

# Rapport De Soutenance

## OCR

Clément FABIEN - Grégoire GENET  
Gabriel CALVENTE - Cédric DAMAIS

Prépa S3  
Novembre 2022 - Décembre 2022

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Présentation du Projet . . . . .	4
1.2	Avancement du projet . . . . .	5
1.3	Attentes . . . . .	6
1.4	Répartition . . . . .	6
<b>2</b>	<b>Rotation de l'image</b>	<b>7</b>
2.1	Rotation Manuelle . . . . .	7
2.1.1	Exatraction des Matrices Unicolonnes . .	7
2.1.2	Matrice De rotation . . . . .	8
2.2	Hough Transform . . . . .	9
2.2.1	Hough : Detection de Ligne . . . . .	9
2.2.2	Algorithme Appliquer . . . . .	10
2.2.3	Meilleur Parametre . . . . .	11
2.3	Rotation Automatique . . . . .	12
<b>3</b>	<b>Binarisation</b>	<b>13</b>
3.0.1	Augmentation Des contrastes . . . . .	13
3.0.2	Algorithme De Otsu . . . . .	13
<b>4</b>	<b>Résoudre la grille</b>	<b>14</b>
4.0.1	Resolution en BackTracking . . . . .	14
<b>5</b>	<b>Filtres</b>	<b>15</b>
5.1	Grayscale . . . . .	15
5.2	Filtre De Gauss . . . . .	17

<b>6</b>	<b>Détection Des Bords</b>	<b>19</b>
6.1	Algorithme De Sobel . . . . .	19
6.2	Algorithme De Canny . . . . .	20
<b>7</b>	<b>Réseau de neurones</b>	<b>22</b>
7.1	Documentation . . . . .	23
7.2	L'objectif de ce premier travail . . . . .	23
7.3	Les challenges . . . . .	23
7.4	Implémentation . . . . .	24
7.5	Sauvegarde des données après entraînement . . .	25
7.6	Améliorations prochaines . . . . .	25
<b>8</b>	<b>Conclusion</b>	<b>26</b>

## 1 Introduction

En cette seconde année de classe préparatoire à l'EPITA, nous sommes de nouveau amené à réaliser un projet en groupe, afin de nous apprendre les bases du travail en groupe, de la recherche individuelle et surtout de nous apprendre à ne pas baisser les bras malgré les difficultés. Ce projet consiste à délivrer sous une date donnée, ce qui est appelé un OCR (Optical Character Recognition), permettant de résoudre un sudoku à partir d'une photo de ce dernier. Notre groupe de projet est constitué de quatre personnes : Calvente Gabriel, Damais Cédric, Genet Grégoire et Fabien Clément, ce dernier étant le chef de projet. Ainsi, lors de ce rapport, nous vous expliquerons en quoi consiste concrètement le projet au global, son état d'avancement, les aspects techniques liés au projet, les difficultés rencontrées, et les attentes quant à notre rendu final.

## 1.1 Présentation du Projet

Le projet de ce troisième semestre à l'EPITA est de faire un OCR (Optical Character Recognition) permettant de résoudre un sudoku à partir d'une photo de ce dernier. Pour ce faire, nous devons effectuer un programme compilable dans le langage C, qui prend en paramètre une photo de sudoku, et qui renvoie la même photo avec la grille de sudoku résolue. Ainsi, notre programme terminé contiendra les éléments suivants : Une partie preprocessing, avec la binarisation de l'image, la rotation, et tout ce qui permettrait d'utiliser optimalement l'image donnée. Ensuite, une partie sur la détection des bords de la grille, puis des cases, mais aussi la sauvegarde des 81 cases séparées dans un dossier auxiliaire. Puis un programme de réseau de neurones, qui permettra de reconnaître séparément les numéros dans les cases différentes. Après avoir reconnu les numéros, ainsi que la disposition théorique de la grille de sudoku, il y a ensuite un programme qui résout le sudoku et renvoie la grille résolue. Puis, un dernier programme mettra tout les numéros sur les images des cases vides, puis recollera les morceaux des 81 cases différentes, en reverra enfin, la grille résolue. Ce programme compilé pourra, s'il reste du temps, être utilisé comme une petite application, afin d'être facile d'utilisation et compréhensible.

## 1.2 Avancement du projet

Le projet avance à grand pas depuis un mois, mais quelques difficultés se font ressentir, notamment sur la détection de la grille et des cases, et c'est pour nous la partie la plus difficile. Nous sommes donc majoritairement avancées sur les requis pour la première soutenance, cependant nous ne disposons pas de la détection des cases de la grille ainsi que de la grille elle même. Nous avons cependant la détection des bords de la grille, ainsi que tout les autres requis de la soutenance, c'est à dire la binarisation, la rotation manuelle, le chargement de l'image dans le programme, une aide visuelle pour voir les images modifiées, un programme de résolution de sudoku sous le format demandé, une manipulation de fichiers basique, ainsi qu'un début de base de données de sudoku afin de tester nos différents programme, puis pour finir un réseau de neurone fonctionnel sachant calculer un XOR, ainsi qu'une manipulation de fichier afin de sauvegarder différents poids et liaisons entre les différents neurones de ce réseau. La majorité des éléments demandés pour cette soutenance sont effectués, mais à cause de la difficulté rencontrées avec la détection des cases et de la grille, nous n'avons pas pu faire. Ainsi, notre prochain objectif est de terminer cela rapidement.

