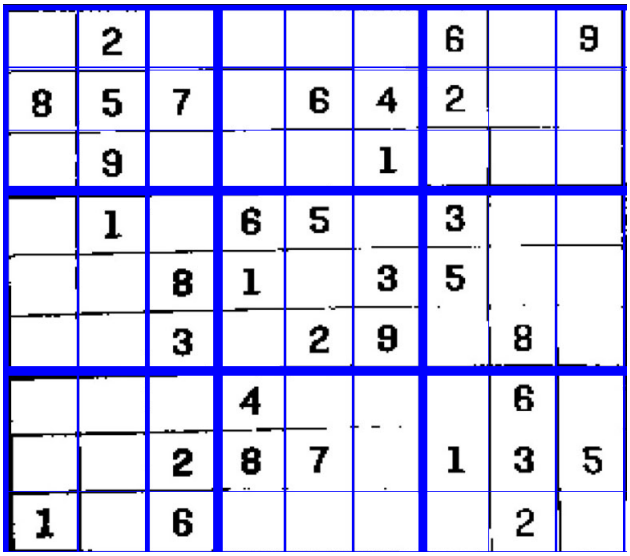
Ensuite, nous allons utiliser chaque histogramme pour trouver l'emplacement de la première et dernière colonne et ligne, et ainsi avoir la largeur et la longueur de la grille en réalisant une soustraction. Avec ces données, on peut donc rogner l'image pour récupérer la grille et ainsi la découper.



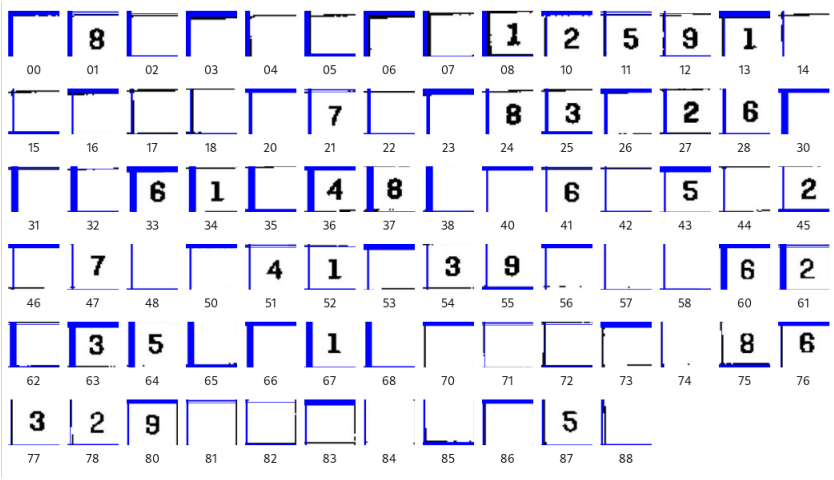
Comme vous pouvez le voir ci-dessus, pour la première étape nous récupérons une image binarisé, bien rotationné et relativement bien cadré sur la grille.



Ensuite, avec l'image précédente, nous allons donc détecter les lignes et les colonnes. Pour cela, nous allons remplacer tous les pixels de chaque ligne ou colonne détectées par des pixels rouge.



Avec cette nouvelle image et les deux histogrammes remplies nous n'avons plus qu'à nous en servir pour isoler la grille, comme ci-dessus.



Finalement, après avoir isolé la grille nous n'avons plus qu'à appliquer le découpage pour isoler les cases et les enregistrer dans un dossier pour leur futur traitement.

4 Solver et affichage (Anatole)

La consigne était de résoudre un sudoku présenté en format texte. Nous devons programmer un lecteur de texte qui nous renvoie une matrice représentant le sudoku. Donc, ce programme ouvre le fichier. Si celui-ci n'existe pas le programme envoie une erreur "parser : le fichier n'existe pas". Dans le fichier, il y a des chiffres et des points qui représentent les cases vides. Le programme examine chaque caractère jusqu'à la fin, vérifie la présence d'un chiffre ou d'un point et les placent dans la matrice à la bonne position. Les points sont des 0 dans la matrice. Si c'est un caractère autre qu'un chiffre ou un espace ou un saut de ligne ou un chiffre mal placé, ou si le fichier contient plus de caractères que nécessaire, le programme nous renvoie une erreur "parser : le fichier pas au bon format".

