OCR

Valentin.Lucotte
Amine.Latti
Guillaume.Wantiez
Edouard.Disaro

Octobre 2021

Table des matières

1 Introduction

1.1 Sujet

Pour cette année le projet de S3 est de réaliser un programme qui fait la résolution d'une grille de sudoku. Cependant, le paramètre d'entrée n'est pas une chaîne de caractère mais un fichier qui contient cette chaîne. Ce fichier sera remplit via la détection d'une image de sudoku qui détectera la grille, les cases, et les chiffres à l'intérieur.
Une fois la résolution de la grille terminée, la nouvelle grille est écrite dans un nouveau fichier qui servira à faire une fonction qui affiche de manière sympathique la grille résolue. Le groupe est constitue de 4 personnes.

Membres du groupe	
Chef de projet	Valentin Lucotte
Membre	Amine Latti
Membre	Guillaume Wantiez
Membre	Edouard Disaro

1.2 Partage des taches

Répartition des tâches	
Traitement de l'image	Valentin Amine
Segmentation	Amine et Valentin
Réseau de neurone	Guillaume
Résolution de la grille de sudoku	Edouard Valentin

2 Traitement de l'image

Pour le traitement de l'image j'ai réalise plusieurs filtre pour facilite le passage de la segmentation au réseau de neurone :

1. Grayscale
2. Gamma
3. Contraste
4. une fonction de la librairie SDL : rotozoomSurface()

2.1 Grayscale

Pour la fonction Grayscale elle n'a pas été difficile à implémenter car nous l'avons vu en TP. Le principe est simple, il suffit de parcourir tout les pixels (Uint 32) et de récupérer leurs valeurs rouge, verte et bleue(toutes en Uint 8). Ensuite, on calcule leurs moyenne puis on modifie le pixel que l'on a récupéré via la fonction :
SDL_MapRGB(image,rouge,vert,bleue) qui modifie directement le pixel ou bien on peut effectuer des décalages vers la gauche des valeurs de pixel.

Image sans filtre

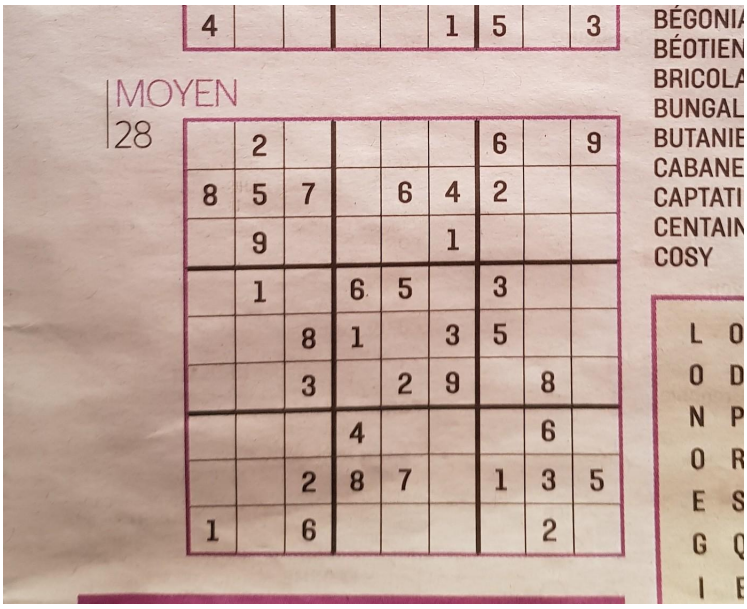
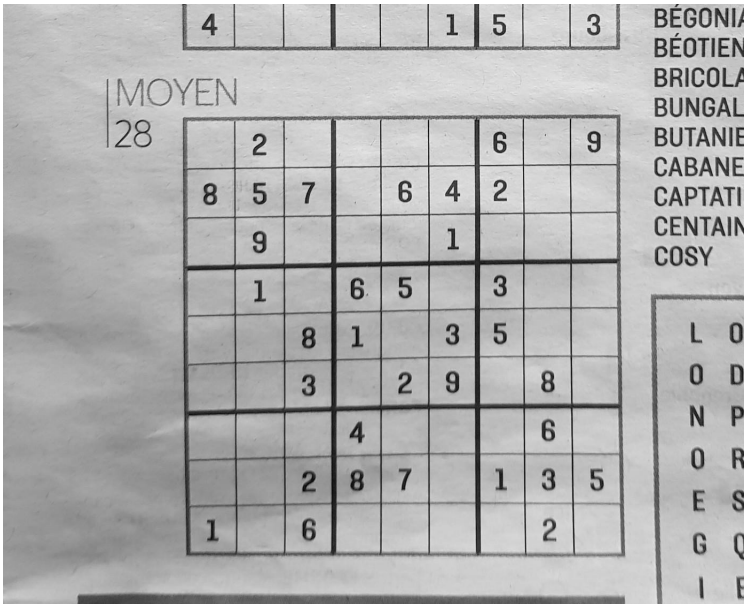


