OCRat

OCR S3

Rapport de soutenance

OCRat composé par :

DAHOUT Ryan
FEORE Anis
KERVELLA Alexis
COL Ugo

Contents

1 Introduction **2**

2 Les membres **2**

2.1 Alexis Kervella 2

2.2 Anis Feore 2

2.3 Ugo Col 2

2.4 Ryan Dahout 3

3 Répartition des tâches **3**

4 Prétraitement **4**

4.1 Filtre médian 4

4.2 Filtre de convolution 5

4.3 Filtre niveaux de gris 5

4.4 Filtre d'Otsu 6

4.5 Conclusion Pretraitement 6

5 Segmentation **7**

5.1 Segmentation 7

5.1.1 Transformée de Hough 7

5.1.2 Rotation Automatique 9

5.1.3 Interpolation Bilinéaire 10

6 Réseau de neurone **11**

6.1 Généralité sur le problème XOR 11

6.2 Structure 11

6.3 Phase d'apprentissage 12

6.3.1 Forward Propagation 12

6.3.2 Backpropagation 13

7 Résolveur de sudoku **14**

7.1 Présentation 14

7.2 Le Résolveur de Sudoku 14

7.3 Résolveur d'hexadoku 15

7.4 Ce qui a été convenu de faire 16

7.5 Ce qui est envisagé pour la soutenance finale 16

8 Avancée générale **17**

9 Conclusion **17**

1 Introduction

Ce document présente l'avancée du projet par OCRat du début du projet jusqu'à la première soutenance. Vous trouverez notamment dans ce document une présentation des membres du groupe, la répartition des charges, l'état d'avancement du projet et les aspects techniques de chaque partie du projet. La création de se groupe s'est décidé de manière très opiné. Nous étions déjà en groupe de deux (Ugo et Alexis) et (Anis et Ryan) mais un groupe devait être composé de quatre personnes donc nous nous sommes naturellement dirigé les uns vers les autres afin de travailler ensemble. Des la constitution du groupe nous avons appris a travailler ensemble. Nous avons organiser un appel discord afin de nous répartir les tâches, voir les contraintes des uns et des autres, et essayer de comprendre comment fonctionne chaque membre du groupe. Notre groupe se complète plutôt bien, en effet lorsque quelqu'un est bloqué sur un problème, il y'a toujours quelqu'un dans le groupe qui debloque la situation rapidement.

2 Les membres

2.1 Alexis Kervella

Je suis tombé dans l'informatique à mes 10 ans lorsque ma famille a acquis un Ipad, j'avais qu'une idée en tête c'était de mettre l'écran de l'iPad sur celui de la télé. Depuis j'ai toujours été curieux et j'ai jamais cessé de bidouiller des ordinateurs, des programmes etc.... Ce projet d'OCR me permettra d'améliorer mes compétences dans divers milieux informatiques notamment la détection de forme, comment fonctionnent les filtres, et savoir comment une image est traité par l'ordinateur.

2.2 Anis Feore

Je me suis déjà intéressé précédemment au fonctionnement des IA en général en regardant de nombreuses vidéos de vulgarisation sur ce sujet telles que celles de ScienceEtonnante ou encore de Defend Intelligence. Ainsi, je suis très content de pouvoir en apprendre plus sur ce sujet et d'avoir l'opportunité de développer mon propre réseau de neurones.

2.3 Ugo Col

Je suis tombé dans l'informatique pendant mon vol de déménagement à la réunion. Il y'a eu un problème d'ordinateur a bord ce qui a entrainé une panne dans le système de climatisation de l'avion. La situation n'était absolument pas dangeureuse mais était impressionnante. Depuis je me suis toujours interessé à l'informatique car je trouve ca fascinant de devoir réfléchir a tous les cas possible. Cet incident m'a conforté dans le choix de mes spécialités au lycée: NSI, SI et maths. Ce projet va me permettre de mettre en pratique ce que j'aime le plus dans l'informatique: devoir réfléchir a tous les cas possible. En effet pour le traitement de l'image il faut pouvoir s'adapter a toutes les images possibles.

2.4 Ryan Dahout

Je n'ai commencé la programmation qu' au lycée. Néanmoins, la programmation a rapidement évoqué un réel intérêt chez moi. J'ai commencé par des programmes simples et des petits sites internet. Malheureusement, je ne me plonge pas dans des plus gros projets. Ce projet est une super opportunité avec un sujet intéressant. Travailler sur la résolution d'un sudoku améliorera ma logique algorithmique. Développer un réseau de neurones me permettra de découvrir plus profondément le fonctionnement des IA.

3 Répartition des tâches

	Alexis	Ugo	Ryan	Anis
Prétraitement	R	S		
Rotation	S	R		
Segmentation	S	R		
Hough	S	R		
Détection grille	R	S		
Réseau de nerones			S	R
Sudoku			R	S

4 Prétraitement

Le prétraitement est une étape indispensable avant de commencer la reconnaissance de caratères. En effet, une image n'est jamais la même, elle peut varier en fonction des appareils photos, des conditions lumineuses, etc.... Le pretraitement permet de normaliser une image afin que elle puisse par la suite être compatible avec les différents traitements que l'on va lui appliquer (détéction des bords) puis lorsque l'on va la soumettre a la reconnaissance de caractères.