Ceci est un exemple de fichier texte qui représente un grille de sudoku non résolu

```
... ..4 58.
... 721 ..3
4.3 ... ...

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

3.6 ... ...
... 158 ..6
... ..6 95.
```

Dans cette partie, nous allons voir comment nous avons résolu le sudoku en question. Pour représenter le sudoku nous faisons une matrice à 2 dimensions de taille 9x9.

Dans cette matrice « grid », les cases vides du sudoku, les cases vides sont représentées par « 0 », les chiffres sont indiqués par leur valeur entre 1 et 9. La fonction « solveur » a pour argument la matrice et celle-ci appelle une fonction recursive et celle-ci prend comme arguments la matrice et deux chiffres (x,y) qui représentent les coordonnées des cases de la matrice. On appelle la fonction récursive avec les coordonnées (0,0). Cette fonction récursive renvoie un booléen. La fonction avance de gauche à droite et de haut en bas. Elle regarde le chiffre à la position de (x,y) de la matrice Si ce chiffre n'est pas 0, on incrémente x de 1 et lorsque x=9, on met x à 0 et on incrémente y de 1. On appelle la fonction récursive avec ces nouvelles coordonnées. Si le chiffre est 0, on prend 1, on regarde si le 1 ne figure ni dans la même ligne, ni dans la même colonne, ni dans le carré 3x3 et on le place dans la grille, on appelle avec la fonction récursive, et si la fonction récursive renvoie « faux », on essaye la même méthode avec 2, jusqu'à 9. Même si 9 ne peut pas être placé, la fonction retourne « faux » La fonction se termine quand y=10 et retourne « vrai ». Cette méthode s'appelle le Backtracking (brut force)

Après avoir résolu le sudoku, il faut que nous sauvegardions le résultat dans un fichier texte. Pour cela, on crée un fichier texte et on écrit les chiffres de la matrice dans le même format que le fichier de base, en remplaçant les points par les nouveaux nombres.

Ceci est un exemple de fichier texte qui représente un grille de sudoku résolu

```
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
```

En bonus pour cette première soutenance, nous avons décidé de créer une image qui représente la matrice. Pour cela nous utilisons la bibliothèque SDL. Cette bibliothèque permet de créer des images ou des fenêtres. Nous avons décidé de créer nous-mêmes un programme qui dessine des chiffres. Nous créons pour chaque chiffre une image de base qui est noire. Chaque chiffre est représenté par une liste d'entiers. Ces entiers représentent une ligne de l'image du chiffre. Cet entier en base 2 permet de savoir quand afficher la couleur blanche. Nous parcourons bit à bit cet entier, le pixel blanc est affiché lorsque nous rencontrons un 1. Nous avons créé une fonction qui construit les traits de la grille de sudoku. Nous avons dû créer une fonction qui permet de créer des carrés de couleur blanche pour éviter de construire une image de petite taille. Cela permet d'obtenir une image assez grande.

deux image de la grille de sudoku de taille different

5 Réseaux de neurones (Jacques)

Tout d'abord, voici un léger rappel du fonctionnement de la fonction XOR : La fonction XOR (aussi appelé OU exclusif) est un opérateur logique à deux opérandes, qui peuvent avoir chacun la valeur VRAI (1) ou FAUX (0), il associe un résultat qui a lui-même la valeur VRAI seulement si les deux opérandes ont des valeurs distinctes.