Image avec filtre



2.2 Gamma

La prochaine étape est de faciliter le passage au contraste de l'image pour ce faire on modifie chaque pixel de l'image en augmentant le niveau de blanc pour qu'il atteigne un niveau au alentours 235-255, le but étant de restreindre et de délimiter le plus possible la démarcation entre le noir et le blanc pour réaliser ce changement de pixel j'ai utilisé une fonction qui va faire tendre chaque couleurs du pixel vers 255 sans qu'il ne dépasse ce seuil : avec 'c' la couleurs on a :

$$Output = 255 * (\frac{c}{255})^{0.1}$$

Image avec Grayscale

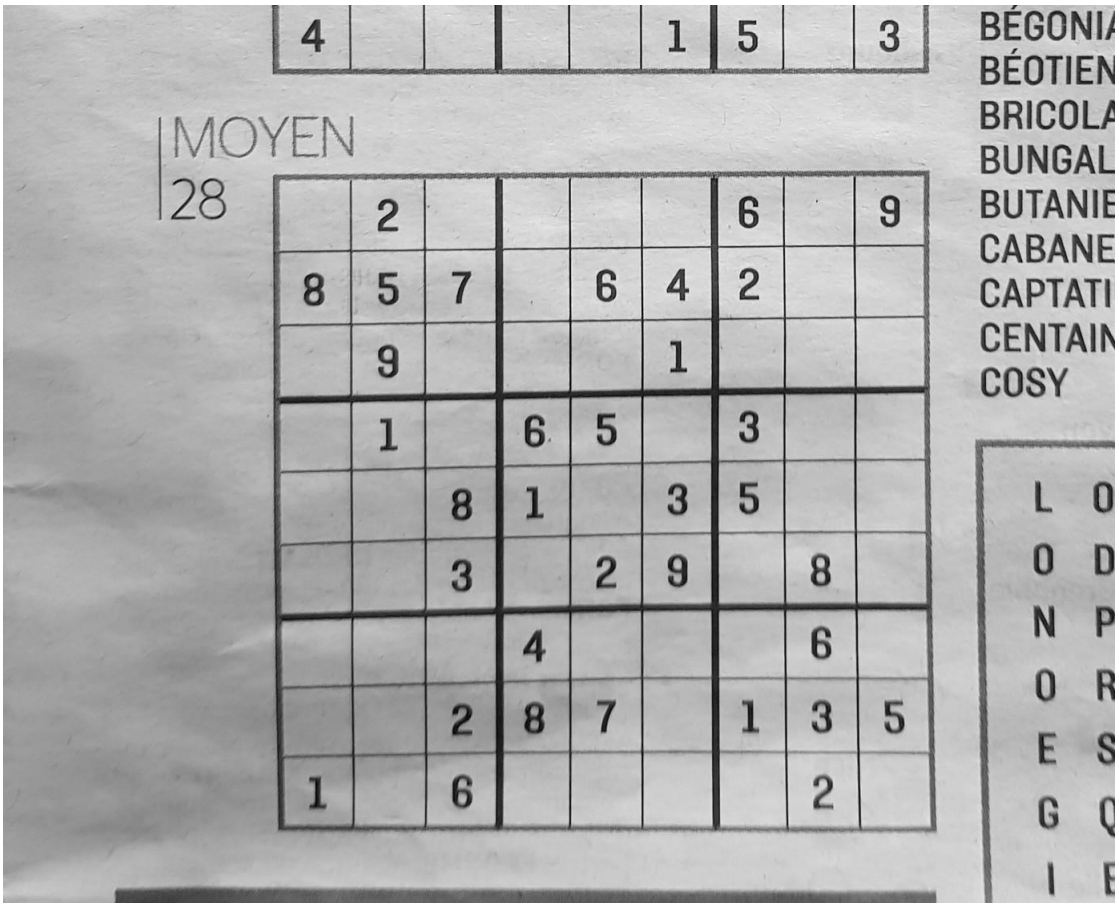
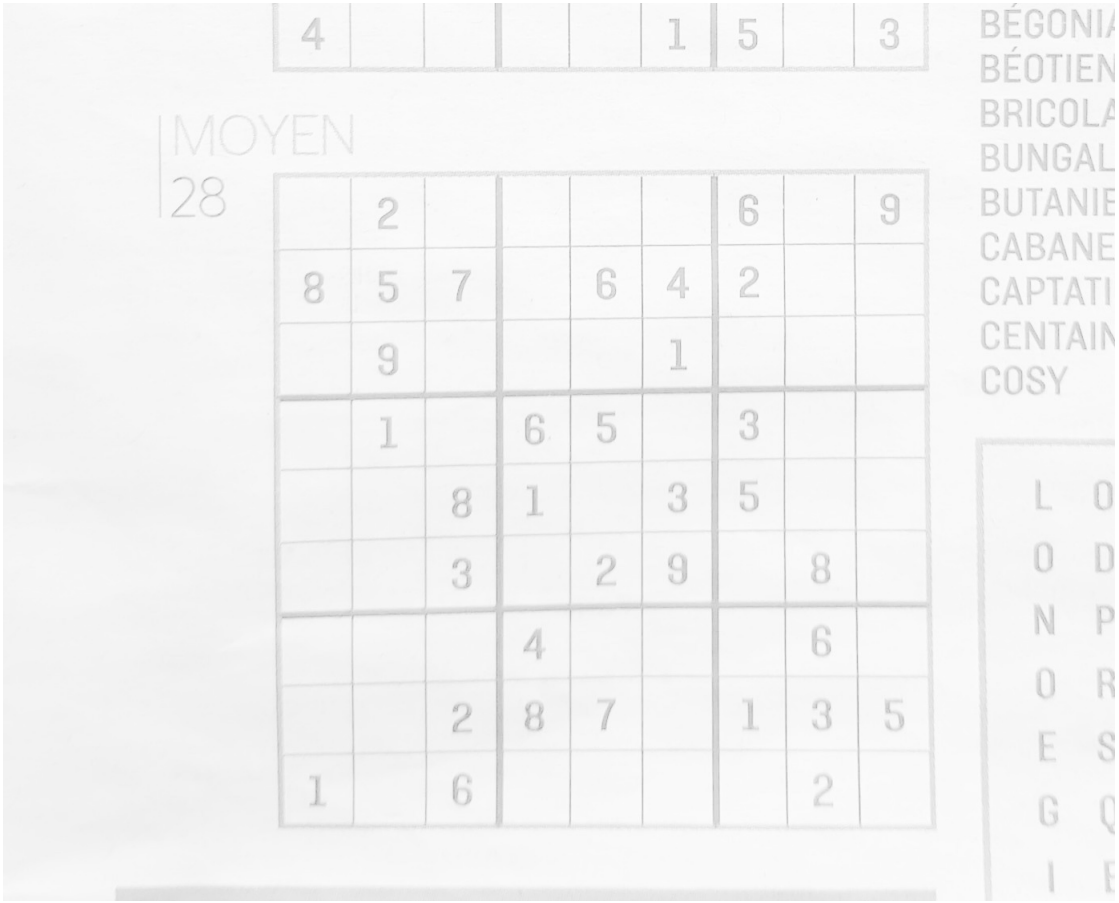


Image avec Gamma



2.3 Contraste

Maintenant que l'image bien préparé au pour la segmentation que nous avons apporté des conditions très serrés pour distinguer la grille de manière claire, la prochaine étape est d'appliquer un filtre noir et blanc avec une condition pour passer du noir au blanc très elevé dans notre situation j'ai choisi de mettre un pixel dit "gris" en blanc si l'une de ses valeurs rgb est dépasse la valeur 240.