4.1 Filtre médian

Le filtre médian nous permet d'éliminer le plus possible les bruits de l'image (les imperfections). Ce filtre est très important car certaines images peuvent être prises avec des appareils photos de qualités moindres. Le filtre médiant est le premier traitement qui sera appliqué à notre image. Le filtre médiant est très simple de fonctionnement :

- On récupère les 8 pixels autour du pixel que l'on souhaite modifier
- On range les 9 valeurs par ordre croissante dans un tableau
- On récupère la valeur médiane du tableau soit l'indice 4

5	6	7
6	111	8
7	8	9

Cette matrice représente nos pixels autour du pixel que l'on souhaite modifier (en l'occurrence ici c'est 111).

5	6	6	7	7	8	8	9	111

On trie ensuite nos 9 valeurs en ordre croissant. On peut remarquer que 111 est à la fin du tableau car c'est une valeur dite aberrante on considère donc que c'est un bruit et donc qu'il faut le supprimer. On remplacera donc la case 111 par la médiane du tableau (en l'occurrence ici c'est 7).

Ci-dessous, on peut remarquer que l'image 1 est couverte de bruit, en utilisant le filtre médiant on sera donc capable de reconstruire une grande partie de l'image

Exemple :

4.2 Filtre de convolution

Le filtre de convolution permet d'appliquer plusieurs filtres. En effet, il utilise une matrice de convolution, et en fonction de cette matrice on peut obtenir différents résultats. Voici un exemple de différentes matrices de convolutions :

Augmenter le contraste	Flou	Amélioration des bords
0 0 0 0 0 0 0 -1 0 0 0 -1 5 -1 0 0 0 -1 0 0 0 0 0 0 0	0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0	0 0 0 -1 1 0 0 0 0
Détection des bords	**Repoussage**	**Filtre de Sobel**
0 1 0 1 -4 1 0 1 0	-2 -1 0 -1 1 1 0 1 2	-1 0 1 -2 0 2 -1 0 0

Pour appliquer ces matrices il suffit de prendre un carrée de 3 pixels par 3 pixels de notre image et d'additionner chaque case une à une, puis de multiplier le résultat de chaque case. Le résultat obtenu nous donne le pixel du milieu du carré. Ce filtre est très pratique car il permet d'appliquer plusieurs filtres avec une seul fonction. Nous l'utilisons pour amélioration des bords, augmentation du contraste, le filtre de Sobel.

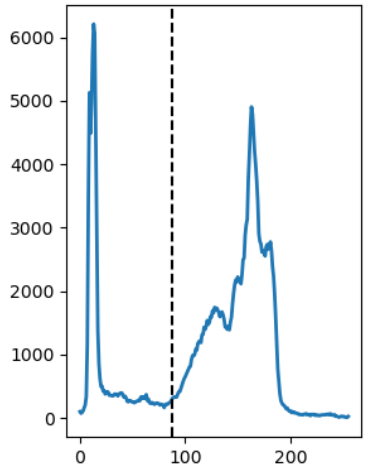
4.3 Filtre niveaux de gris

Le filtre niveau de gris est important car il permet de créer une image pour ensuite la traiter avec l'algorithme d'Otsu. Pour cela rien de plus simple on utilise la formule :

$$\text{Pixel} = 0.3 * (\text{pixel rouge}) + 0.59 * (\text{pixel vert}) + 0.11 * (\text{pixel bleu})$$

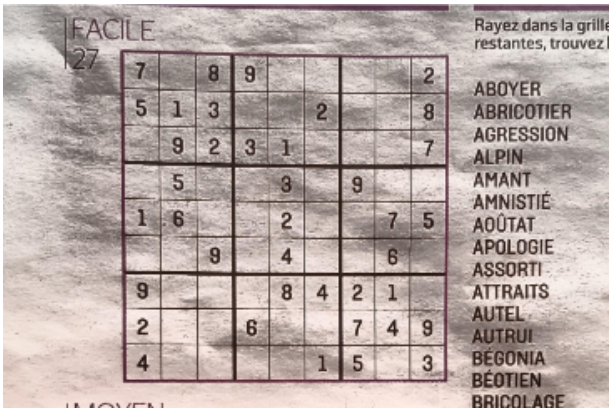
4.4 Filtre d'Otsu

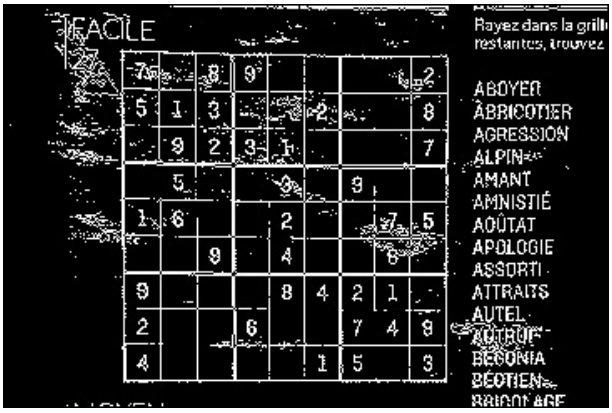
Le filtre d'Otsu est un filtre très pratique car il permet de bien faire la différence entre les éléments du premier plan et les éléments du second plan. Pour faire cela, il faut créer un histogramme des pixels à partir d'une image en niveaux de gris. Ensuite on calcul une valeur de seuil et tous les indices de l'histogramme au-dessus de cette valeur de seuil seront considérer comme étant au premier plan donc seront noir et celles au-dessus seront considérer comme étant au second plan donc seront blanche.