5.1 Qu'est ce qu'un réseau de neuronne ?

Le réseau de neurones est un élément de l'OCR très important car en effet c'est lui qui sera capable d'identifier les chiffres afin de réaliser le sudoku. Cette partie est compliquée à mettre en place car le réseau agit comme un cerveau : il est capable d'apprendre pour se perfectionner et ainsi avoir des résultats de plus en plus proches de la vérité.
Les réseaux de neurones sont constitués de plusieurs couches (au minimum deux) :

- couche d'entrée : les neurones de cette couche contiendront l'information que l'on souhaite tester
- couche de sortie : ces neurones nous donneront le résultat du problème que le réseau aura résolu
- une ou plusieurs couche(s) cachée(s) : dans cette couche les neurones serviront seulement pour le calcul, leurs valeurs ne seront exploitées que par le réseau de neurones. Pour notre part, nous avons décidé de réaliser un réseau avec deux neurones dans sa couche caché.

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

5.2 Stockage des valeurs

L'implémentation du réseau de neurone est sous forme de matrice. Nous avons donc créer une nouvelle structure Matrix, contenant le pointeur vers la liste stockant les valeurs de la matrice, sa taille ainsi que plusieurs fonction indispensables tel que l'initialisation d'une Matrice, le changement de ses valeur, etc. Notre réseau est donc principalement divisés en plusieurs matrices : deux pour les poids (une entre l'entrée et la couche cachée, et une autre entre la couche cachée et la sortie), deux pour les biais (une pour ceux des neurones de la couche cachée, et une autre pour les neurones en entrée), sans oublier toutes celles nécessaires aux calculs des dérivés, des gradients ainsi que des variations des poids et des biais. Chacune d'elles est initialisée en fonction du nombre d'entrées, du nombre de neurones dans la couche cachée et de neurones en sortie. Ce qui nous permet d'avoir seulement 3 variables à modifier si l'on veut changer la taille du réseau.

5.3 Les différentes étapes de l'apprentissage

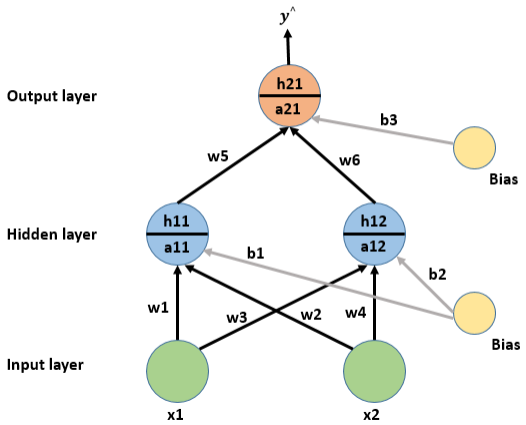
Le but de l'apprentissage est de permettre au réseau de neurone de s'améliorer. Cela consiste simplement à modifier les poids de chaque connexion jusqu'à qu'elles atteignent leurs valeurs optimales, mais cela nécessite plusieurs étapes pour parvenir à obtenir les nouvelles valeurs de poids (que l'on traitera par la suite). Il aussi possible d'agir sur la vitesse de d'apprentissage en rajoutant plus de neurones, en connectant les neurones d'entrée directement à ceux de sortie ou encore en rajoutant des neurones de biais (des neurones qui ne sont connecté qu'à la couche suivante mais pas à la précédente). Le pas d'apprentissage est une valeur qui permet de régler la vitesse à laquelle le réseau apprend mais aussi la précision de l'apprentissage. Plus le pas sera grand, plus l'apprentissage sera rapide mais moins précis, et inversement. Il existe différentes méthodes directes pour qu'elle soit le plus précis possible comme : le *Time-Based Decay*, le *Step Decay* ou bien l'*Exponential Decay* (méthodes que nous n'utiliserons pas ici car une implémentation manuelle est largement suffisante pour une fonction XOR).