Image avec Gamma

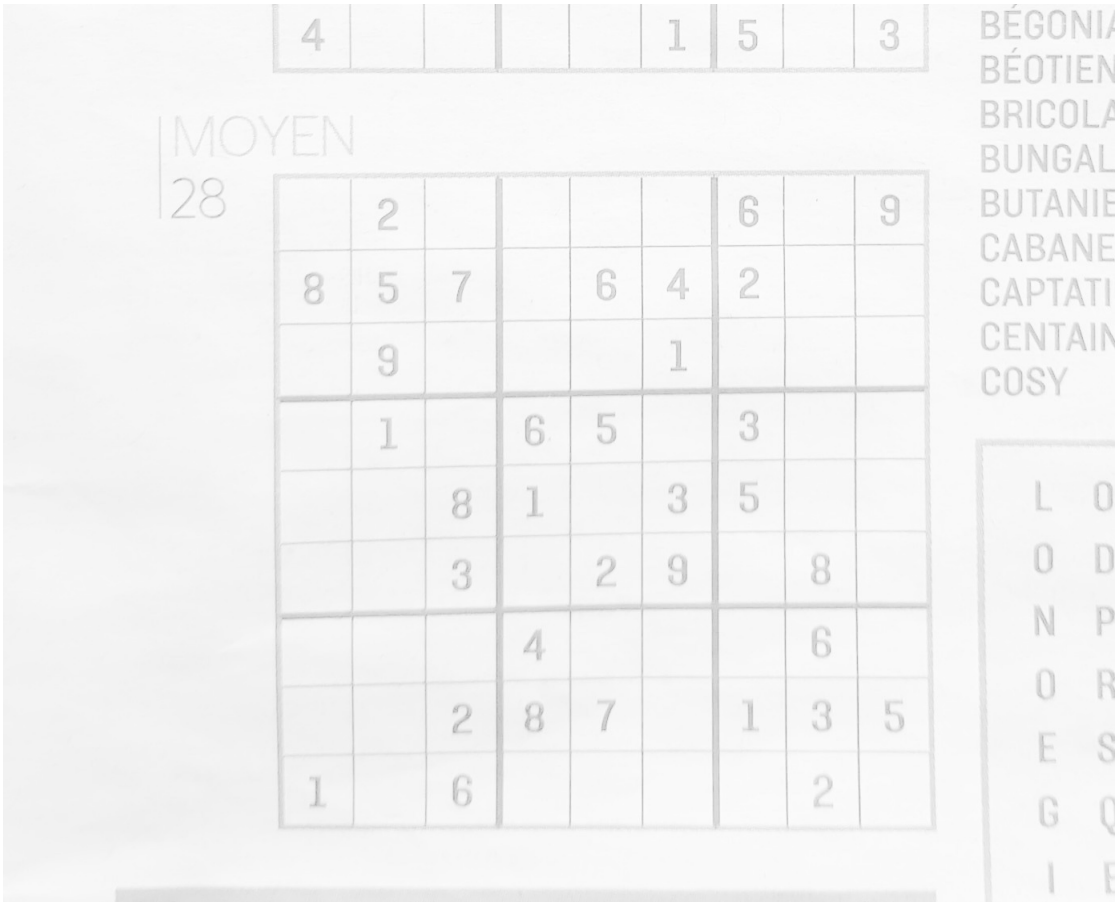