4.5 Conclusion Pretraitement

A l'heure actuelle, après l'application des differents filtres, on obtient :





5 Segmentation

Une fois le pretraitement terminé, les images sont depourvu de tout bruits, de toutes nuisances et la grille du sudoku est blanche sur fond noir. Avant de pouvoir utiliser le reseau neuronal sur ces images, il faut reussir à extraire chaque case contenant un chiffre ou non. Pour ca, 3 etapes nous attendent :
- La detections des lignes.
- Redressage manuelle (puis automatique) de l'image
- Le decoupage de l'image en fonction de ces lignes.

5.1 Segmentation

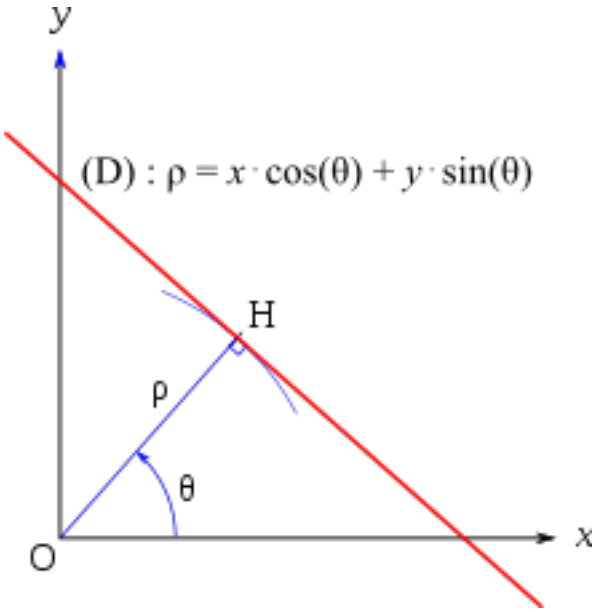
Pour la detection des lignes blanches sur fond noir, on a utilisé la Transformée de Hough.

5.1.1 Transformée de Hough

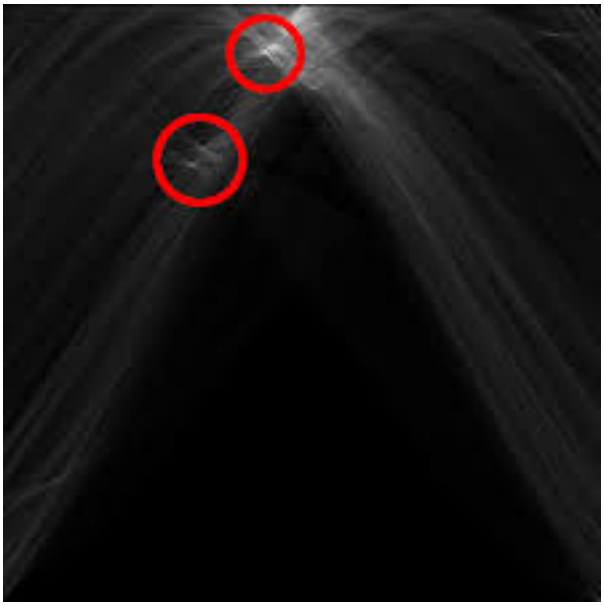
Le principe même de la Transformée de Hough est assez simple mais la transcription algorithmique est très délicate.
Avant tout, on pose $\theta \in [-\pi, \pi]$ et $\rho \in [-d, d]$ avec d defini comme la diagonale de notre image ($\sqrt{la * la + lo * lo}$ avec la = largeur et lo = longueur de notre image). Ce θ et ce ρ vont nous permettre de créer une matrice de taille ρ * θ. Cette matrice nous servira d'accumulateur.
Une fois tout les paramètres défini, on peut enfin commencer l'algorithme. Pour ce faire, on va parcourir tout les pixels de l'image en faisant varier x_1 et y_1. Pour chacune de ces variations, on va calculer ρ selon la formule :

$$\rho = x * \cos\theta + y * \sin\theta$$

Si le pixel aux coordonnées (x_1,y_1) est un pixel blanc alors, on va faire varier θ. Cela va permettre à l'algorithme d'explorer toutes les droites possibles.



Pour chacune des valeurs de ce dernier, il faudra incrementer l'accumulateur aux coordonnées (ρ, θ).
Après avoir parcourus tout les pixels, et en normalisant les valeurs de l'accumulateur, on obtient une image équivalente à :



Ici, les cercles rouges representes le croisements de 2 lignes.

A ce stade de l'algorithme, on a un accumulateur contenant des valeurs plus ou moins importantes representant le nombre de pixels blanc rencontré par une droite. Il faut donc utiliser cette matrice pour mettre en surbrillance les lignes de notre image.