### 1.3 Attentes

Pour la suite, après la première soutenance, nous voulons avoir terminé tout notre projet, et avoir même la création d'une petite application permettant de prendre une photo d'un sudoku, et rendant une image de cette grille résolue, afin d'avoir une facilité d'utilisation agrandie, ainsi qu'une exécution plus rapide et accrue.

Ainsi toutes les attentes pour la seconde soutenance devraient être satisfaites.

### 1.4 Répartition

Pour ce qui est de ce projet, nous nous sommes donnés chacun des tâches différentes, afin de maximiser le temps de travail. Ainsi, Gabriel s'est occupé de faire le concept de notre réseau de neurone, Cédric et Grégoire se sont occupés du preprocessing de l'image, c'est à dire le chargement de l'image, la rotation, la binarisation. Cédric à la rotation manuelle, puis début de l'automatique, et Grégoire au chargement de l'image ainsi que la binarisation. Grégoire s'est aussi occupé de la résolution de la grille de sudoku. Finalement, Clément s'est occupé de la détection des bords de la grille. Pour ce qui est de la suite, étant donné que ce n'est pas nécessaire pour la première soutenance, nous ne nous sommes pas penché sur le sujet, et ne savons donc pas encore comment se répartira la suite du projet, à part que Gabriel se penchera sur le sujet du réseau de neurones afin de reconnaître les différents numéros.

## 2 Rotation de l'image

Pour utiliser l'image donnée par l'utilisateur à notre programme, il faut pouvoir la reconnaître en toutes circonstances. Ainsi, il faut retourner l'image, pixels par pixels pour faire en sorte qu'elle soit droite et utilisable par le programme (notamment le réseau de neurones).

### 2.1 Rotation Manuelle

#### 2.1.1 Extraction des Matrices Unicolonnes

La rotation manuelle, c'est à dire faire tourner l'image d'un certain degré. En théorie, c'est une transformation très facile à comprendre et elle sera très utile plus tard pour la rotation automatique. Toute la difficulté était de réussir à bouger chaque pixel de notre image à leur nouvelle bonne position en fonction de l'angle donné. Pour parvenir à faire cette transformation de l'image nous avons considéré les coordonnées (x,y) du pixel dans le tableau comme une matrice unicolonne.

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

### 2.1.2 Matrice De rotation

Nous avons réalisé cette transformation des coordonées pour pouvoir ensuite faciliter le changement de position en fonction d'un angle à l'aide d'une matrice de rotation.

En deux dimensions, les matrices de rotation ont la forme suivante :

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Avec cette matrice de rotation, trouver la nouvelle position ( $x', y'$ ) des pixels n'est qu'une simple multiplication de matrice :

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

Une fois les coordonées ( $x', y'$ ) du pixel trouvés il ne suffit que de le placer à sa nouvelle position. Puis répéter ce processus sur tous les pixels de notre image.

## 2.2 Hough Transform

La transformation de hough etait necessaire pour arriver à faire la detection de ligne et trouver l'angle necessaire pour avoir une image a la bonne position.

### 2.2.1 Hough : Detection de Ligne

Apres la Binarization de l'image nous nous retrouvons avec des milliers de pixels blanc ou noir. Dans ces pixels, se trouve les lignes que l'on cherche a detecter, et la a ete la plus grosse difficulte rencontrer retrouver les pixels qui forment les lignes qui constituent les lignes. On se retrouve donc à chercher un droite qui pour equation :

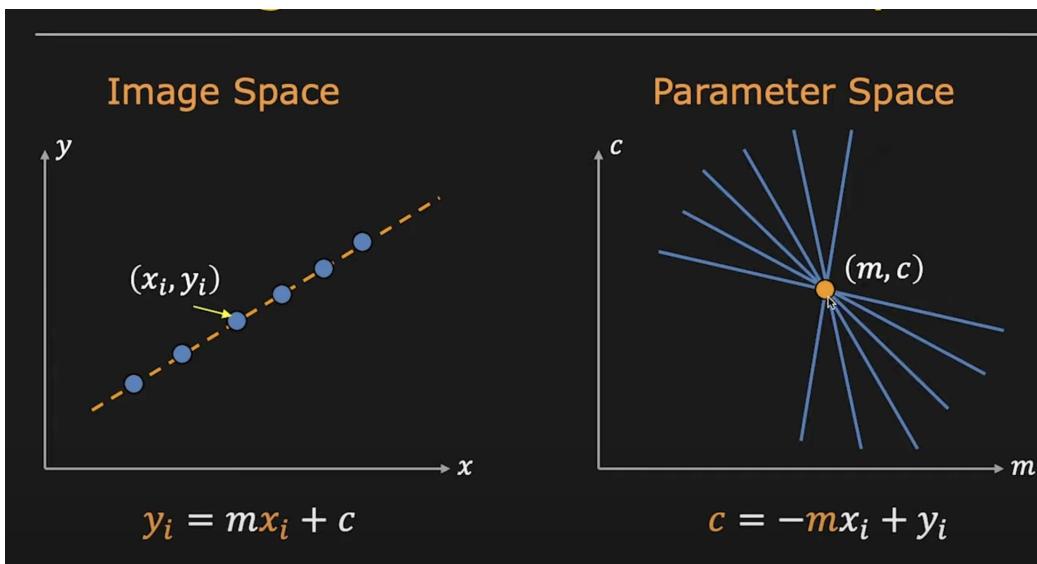
$$y = mx + c$$

dans un tableaux de pixels a des positions  $(x_i, y_i)$ .