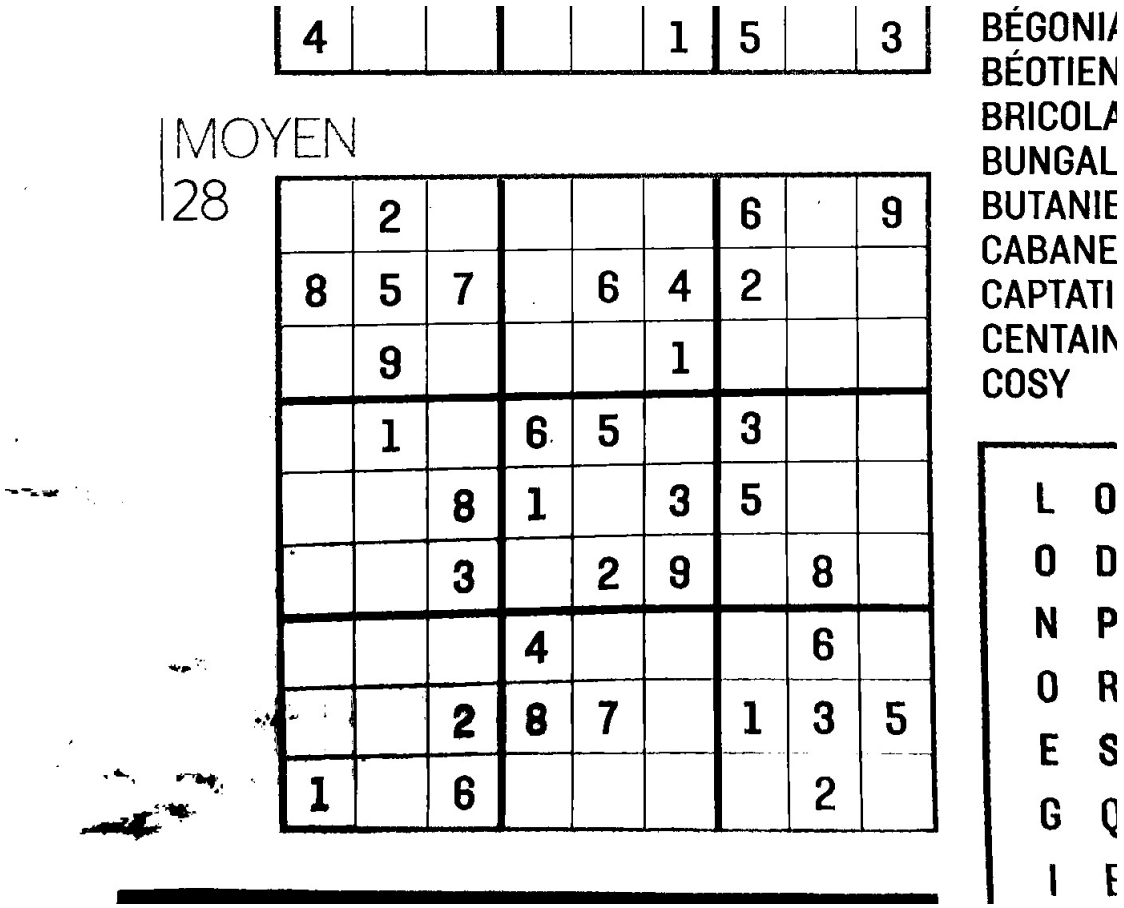