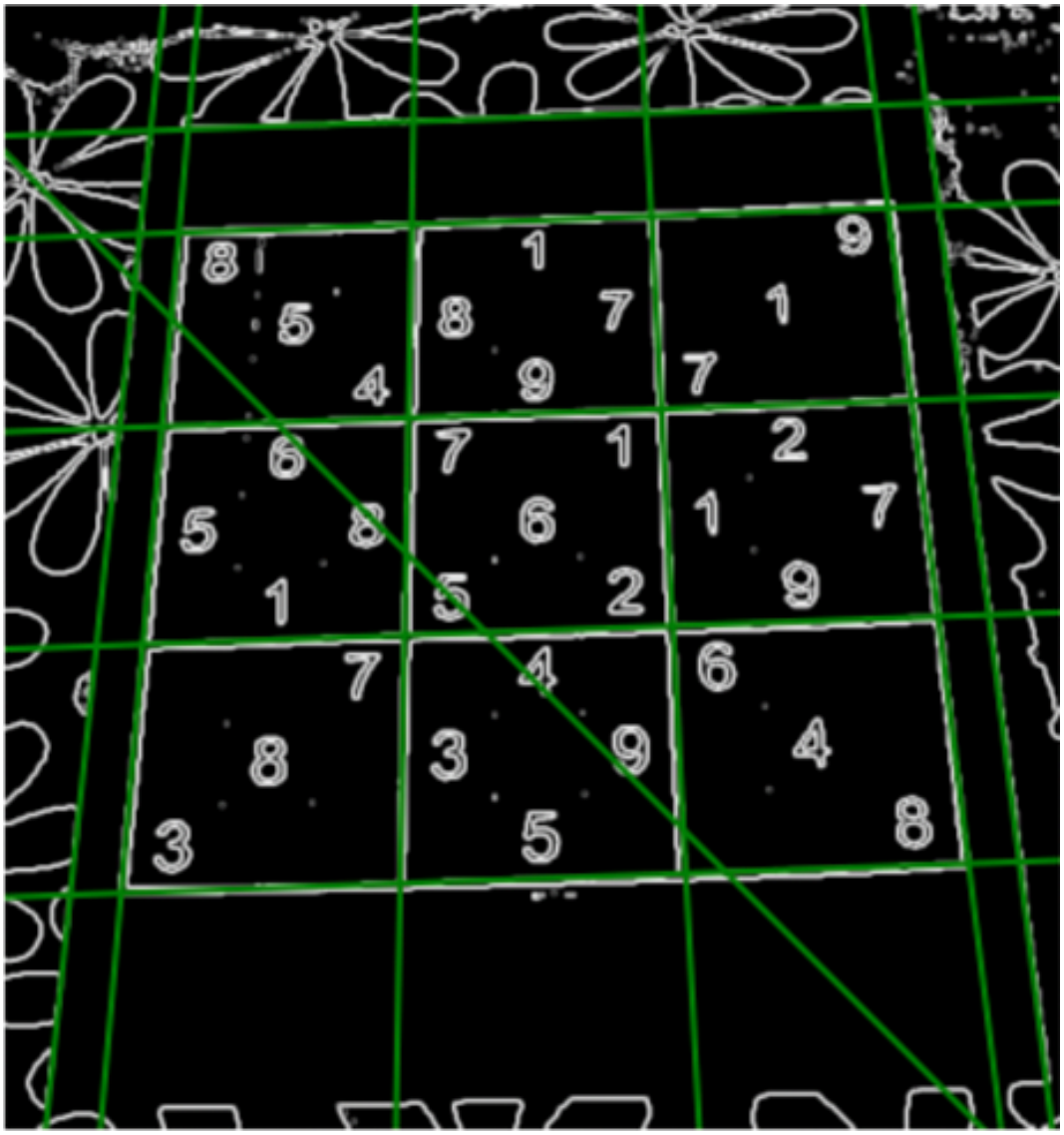
Pour cela, on va decider arbitrairement ou non d'un certain seuil, par exemple, 70% de la valeur maximum de notre matrice, puis on va parcourir notre matrice. Ainsi, notre ρ va evoluer entre $-d$ et d et θ entre $-\pi$ et π. Si la valeur de l'accumulateur aux coordonnées (ρ,θ) est superieur à ce seuil alors il y a de fortes chances pour que la droite définir par ρ et θ soit une bordure. De ce fait, on va calculer :

$$x_0 = \rho * \cos\theta \text{ et } y_0 = \rho * \sin\theta$$

Ces deux coordonnées representent un point d'une droite. Or pour pouvoir tracer cette droite, il nous faut au moins un autre point :

$$x_1 = x_0 + (d * \cos\theta) \text{ et } y_1 = y_0 + (d * \sin\theta)$$

Une fois toutes les droites tracées, l'image devrait ressembler à :



Durant cette partie du projet, nous avons rencontré de nombreuses difficultés. Effectivement, malgré plus de 70% du temps de travail total de l'un de nos membres, l'algorithme n'est toujours pas fonctionnel. A l'heure actuelle nous ne connaissons pas encore le problème qui nous empêche de continuer le traitement de l'image convenablement. Ce bloquage explique pourquoi la rotation automatique et le decoupage de l'image en sous image n'est pas encore commencé.