Grace à la Transformation de hough on arrive a resoudre se probleme en regardant le probleme dans deux espace different. En representant la droite sur deux plans differents car on a :

$$y_i = mx_i + c \Leftrightarrow c = -mx_i + y_i$$

On a l'espace de l'image dans lequelle on cherche nos lignes, grace a notre equivalence dans les deux plans, un point sur une des lignes dans le plan de l'image est une droite sur notre deuxième plan. Chaque point sur une même lignes du plan image vont cree des droites sur le deuxième plan qui vont etre secante.



### 2.2.2 Algorithme Appliquer

Apres avoir compris le concept de hough l'agoritme appliquer etait assez simple à implementer :

Etape 1 : Cree un Accumulateur ( Tableau initialiser avec des 0 )

$$A_m(m, c)$$

La difficulter rencontrer lors de la manipulation de l'accumulteur est que si le teableau est trop grand des lignes se fusionnent, si il est trop petit, le bruit sur l'image empeche la detection de certaines lignes sur l'image. On donc pris comme taille du tableau la diagonale de l'image binariser.

Etape 2 : Pour tout points  $(x_i, y_i)$  sur les lignes du plan image Si  $(m, c)$  est sur la ligne  $c = -mx_i + y_i$

$$A(m, c) = A(m, c) + 1$$

Etape 3 : Parcourire l'accumulateur pour trouver les maximum qui represente les lignes sur le plan image.

### 2.2.3 Meilleur Parametre

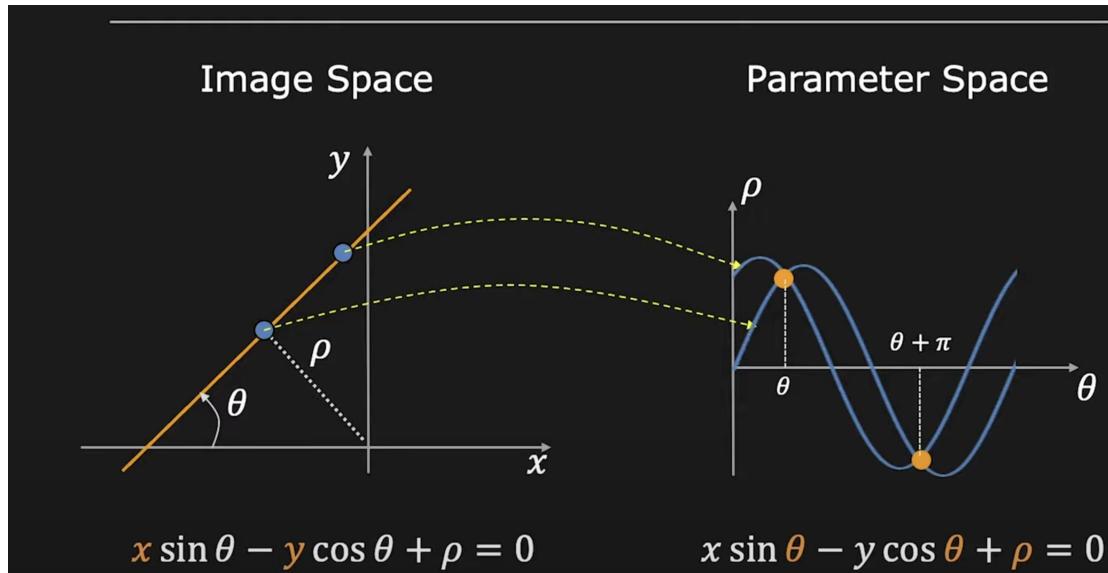
Durant l'implementation de la transformation de hough, on a rencontrer un probleme qui etait le fait que le coefficient directeur :

$$-\infty \leq m \leq \infty$$

Cette contraite nous obligaient à avoir un accumulateur tres grand, et comme nous l'avons precise cela ne permet pas une bonne detection de lignes. De plus un grand accumulateur necessite l'utilisation de beaucoup plus de memoire . Comme solution a se probleme, On utilise les coordones polaires  $(\theta, \rho)$  avec  $\theta$  l'angle crée par la droite et  $\rho$  la distance de la droite à l'origine du plan. En utilisants des equations de passages pour convertir des coordonnées cartésiennes en coordonnées polaires on se retrouve avec :

$$x \sin(\theta) - y \cos(\theta) + \rho = 0 \Leftrightarrow x \sin(\theta) - y \cos(\theta) + \rho = 0$$

Avec ces nouveau parametre l'algorithme reste le même avec un accumulateur  $Am(\theta, \rho)$  :



Une fois que l'accumulateur est remplie on va le parcourir pour trouver l'angle nécessaire à la rotation.

## 2.3 Rotation Automatique

La rotation automatique n'a pas été très difficile à implémenter car, après avoir réalisé la Transformation de Hough qui nous a permis de trouver l'angle nécessaire pour mettre l'image à la bonne position, il n'a suffi que d'extraire cette valeur puis d'appliquer la rotation manuel avec l'angle trouvé après la transformation de hough.