Image avec Contraste



2.4 RotozoomSurface

Pour finir la rotation de l'image se fait avec une fonction qui se situe dans la librairie SDL. elle s'appelle de cette manière :
Zoom = rotozoomSurface(image,angle,zoom)
Zoom : la surface sur laquelle sera "collé" le zoom
image : l'image qui va être zoomée et/ou tourne
angle : l'image va tourner de cet angle
zoom : l'image va scale xZoom c'est un double

Image avec contraste

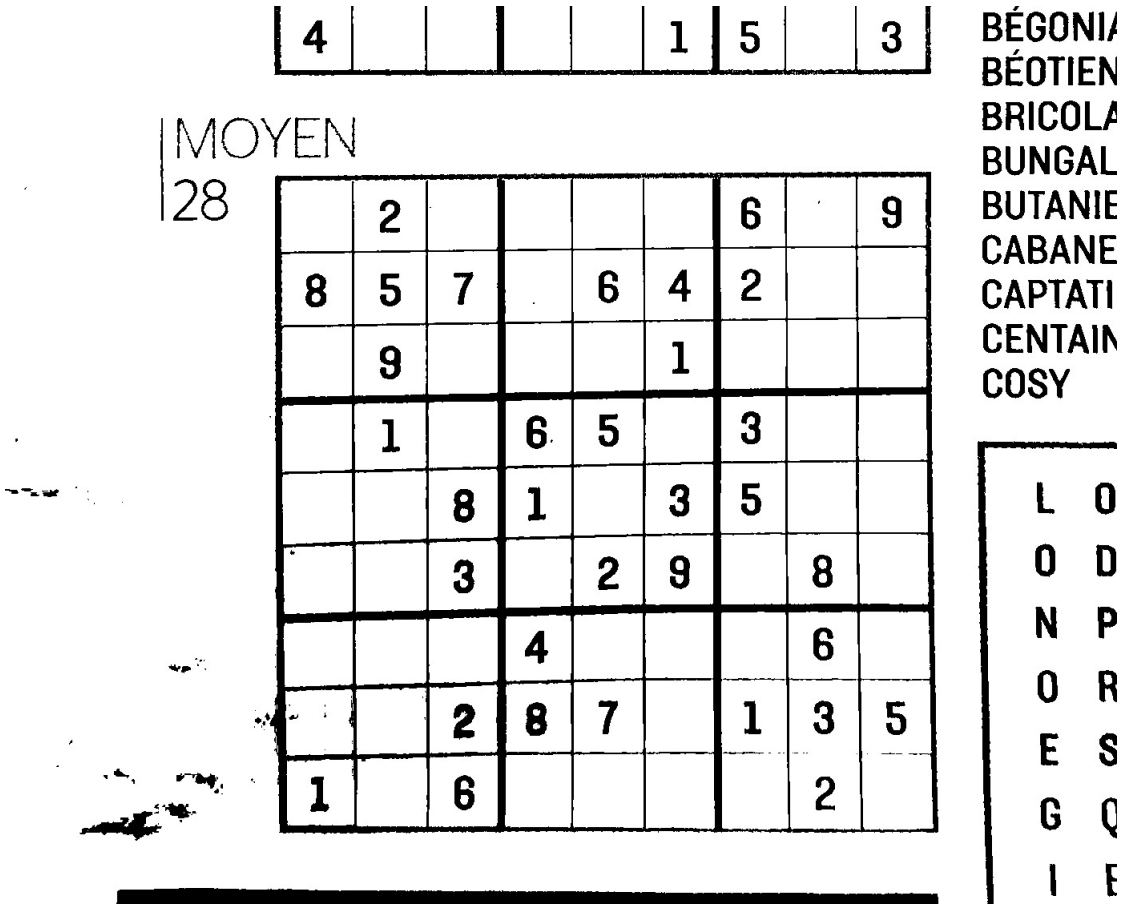


Image avec rotozoom

3 Segmentation

3.1 Principe

La segmentation est un principe permettant de diviser une image en plusieurs sections. Dans notre cas, il s'agit de diviser l'image d'un sudoku en 81 autres images. Chacune de ces images représentant une case du sudoku.

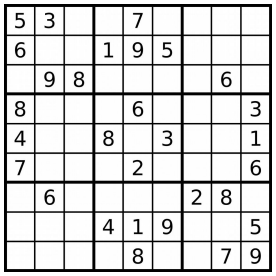
3.2 Problématique

Pour se faire, il faut en premier lieu détecter la grille. Aux premiers abords cela ne parait pas être un grand problème car aux yeux d'un être humain, faire la distinction entre un chiffre et une case semble trivial. Mais pour un ordinateur ce n'est pas la même histoire. En effet un ordinateur ne perçoit que des pixels et ne sait pas faire la différence entre une case et un chiffre. Il faut donc lui instruire des méthodes lui permettant de le faire.

3.3 Solution

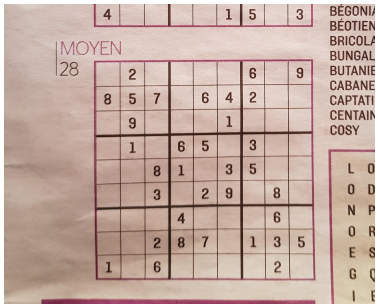
Afin d'apprendre a l'ordinateur a faire la différence entre un une case et un chiffre, il faut tout d'abord se poser la question : "Comment je fais pour différencier une case et un chiffre ?". Après un peu d'introspection, on se rend compte que pour les différencier, une case est un carré tandis qu'un chiffre n'est pas un carré. Il faut donc parcourir l'image et traiter les cas ou un ensemble de pixel noir forme un carré.

En revanche cette méthode la ne marche pas forcement dans tout les cas. En effet, si l'image n'est pas en noir et blanc ou bien si l'image n'est pas droite ou bien si la photo n'est pas exactement droite.



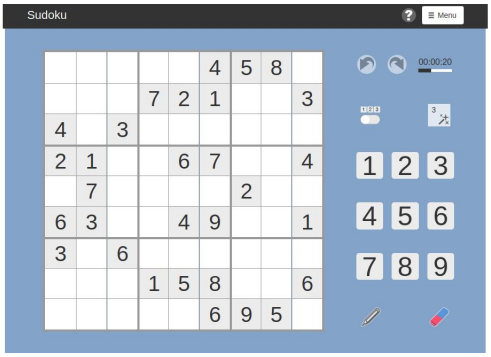
Cas Positif Pour la Méthode

Dans ce cas précis, l'image est parfaite pour notre méthode. L'image est parfaitement droite, centrée et en noir et blanc. On peut donc facilement détecter les cases via la méthode énoncée précédemment.