5.1.2 Rotation Automatique

L'objectif maintenant est de redresser automatiquement l'image pour que chaque chiffre soit à l'endroit. Pour cela, il nous faut 2 algorithme :

- Rotation manuelle selon un certatin degré
- Detection de l'angle opitmale de rotation

Actuellement, seul le premier algorithme est complet.

Rotation Manuelle Pour la rotation manuelle de l'image, on utilise une formule afin de trouver les nouvelle coordonnées (x_2, y_2) en fonction des coordonnées actuelles (x_1, y_1). On a donc :

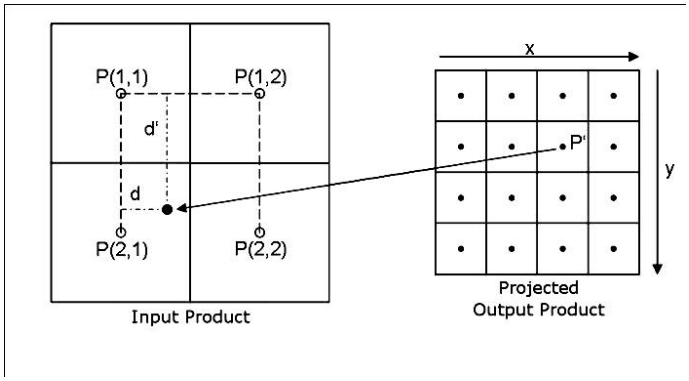
$$x_2 = \cos(\theta) * (x_1 - c_x) + \sin(\theta) * (y_1 - c_y)$$
$$y_2 = \cos(\theta) * (x_1 - c_x) - \sin(\theta) * (y_1 - c_y)$$

Où θ est l'angle de rotation desiré en radian, c_x le centre de l'abscisse et c_y le centre de l'ordonné.

Le problème avec ces formules c'est que le resultat n'est que rarement entier. Ainsi, pour eviter de grosses pertes d'informations, on va interpoler la valeur de ces pixels. Pour cela, on va utiliser l'interpolation bilinéaire.

5.1.3 Interpolation Bilinéaire

L'interpolation bilinéaire est une methode qui permet d'interpoler la valeur d'une fonction a aprtir du resultat des 4 valeur entieres les plus proches. Appliqué à une image, on obtient :



A partir de ce principe, on peut obtenir $f(x, y)$ la valeur du pixel (x,y), où x et y sont issue des formules vu précédement dans la rotation, à partir de x_1 et y_1, les valeurs entières de x et y. Posons $u = x - x_1$ et $t = y - y_1$. On a :

$V_b = (1 - u) * f(i + 1, j) + u * f(i + 1, j + 1)$
$V_a = (1 - u) * f(i, j) + u * f(i, j + 1)$

$f(x, y) = (1 - t) * V_a + t * V_b$

De cette manière, l'image ne subit qu'une perte negligeable lors de la rotation.