## 3 Binarisation

Afin d'utiliser l'image pour le programme, il faut binariser l'image ; c'est à dire faire en sorte de convertir l'image avec seulement deux couleurs : le blanc et le noir. Pour se faire, il faut utiliser un seuil, à partir duquel chaque pixel ayant une couleur supérieure à ce seuil sera considérée comme blanche, et à l'inverse, comme noire. Ainsi, il existe plusieurs méthodes pour trouver ce seuil adaptés aux différentes images, tel que l'algorithme d'Otsu, de Sauvola, ou bien de Gauss.

Pour ce projet, nous avons décidé d'utiliser la méthode d'Otsu, qui utilise la variance des pixels dans l'image pour trouver le seuil adaptatif à l'image. C'est une méthode qui est abordable et rapide à comprendre, c'est pour cela que nous l'avons choisi.

### 3.0.1 Augmentation Des contrastes

Avant de passer notre image dans l'algorithme d'Otsu, il faut tout d'abord augmenter les contrastes de l'image pour faciliter la recherche d'un seuil. Pour cela, il faut faire l'histogramme des pixels de l'image, puis égaliser cet histogramme. Cela permet de répartir les éléments de l'histogramme en son sein. Puis, changer l'image grâce à l'histogramme égalisé.

### 3.0.2 Algorithme De Otsu

Apes avoir obtenue une image égalisée on applique sur celle ci l'algorithme d'Otsu, pour trouver un seuil à adapté à notre image. Ensuite, binariser l'image en utilisant le seuil trouvé, et enfin, sauvegarder notre image binarisée pour la garder afin de pouvoir l'utiliser dans les programmes suivants.

## 4 Résoudre la grille

Après avoir analysé la grille grâce au réseau de neurone, nous pouvons utiliser toutes les 81 cases afin de créer un document texte avec une représentation de la grille en photo, avec des ‘.’ pour les cases à remplir, et les chiffres autrement.

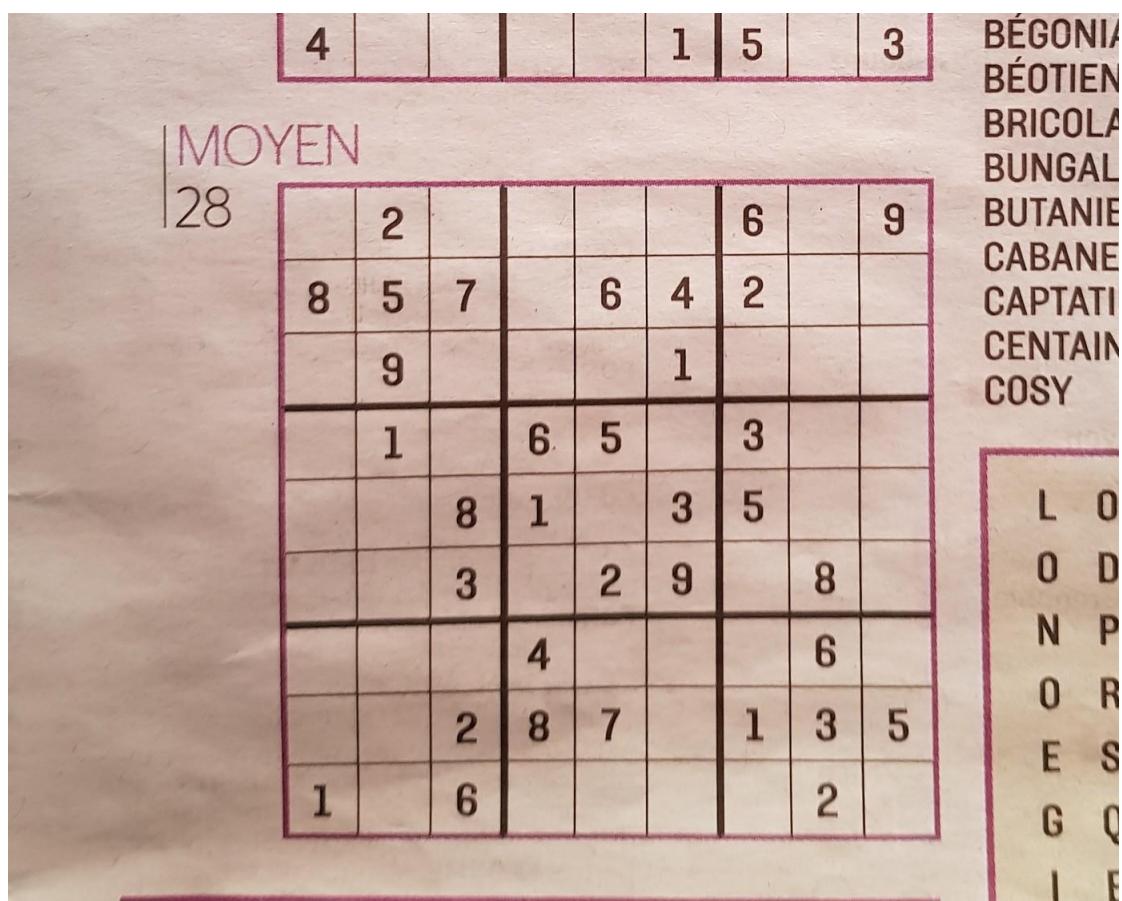
### 4.0.1 Resolution en BackTracking

Puis, avec un algorithme de résolution, grâce à un système de backtrack (récursion successive sur un chemin donné), la grille est résolue et est renvoyée dans un nouveau document texte sans aucun ‘.’. Ainsi, la résolution de la grille n'est pas le challenge de notre projet, étant donné que c'est assez simple, tant pour le programme que pour la création de fichiers.