Cas Négatif Pour la Méthode

Cette image présente plusieurs problèmes pour notre méthode. Pour commencer, elle n'est pas droite, ce qui signifie que les lignes des cases ne sont pas droites. Ensuite l'image n'est pas en noir et blanc et pour finir, la grille de sudoku n'est pas au centre de l'image.



Cas Négatif Pour la Méthode

Ici, la grille n'est pas centrée et l'image n'est pas en noir et blanc. De plus d'autres cases sont présentes et peuvent corrompre nos résultats.

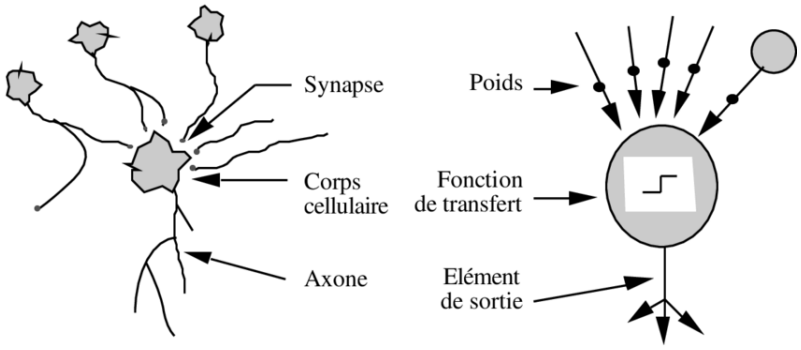
3.4 Que Faire ?

Traiter chacun de ces cas individuellement ne semblait pas être une bonne solution, car elle serait trop coûteuse en temps et en ressources. Nous avons donc décidé de traiter les images en amont de la segmentation afin de toutes les avoir sous le même format que celui du premier exemple. De cette façon, nous pouvons donc simplifier la segmentation et limiter les erreurs. En effet, l'image sera uniquement constituée de la grille en noir et blanc. Ce qui nous permet de simplement récupérer les cases de la grille en découpant l'image en 81 sections de taille équivalente.

4 Neural Network

4.1 Passage du biologique à l'informatique

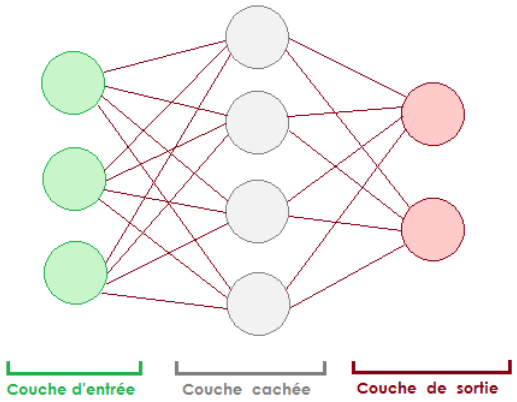
Le fonctionnement d'un réseau de neurone se base sur la modélisation d'un neurone humain, en effet en biologie ce sont des structures permettant de faire circuler l'information électrique de proche en proche entre chaque neurone dans le cerveau. Il est constitué de trois parties : Les synapses, le corps cellulaire et les axones, on modélisera par la suite ces trois éléments. Ainsi, transformées, les synapses deviennent les poids, le corps cellulaire sera la fonction de transfert et enfin les axones seront les sorties.



Modélisation d'un neurone via une fonction de transfert

Les réseaux de neurones, en informatique, sont un assemblage de plusieurs lignes contenant les noeuds (neurones) ces lignes sont appelées "couches". Elles peuvent être des couches externes ou des couches internes, et chaque couche possède un nombre défini de neurone. Chaque noeud (neurone) influe les uns sur les autres et chaque couche les unes entre les autres, et permettent le calcul d'une sortie en fonction d'une entrée. Il existe plusieurs types de structures neuronales différentes en fonction de la tâche que doit accomplir le réseau.

Toute couche interne est dite "cachée", et les couche externes représentent soit les entrées soit les sorties.