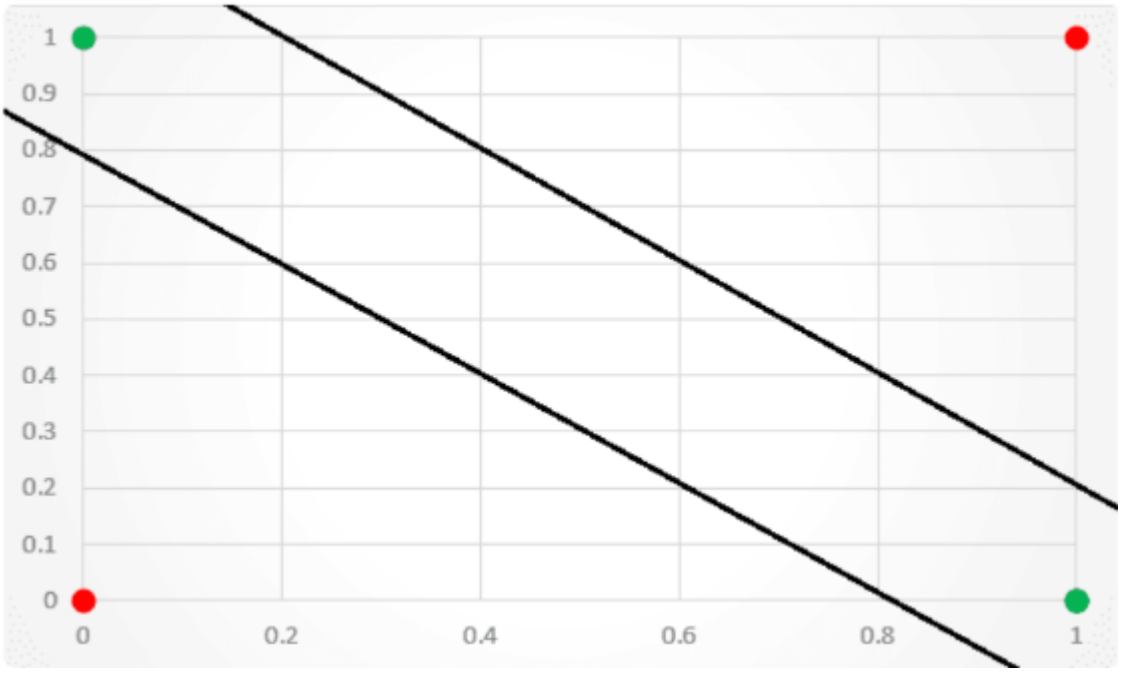
6 Réseau de neurone

6.1 Généralité sur le problème XOR

La problématique est ici qu'il faut construire un réseau de neurones capable de prédire les sorties de portes logiques XOR à partir de deux entrées binaires. Une fonction XOR doit renvoyer une valeur vraie (1) si les deux entrées ne sont pas égales et une valeur fausse (0) si elles sont égales. Tout d'abord, nous pouvons dire que le problème XOR est un problème de classification car on cherche à prédire la valeur d'une variable discrète, c'est à dire une variable qui peut prendre seulement certaines valeurs (dans notre cas 0 ou 1). Nous allons donc utiliser l'apprentissage supervisé.

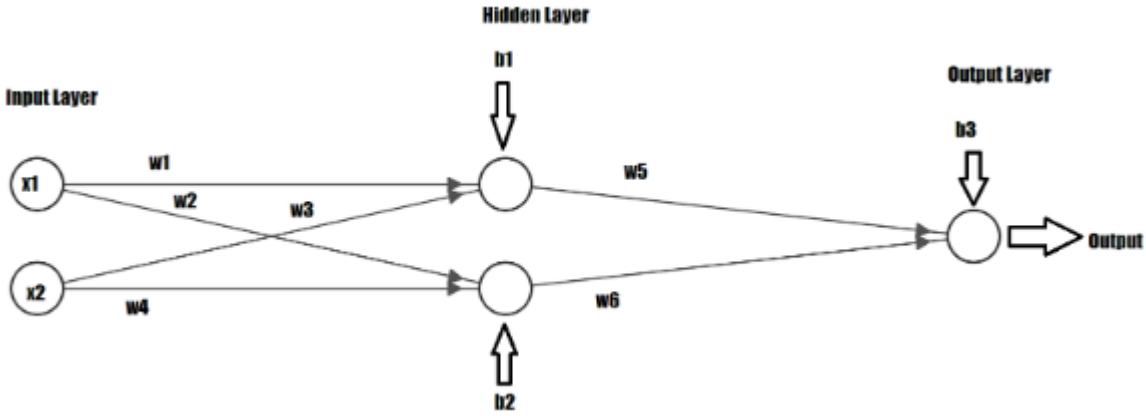
L'objectif en entraînant notre réseau de neurones est de trouver les poids qui permettent de déterminer l'emplacement correct de la ligne de classification, c'est-à-dire la ligne qui sépare les points de données en groupes de classification. Si tous les points de données situés d'un côté de la ligne de classification se voient attribuer la classe 0, tous les autres sont classés 1.

À première vue, comme le problème paraît simple, nous pouvons nous dire que nous n'avons besoin d'utiliser que deux couches (une couche d'entrée et une couche de sortie) pour résoudre ce problème. Cependant, cette structure de réseau de neurones est limitée car elle n'est capable de séparer les points de données que par une seule ligne. Or, la fonction XOR n'est pas linéairement séparable. Cela est particulièrement visible sur l'image ci-dessous. En effet, il n'y a aucun moyen de séparer les prédictions 1 et 0 avec une seule ligne de classification. Nous avons besoin de deux lignes de classifications.



6.2 Structure

Le réseau de neurones est constitué de 3 couches : la couche d'entrée, la couche cachée et la couche de sortie. La couche d'entrée contient deux neurones car la fonction XOR possède deux opérandes. La couche cachée contient deux neurones et la couche de sortie contient un neurone car la fonction XOR ne renvoie qu'un seul résultat. Voici un schéma du réseau de neurone :



Le réseau de neurone possède donc 6 poids et 3 biais qu'il va falloir ajuster pour que le réseau de neurone apprenne la fonction XOR.

6.3 Phase d'apprentissage

La phase d'apprentissage permet au réseau de neurones d'apprendre le lien entre entre les entrées et les sorties des données du DataSet. Ici, nous voulons que le réseau de neurone apprenne la fonction XOR donc le DataSet est :

Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

Input 1 et Input 2 correspondent aux opérandes de la fonction XOR et Output correspond au résultat associé. Pour un certain nombre d'itérations (ici 10 000), nous appliquons pour chaque donnée du DataSet la forward propagation puis la backward propagation.

6.3.1 Forward Propagation

Lors de la forward propagation, les données d'entrée sont transmises au réseau de neurones et se propagent vers l'avant. La couche cachée accepte les données d'entrée, les traite selon la fonction d'activation et passe à la couche suivante jusqu'à la couche de sortie qui renvoie la valeur prédite par le réseau de neurone pour ces donnés d'entrées. L'objectif de la forward propagation est donc de faire prédire au réseau de neurone une valeur de sortie pour telles données d'entrées. La fonction d'activation de notre réseau de neurone est la fonction sigmoïde, soit

$$\varphi(x) = \frac{1}{1+e^{-x}}$$

6.3.2 Backpropagation

Une fois que la forward propagation est réalisée, il faut changer les poids et les biais du réseau de neurones pour se rapprocher au maximum de la bonne solution (valeur réelle). C'est ce que fait la backpropagation. L'idée générale de la backpropagation est que nous voulons savoir l'impact que provoque la légère modification d'un poids ou d'un biais du réseau sur la valeur de sortie du réseau. Nous voulons donc la réponse à la question : Comment la valeur prédite par le réseau est affectée lorsque nous modifions chacun des poids et des biais du réseau ?