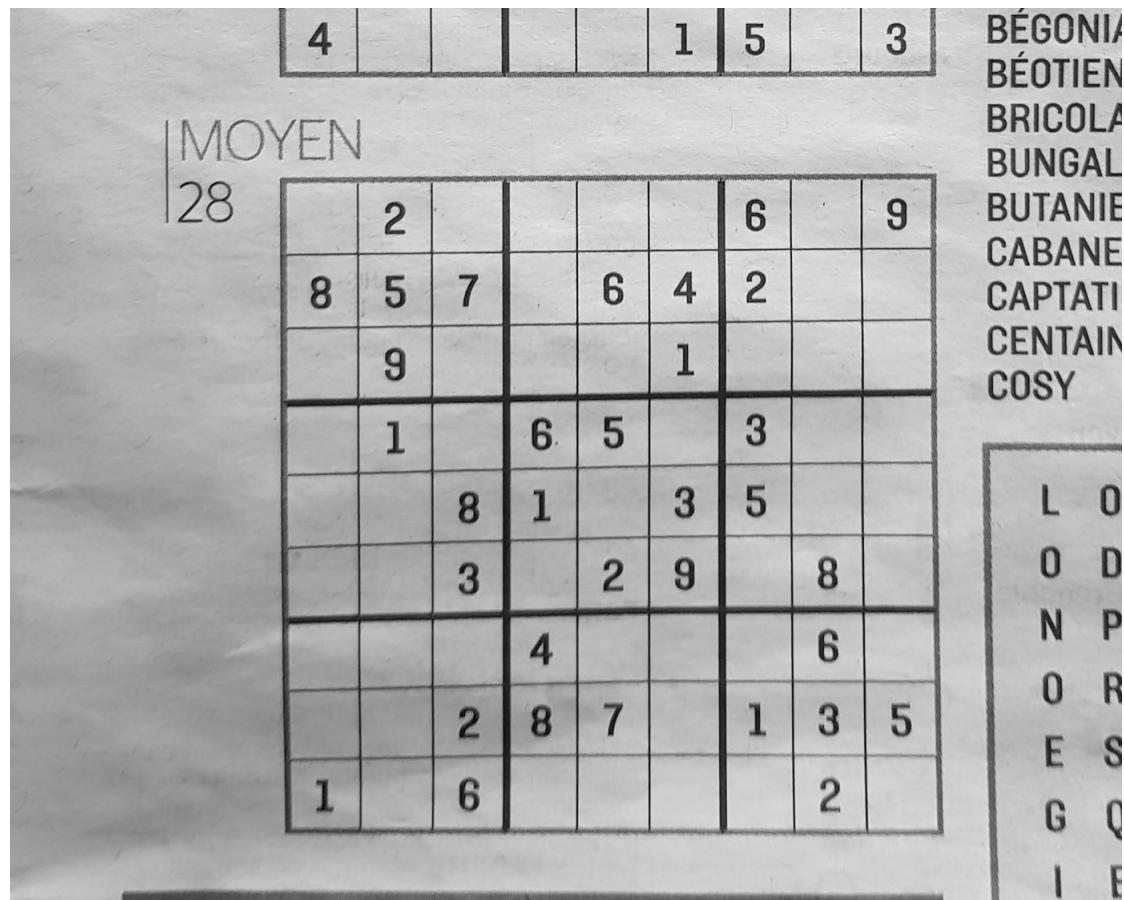
## 5 Filtres

### 5.1 Grayscale

Avant de pouvoir traiter l'image du sudoku en tant que tel, il faut supprimer toutes couleurs afin de pouvoir mieux détecter la grille, les cases et les chiffres. Ainsi nous pouvons appliquer un filtre de gris qui nous permet de transformer l'image entièrement en noir et blanc.