5.3.1 Forward Propagation (entrée vers la sortie)

La première étape par laquelle passe notre réseau de neuronne est l'algorithme de calcul en avant (*Forward Propagation*). Cette algorithme permet de trouver les valeurs de sortie du réseau de neurones. La première étape est de calculer une valeur d'activation pour chacun des neurones. Cette valeur se calcule grâce aux poids placés sur les connections, elle correspond à la somme des poids des connections multipliés par la sortie des neurones correspondants de la couche supérieure. Cette valeur est ensuite appliquée à une fonction qui donnera une valeur de sortie. Cette fonction est la fonction sigmoïde, qui a la particularité de renvoyer des images entre 0 et 1, ce qui correspond exactement à nos valeurs de sorties (elles aussi comprises entre 0 et 1). La fonction sigmoïde est la suivante :

$$h1 = \frac{1}{1 + e^{-a1}}$$

Dans notre exemple, voici a quoi ressemble notre réseau de neurones :



a : variable de pré-activation, h : variable d'activation calculé à l'aide de la fonction sigmoïde.

5.3.2 Backpropagation (sortie vers l'entrée)

la rétro-propagation du gradient est une méthode pour entraîner un réseau de neurones en mettant à jour les poids de chaque neurone de la dernière couche vers la première. Pour pouvoir appliquer cet algorithme il va falloir que le réseau connaisse la valeur qu'il doit obtenir ce qui permettra de trouver l'erreur et permettre de mettre 'a jour les poids. Le processus d'apprentissage se passera en cinq étapes :

— Tout d'abord il faut effectuer le calcul en avant pour connaître toutes les sorties du réseau (*Forward Propagation*)

— Ensuite sur les neurones de sortie on calcule une valeur d'erreur, en connaissant leurs valeurs de sortie ainsi que celles qu'ils doivent avoir (valeur théorique connue). Soit t_k la valeur souhaitée de l'unité k et o_k sa valeur d'activation. T pour « target » et O pour « output ». On appelle d_k le « signal d'erreur » défini par la formule : $d_k = (t_k - o_k)\ f'(net_k)$
où f' est la dérivée de f. Si on utilise pour f la fonction sigmoïde, alors on peut montrer facilement que : $f'(net_k) = o_k\ (1 - o_k)$

— Ensuite on met à jour le poids de chaque connexion allant vers les neurones de sortie. La formule pour changer le poids w_{jk} entre l'unité de sortie k et l'unité j est (avec γ le pas de l'apprentissage) : $\Delta w_{jk} = \gamma\ d_k\ o_j$

— On calcule l'erreur sur la couche cachée précédant celle que l'on vient de corriger. La formule pour calculer le signal d'erreur dj d'une unité cachée j est : $d_j = f'(net_j)\ \Sigma_k\ d_k\ w_{jk}$
La somme s'applique pour toutes les connexions arrivant sur un neurone k et partant du neurone j.

— Enfin, On met à jour les poids des connexions allant sur la couche caché. Pour cela, on applique la formule pour changer le w_{ij} entre l'unité cachée j et l'unité d'entrée i : $\Delta w_{ij} = \gamma\ d_j\ o_i$

On répète les pas 1 à 6 pour les autres exemples à apprendre. L'ensemble de ces répétitions se nomme une itération. Après une itération, la sortie du réseau sera un peu plus proche de la bonne solution. L'algorithme « *backprop* » consiste à effectuer plusieurs itérations jusqu'à ce que l'erreur soit suffisamment petite.

6 Répartitions et avancements des tâches

6.1 Répartition des tâches

Tâches	Anatole	Jacques	Axel	io
Chargement d'une image				x
Suppression des couleurs				x
Prétraitement				x
Détection de la grille			x	
Détection des cases de la grille			x	
Réseau de neurones (XOR)		x		
Reconstruction de la grille	x			
Résolution de la grille	x			
Affichage de la grille résolue	x			
Sauvegarde de la grille résolue	x			

6.2 Avancement

Tâches	1ère soutenance
Chargement d'une image	100%
Suppression des couleurs	60%
Prétraitement	30%
Détection de la grille	66%
Détection des cases de la grille	66%
Réseau de neurones (XOR)	100%
Reconstruction de la grille	100%
Résolution de la grille	100%
Affichage de la grille résolue	50%
Sauvegarde de la grille résolue	50%