Composition d'un réseau de neurone

4.2 Apprentissage d'un réseau de neurone

Un réseau de neurone apprend grâce au calcul d'une courbe du taux d'erreur, cette courbe se calcule grâce à l'actualisation des différentes valeurs contenues dans nos neurones : on appelle cette actualisation "Propagation", la valeur des poids des neurones de la prochaine couche est donnée par l'équation suivante :

$$v = \sum (w_1 * va_1 + b) \tag{1}$$

1. w : Poids
2. a : Valeur d'activation
3. b : Biais
4. v : Nouvelle valeur du neurone

Cette propagation doit se faire dans les deux sens, le réseau à l'essai 1 n'est pas assez entraîné pour donner le résultat voulu donc on doit rappeler le calcul et changer les poids des neurones jusqu'à obtenir le bon résultat. On appelle ça la rétro-propagation et elle est donnée par le calcul de la fonction dite de gradient descendant suivante (on suppose qu'on se place dans le neurone i de la couche d'entrée) :

$$\frac{\partial Error}{\partial wh} = (t - y_0) * y_0 * (1 - y_0) * w_0 * x_0 * (1 - x_0) * x_h \tag{2}$$

1. Wi : Poids du i-ème neurone
2. Yi : résultat obtenu du i-ème neurone
3. Xi : entrée du i-ème neurone
4. Xh : entrée du neurone de la couche cachée
5. t : résultat attendu

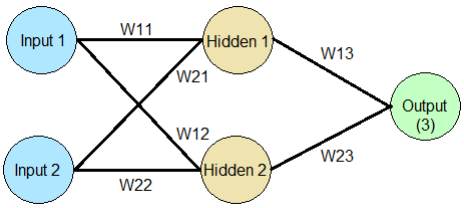
En rappelant ce principe en boucle on affine les résultats du neurone jusqu'à ce que celui ci calcule les sorties correctes en fonctions des entrées, une fois les bons taux obtenus il faut traduire les flottants en 0 ou 1 via une fonction d'activation appelée sigmoïde donnée par :

$$S(x) = 1/(1 + e^{-x}) \tag{3}$$

1. x : valeur en flottant.

4.3 Réseau de neurone XOR

L'objectif de cette soutenance est de créer un réseau de neurone capable d'apprendre le XOR (Ou-exclusif), il nous faudra donc deux entrées et une sortie. Le réseau aura 2 neurones sur sa couche externe d'entrée et 1 neurone sur sa couche externe de sortie, on choisira une couche cachée de deux noeuds pour traiter les données. En appliquant les principes de propagation et de rétro-propagation via des matrices et des pointeurs, j'ai réussi à obtenir un réseau fonctionnel, son schéma est le suivant :



Mon réseau de neurone XOR

5 Résolution de la grille

5.1 Règle du jeu

Une grille de sudoku est dite résolue lorsque toutes ses lignes, colonnes et carrés de taille 3*3 sont remplis de sorte qu'il n'y a pas plus d'une fois le même chiffre dans la ligne/colonne ou carré 3*3

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

Exemple d'une grille de sudoku résolue

5.2 Étape de la résolution

Pour se faire, j'ai tout d'abord implémenté une fonction qui vérifie si un nombre est déjà présent ou non dans la ligne/colonne/carré.

Ensuite à l'aide de cette fonction j'ai conçu le programme qui résout la grille colonne par colonne et ligne par ligne en calculant toutes les possibilités. Si la grille n'est pas solvable, le programme l'indiquera.

5.3 Lire dans un fichier

Pour lire dans un fichier j'ai utilisé la fonction :
fopen(nom_du_fichier,"r") avec "r" pour "read" pour créer un flux. J'ai ensuite parcouru le fichier avec une boucle qui se stop lorsque la fin du fichier est atteinte, c'est-à-dire que je prend caractère par caractère avec la commande :
fgetc("flux") qui prend les caractère 1 à 1 et qui le compare a EOF (End Of File). Un problème a subsisté, avec fgetc() je récupère un char alors que je veux un int dans ma grille je me suis donc servi de la table ASCII et j'ai remarqué que en soustrayant le caractère et '0' j'obtiens le int de la table ASCII : ma conversion est donc faite.

5.4 Écrire dans un fichier

Pour écrire dans un fichier j'utilise la même commande que pour lire c'est-a-dire fopen() mais a la place de "r" je met "w" pour "write" et pour l'extension j'ai ajoute ".result" au fichier de base. Je procède de la même manière que pour lire mais avec la fonction fputc() qui permet d'écrire un caractère dans un fichier.
Pour pouvoir afficher un résultat ergonomique j'ai rajouté des espaces et retour à la ligne supplémentaire pour donner l'effet d'une grille.