*Image non traitée*



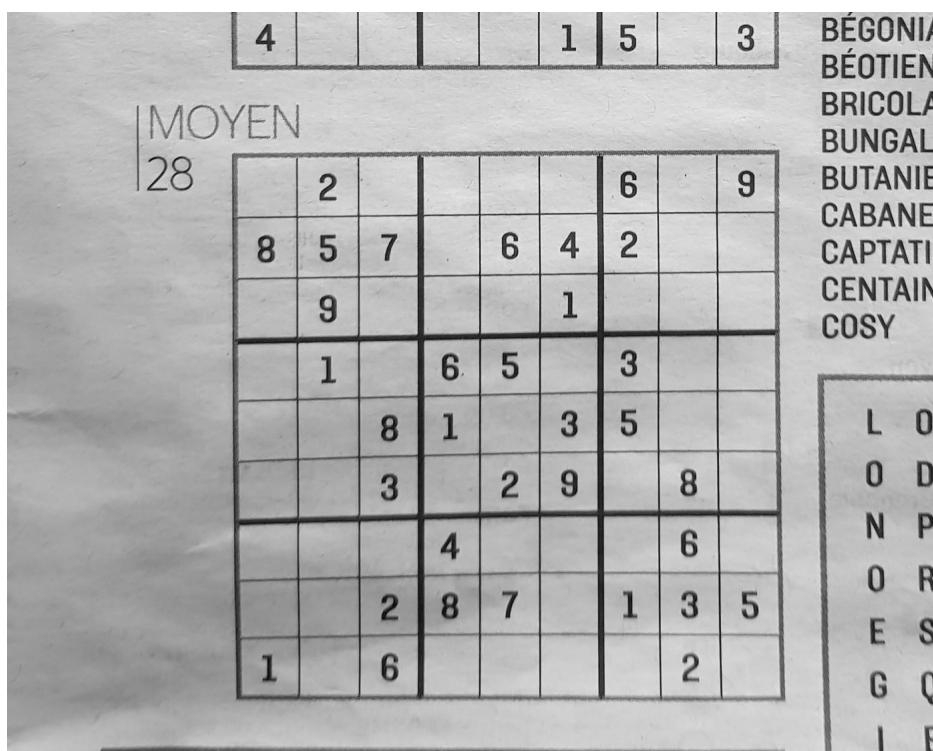
*Image après un filtre de gris*

Pour avoir ce résultat, on applique sur chaque pixel une moyenne des couleurs (RGB) et on le réécrit avec cette nouvelle valeurs.

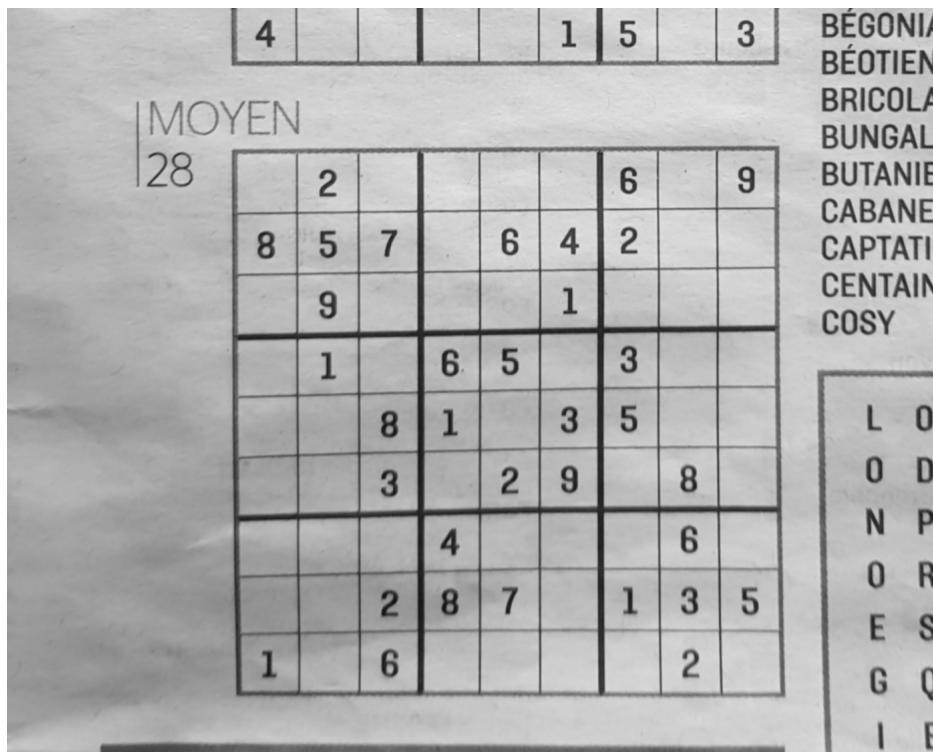
$$average = 0.3 * r + 0.59 * g + 0.11 * b$$

## 5.2 Filtre De Gauss

Après avoir appliqué notre filtre de gris, on doit appliquer un filtre de gauss. Ce filtre permet de réduire le bruit de l'image en la flouttant un tout petit peu. Ce flou permet de rendre la binarisation plus tard meilleur en enlevant certaines tâches visible autrement.



*Image après un filtre de gris*



*Image après un filtre de gris et filtre de Gauss*

La méthode de Gauss consiste à regarder chaque pixel et à faire une moyenne de ses pixels voisins (3x3, 5x5, 7x7, etc...) et appliquer un poids à chacun de ces pixels. Cette matrice qui stocke les poids des pixels s'appelle un kernel. Nous utiliserons un kernel de 5x5 pour notre projet avec les valeurs ci-dessous.