Pour obtenir la réponse à cette question et donc ajuster correctement les poids et les biais du réseau, il faut effectuer une descente de gradient. Pour notre réseau de neurone la fonction de coût choisie est l'erreur quadratique moyenne, soit : (Valeur prédite - Valeur réelle)**2. Cette fonction de coût permet de calculer l'erreur entre la valeur prédite et la valeur réelle. L' idée générale de la descente de gradient est de se déplacer dans le sens négatif de la pente de la fonction de coût jusqu'à ce qu'on trouve la valeur minimale de la fonction de coût. Pour faire cela, il faut calculer les dérivées partielles de la fonction de coût par rapport à chaque paramètre (poids et biais), puis se déplacer dans la direction négative de ces dérivées.

Ainsi la formule générale pour ajuster par exemple la valeur du poids w1 est :

$$w1^{+} = w1 - \eta \frac{\partial Cost}{\partial w1}$$

avec : w1+ = la nouvelle valeur du poids w1 n = learning rate (constante) = 0.1 pour notre réseau de neurone

7 Résolveur de sudoku

7.1 Présentation

Premièrement, rappelons le concept de sudoku pour faciliter la compréhension de la suite : La règle est simple. En partant des chiffres déjà inscrits, vous devez remplir la grille de manière à ce que chaque ligne, chaque colonne, chaque carré de 3 sur 3 contiennent une seule fois les chiffres de 1 à 9. Maintenant que les règles sont rappelées, on peut se plonger dans la résolution de celui-ci.

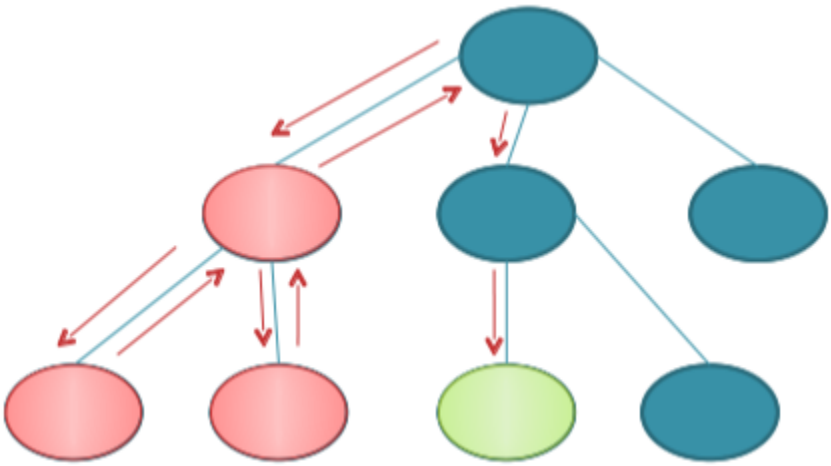
4	1	9	7	8	5	3	6	2
7	2	5	4	6	3	9	1	8
6	3	8	1	9	2	4	5	7
2	9	3	5	7	8	6	4	1
8	7	1	6	3	4	2	9	5
5	4	6	2	1	9	7	8	3
3	5	7	9	4	1	8	2	6
9	8	2	3	5	6	1	7	4
1	6	4	8	2	7	5	3	9

7.2 Le Résolveur de Sudoku

Pour la résolution du sudoku, nous avons décidé d'utiliser le backtracking, ou retour sur trace en français.

Cette méthode consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage. La méthode des essais et erreurs constitue un exemple simple de backtracking. Le terme est surtout utilisé en programmation, où il désigne une stratégie pour trouver des solutions à des problèmes de satisfaction de contraintes.

Le backtracking peut s'illustrer avec la notion d'arbre et de son parcours profondeur avec une contrainte sur les nœuds.