7 Ressentis

7.1 Io

Pour cette première soutenance, j'ai sous estimé le travail demandé. J'ai fais des fonctions assez naïves de ce qu'on me demander jusqu'à comprendre assez tard, qu'elles suffiraient pas. J'ai donc été un peu prise de cours par le temps. Cependant, le fait de faire de la documentation seule sur langage c, m'a vachement aidé à progresser, je l'ai ressentie notamment dans les TP. J'ai eu un peu de mal avec la rotation et la librairie SDL, mais ce n'ai que partie remise, je serais bien plus compétente, à la fin de ce projet, ce qui me motive encore plus. J'ai le sentiment d'avoir appris pas mal de choses, notamment une petite partie sur le traitement d'image qui était un domaine encore peu connu pour moi. Et j'ai été assez époustouflé des techniques existantes et efficaces sur le traitement d'image.

7.2 Axel

Avec ce début de projet, j'ai pu en apprendre beaucoup sur les différentes étapes de développement d'une application de résolution de Sudoku. J'ai beaucoup apprécié travailler sur ma partie, j'ai découvert les différentes méthodes de détection de paternes, et ainsi en apprendre vraiment plus dans le traitement d'image. J'ai ainsi pu mettre en application ce que nous avons pu voir en TP de C et notamment approfondir mes connaissances sur la bibliothèque SDL. Cette première avancée du projet m'a donc permis d'en apprendre beaucoup plus sur le C et le traitement d'image, mais aussi sur la gestion de projet en groupe et la répartition des différentes taches.

7.3 Anatole

Je trouve ce projet est plus intéressant que les années précèdent. Fait les fonction m'amuse assez et je découvre des méthodes extraordinaires. Il y a une bonne ambiance dans le groupe. Dessiner pixel par pixel m'a beaucoup amuser à faire. Prochaine je dois être plus accès pour mes camardes pour les aider rapidement en cas de problème trouvez quoi faire dans la suite du projet comme faire l'interface graphique du programme.

7.4 Jacques

La première partie de ce projet à été pour moi une véritable expérience. Programmer un réseau de neurone est un peu comme un rêve d'enfant qui se réalise. Grâce à l'infinité de documentations sur internet, j'ai pu en apprendre énormément sur le fonctionnement d'un réseau de neurone et son implémentation. Le travail de recherche à été l'étape principale pour cette partie : il me fallait comprendre comment réaliser ce réseau. C'est pour cela que je m'y suis pris le plus tôt possible, c'est à dire, dès l'instant où je savais que je m'occuperai de cette partie. Ce travail était nécessaire, car, comme pour beaucoup d'entre nous, le réseau de neurone était pour moi une chose totalement inconnu. C'est donc pour cela que la partie la plus compliqué fut, pour moi, la recherche dans ce *monde vaste et inconnu*.
Puis il m'a fallut réfléchir à une méthode pour implémenter cette fonction dans le langage C, je suis donc re-partie faire des recherches, mais cette fois ci, sur le langage que nous allions utiliser. Et cela m'a permis d'en apprendre plus sur les pointeurs, les *struct* ...
Une fois ces deux premières parties terminés, j'ai pu commencer l'implémentation de notre réseau de neurones. Cela m'a pris un peu de temps au début, il me fallait imaginer comment gérer chaque étapes. Mais une fois compris, il ne me restait plus qu'a appliquer les différents algorithmes (*forwardpropagation* et *backpropagation*).
Une fois le réseau de neurone implémenté, je me suis occupé de créer une fonction main et un makefile regroupant chacune de nos parties.

8 Conclusion

Pour conclure, après cette première période du projet, nous sommes dans l'ensemble satisfait du travail que nous avons produit. Toutes les recherches effectuées et les connaissances acquises sont nécessaires à notre formation, ce qui nous encourage à faire de notre mieux. Nous avons une bonne cohésion de groupe ce qui renforce notre travail.

En espérant continuer dans cette voie ou faire mieux rendez vous à la prochaine soutenance !

XOXO,

L'équipe KAZUKAKI.