$$kernel = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

## 6 Détection Des Bords

### 6.1 Algorithme De Sobel

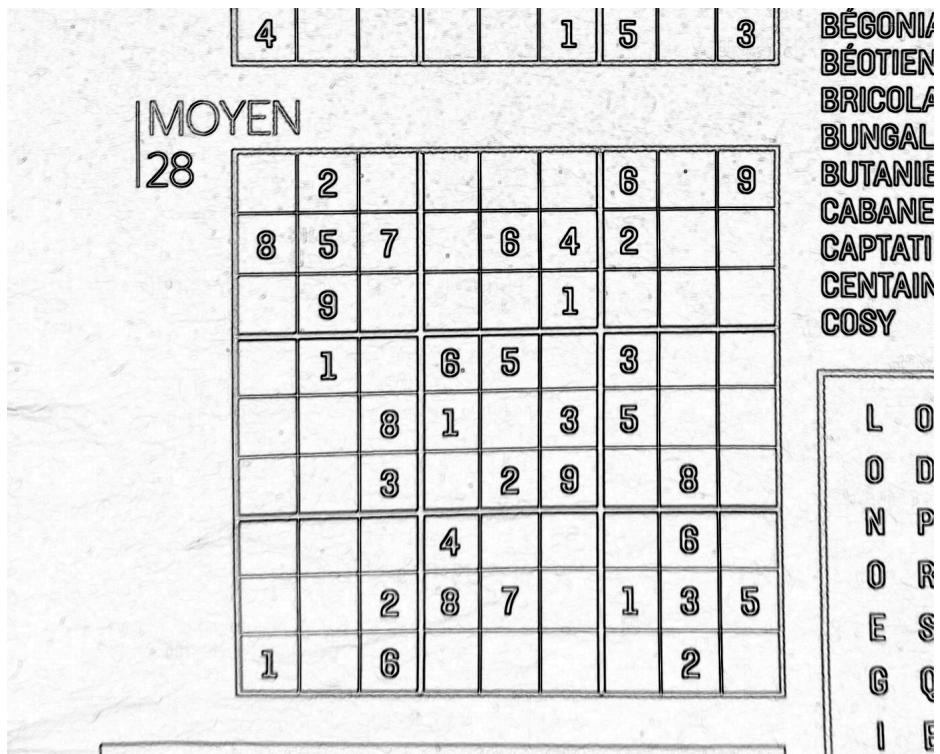
Similairement au filtre de Gauss, l'algorithme de Sobel va utiliser des kernels. Cette fois deux de 3x3. Un en x :

$$xKernel = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

et un en y :

$$yKernel = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Une fois appliqué, ces kernels nous permettent d'avoir des bords très nets.



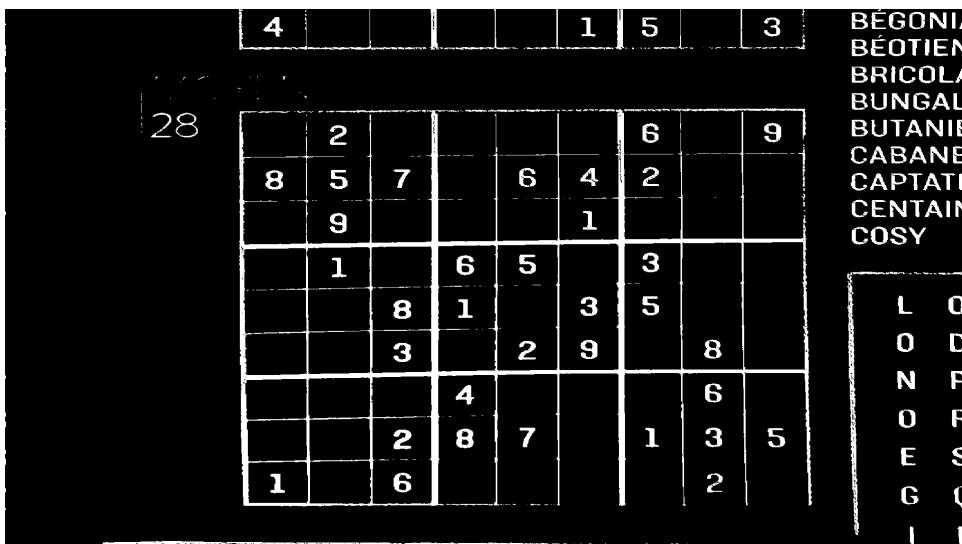
*Image après un filtre de gris, un filtre de Gauss et l'algorithme de Sobel*

Cet algorithme est également très pratique car il nous permet de savoir dans quel sens est le bord. Ainsi nous pouvons appliquer l'algorithme de Canny qui nous permet de garder que les bords les plus forts en fonction de leur angle.

## 6.2 Algorithme De Canny

Après l'algorithme de Sobel, nous avons pu récupérer l'orientation de chaque ligne dans le sudoku. Ainsi l'algorithme de Canny n'as plus qu'a vérifier si un bord est assez fort pour être gardé. Pour ce faire, l'algorithme de Canny peut se décomposer en plusieurs étapes. La première est d'affiner tous les bords pour qu'ils ne fassent plus qu'un pixel de largeur. Ensuite, nous devons regarder pour chaque pixel appartenant à un bord si c'est un maximum local, c'est-à-dire si dans l'orientation, donné par l'algorithme de Sobel, ses voisins sont plus intense ou pas. Si ils le sont alors on enlève ce pixel du bord et on garde les autres.

Pour finir, nous pouvons faire une petite correction pour être sur que les bords restent cohérents avec un double threshold. Ainsi, les bords dans l'images apparaissent très nettement et après binarisation la grille ressort correctement.



*Image après un filtre de gris, un filtre de Gauss, l'algorithme de Sobel, l'algorithme de Canny et la binarisation*

## 7 Réseau de neurones

Dans ce projet, le réseau de neurones est une composante capitale dans le bon fonctionnement du programme dans sa globalité puisque c'est lui qui va permettre de reconnaître les numéros dans chaque case. Sans cette étape de reconnaissance de caractère, il est impossible de compléter la grille de sudoku et le produit devient inutilisable. Mais la réalisation d'un tel réseau de neurone n'est pas chose des plus simples pour des deuxième année d'EPITA et nous avons donc décidé de procéder par étape.