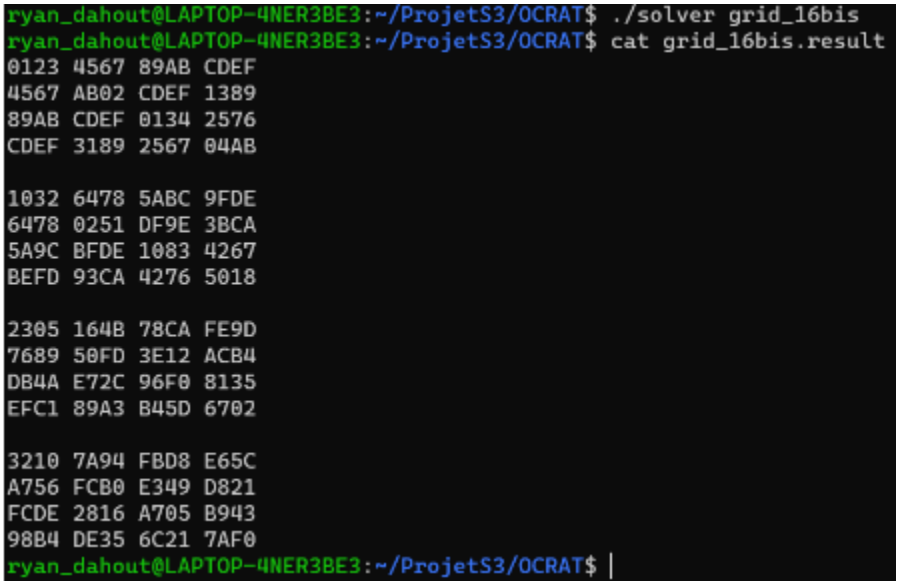
Le principe est de commencer dans le nœud parent puis de descendre sur le premier fils respectant la contrainte. Le nœud dans lequel nous sommes descendus devient le nouveau nœud parent est le processus de descente reprend identiquement. Lorsqu'on tombe sur un nœud ne respectant pas la contrainte, on remonte au nœud parent pour tester ses fils suivants. On trouve la solution quand nous sommes arrivés sur un nœud qui n'a pas de fils, appelé feuille, respectant la contrainte. En revanche, si on atteint la dernière feuille et qu'elle ne respecte pas la contrainte alors il n'existe pas de solution au problème. Dans notre cas, celui de la résolution du sudoku, la contrainte sera que la grille est valide avec la chiffre que l'on ajoute. On va parcourir la matrice, donnée par la grille, lorsqu'on sera sur une case vide, on testera l'ensemble des chiffres de 1 à 9, le premier chiffre respectant la contrainte sera ajouté dans la case puis on testera si la grille reste valide si on continue avec cette valeur.

- Si on arrive à la fin de la grille complétée alors on a trouvé la solution et on la renvoie.
- Sinon, c'est que la valeur a posé problème au bout d'une certaine itération.

Dans ce dernier cas, alors l'algorithme remonte à l'information de cette valeur puis teste la prochaine valeur respectant la contrainte. Ce processus est répété jusqu'à que la bonne valeur soit trouvée.

7.3 Résolveur d'hexadoku

Le résolveur de sudoku étant terminé et opérationnel, nous avons poussé un peu plus loin sa capacité en le modifiant afin qu'il puisse s'appliquer sur des grilles de 16 par 16 accompagnée de l'implémentation de la notation hexadécimale afin de faciliter la lecture. Pour les hexadokus, c'est exactement le même comportement que prend le résolveur. Le fichier texte, quant à lui, est désormais composé de 19 lignes dont 3 lignes vides séparant chaque groupe de 16 cases et de 19 caractères par ligne dont 3 espaces séparant chaque groupe de trois cases. Notre résolveur déduit de lui-même s'il doit résoudre un sudoku ou un hexadoku. Il va ouvrir une première fois le fichier texte, calculer la taille de la première ligne dans le but de déterminer s'il s'agit d'un simple sudoku ou d'un hexadoku. On obtiendra, ainsi, un résolveur de sudoku et d'hexadoku.



7.4 Ce qui a été convenu de faire

Pour cette première soutenance, la consigne était de d'implémenter complètement un résolveur de sudoku en ligne de commande. Cet exécutable s'appelle avec un seul fichier texte contenant une grille de sudoku. Cette grille incomplète illustre les cases vides (chiffres manquants) par des points. Elle est composée de 11 lignes dont 2 lignes vides séparant chaque groupe de 9 cases et de 11 caractères par ligne dont 2 espaces séparant chaque groupe de trois cases. Le résolveur devra donc créer un nouveau fichier texte avec le même nom que le paramètre suivi de l'extension ".result" représentant le sudoku résolu. Ce nouveau fichier aura exactement la même forme avec une seule différence. Les points seront remplacés par les chiffres respectant les règles du sudoku.

7.5 Ce qui est envisagé pour la soutenance finale

Le résolveur de sudoku est terminé et opérationnel. Néanmoins, on pourrait pousser un peu plus loin sa capacité en le modifiant afin qu'il puisse s'appliquer sur des grilles de 16 par 16 accompagnée de l'implémentation de la notation hexadécimale afin de faciliter la lecture. On obtiendra, ainsi, un résolveur d'hexadoku.

8 Avancée générale

	1ere soutenance	Soutenance finale
Prétraitement	90%	Terminé
Rotation	Terminé	Terminé
Segmentation	80%	Terminé
Détéction	Terminé	Terminé
Réseau de neurone (XOR)	Terminé	Terminé
Réseau de neurone (reconnaissance des caractères)	Non commencé	Terminé
Sudoku	Terminé	Terminé
Interface graphique	Non commencé	Terminé

9 Conclusion

Nous sommes satisfaits de nos avancées jusqu'ici. Tous les éléments demandés pour la première soutenance, c'est à dire le chargement d'une image, suppression des couleurs, la rotation manuelle de l'image, détection de la grille et de la position des cases, découpage de l'image, l'algorithme de résolution d'un sudoku et le réseau de neurone capable d'apprendre la fonction OU EXCLUSIF sont présents. De plus, nous avons pu prendre de l'avance dans la partie du Prétraitement. Également, la résolution d'hexadoku (bonus) a été réalisée.