## 7.1 Documentation

Dans un premier temps, nous avons décidé de se documenter et de comprendre la théorie derrière un réseau de neurone. Pour ce faire, de nombreuses sources de qualité sont à disposition sur internet comme la série de vidéo sur les réseaux de neurones faite par **3Blue1Brown** ou bien le livre de Mnielsen. Pour cette première soutenance, nous nous sommes donc concentré sur comprendre la théorie des réseaux de neurones et la mettre en pratique sur un exemple plus rudimentaire que la reconnaissance d'images. Nous avons donc réaliser un réseau de neurone qui apprend la fonction OU EXCLUSIF.

## 7.2 L'objectif de ce premier travail

Le but de ce premier réseau est vraiment de comprendre et maîtriser l'implémentation d'un réseau sur un exemple simple pour ensuite généraliser cette implémentation pour le réseau qui reconnaîtra les numéros. Ainsi, pour ce premier réseau, nous sommes parti sur un réseau extrêmement simple, composé de deux entrées, une couche cachée avec deux neurones dans celle-ci et une sortie. Ainsi, nous connaissons déjà les dimensions des matrices contenant les poids et les vecteurs contenant les biais. Ainsi, cette implémentation n'est pas très adaptable, en revanche elle est parfaitement fonctionnelle. De plus son implémentation, se révéla être extrêmement pédagogique pour comprendre l'aspect mathématique derrière les réseaux de neurones bien que celui-ci n'est pas encore maîtrisé à cent pour cent.

## 7.3 Les challenges

Le premier challenge fut de s'approprier le vocabulaire autour des réseaux de neurones. Un **neurone** est en réalité qu'un

nombre flottant qui correspond à sa valeur d'activation. Chaque neurone a un **biais** qui lui sont associés qui vont aider à décider si le neurone doit être actif ou inactif. Ces neurones sont contenus dans des **couches** qui sont juste un ensemble de neurones. Enfin, chaque neurone possède un lien avec chacun des neurones de la couche suivante. Ces liens ont un **poids** qui va déterminer l'importance que va avoir ce lien dans le calcul de la valeur d'activation du neurone de la couche suivante.

Le second challenge fut de comprendre les différentes phases qui composent l'apprentissage du réseau de neurone. L'algorithme de "**Feed Forward**" est extrêmement simple à comprendre : pour avoir la valeur d'activation du neurone de la couche prochaine, il faut faire une somme pondérées des neurones de la couche précédente multipliés par le poids les reliant au neurone de la couche suivante le tout ajouté au biais du neurone de la couche suivante. Ensuite une fois arrivée à la couche de sortie, on calcule l'erreur, c'est à dire on associe une valeur à l'écart qu'on a obtenu entre la valeur de sortie que nous voulions et ce que nous a rendu le réseau de neurones. Ensuite vient l'algorithme de **Backpropagation** qui va procéder à une descente de gradient et chercher à minimiser l'erreur sur chaque couche du réseau. Ainsi, cet algorithme calcule pour chaque couche à quelle point elle a faux et réajuste la valeur des biais et des poids en fonction.

Le dernier challenge fut d'apprendre à s'approprier les concepts que nous avions vu en cours magistral de programmation mais que nous n'avions pas encore mis en pratique en TP.

## 7.4 Implémentation

Comme dit précédemment, le réseau de neurones est très simple avec une couche cachée contenant deux neurones et une couche de sortie contenant un seul neurone. Avec l'implémentation ac-

tuelle du neurone, il est possible de le faire apprendre avec un nombre d'itérations et un taux d'apprentissage fixé par l'utilisateur, ou bien tout simplement de le faire sortir toutes les valeurs du OU EXCLUSIF d'un réseau qui a déjà été entraîné.

## 7.5 Sauvegarde des données après entraînement

Pour que le réseau de neurones prenne toute son utilité, nous ne voulons pas que celui-ci apprenne de nouveau à chaque fois que nous voulons l'utiliser. Par conséquent, nous avons dû implémenter une méthode de sérialiser le réseau de neurone et de le désérialiser pour pouvoir stocker les données associées à un réseau entraîné, et si besoin, pouvoir charger ses données pour éviter toute la phase d'apprentissage. Ce processus de persistance se fait via un stockage des poids, biais et valeurs d'activation des neurones du réseau dans un fichier texte. La fonction de chargement de ces valeurs s'occupe de lire ces valeurs et de les charger au bon endroit.

## 7.6 Améliorations prochaines

Ainsi, nous voulons améliorer notre implémentation du réseau de neurones en la rendant très adaptable et plus générique dans son traitement. A l'aide de struct notamment nous établirons une sorte de classe "Réseau de neurone" contenant des objets appartenant à la struct "couche", eux-même contenant des "neurones". En plus de développer un réseau adaptable, la procédure de sérialisation sera de même complètement dynamique et adaptée au modèle de réseau que nous cherchons à stocker.

## 8 Conclusion

Ainsi, malgré la difficulté de ce projet, nous avons tout de même bien avancé et réussi à remettre un travail conséquent. Cependant, nous n'avons pas réussi à remettre à temps tous les requis pour cette soutenance, et nous en excusons grandement. Nous sommes d'autant plus déterminé à travailler afin de remettre un projet fiable, fonctionnel, et facile d'utilisation.