

# Rapport de projet Projet S3 EPITA

Théo Gille

Clément Grégoire

Leandro Tolaini

Décembre 2023

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Présentation du groupe . . . . .	2
<b>2</b>	<b>Répartition des tâches</b>	<b>4</b>
<b>3</b>	<b>Prétraitement et rotation : Théo</b>	<b>5</b>
3.1	Prétraitement . . . . .	5
3.2	Parseur de l'image . . . . .	10
3.3	Ressentis . . . . .	10
<b>4</b>	<b>Parseur de l'image : Théo</b>	<b>13</b>
<b>5</b>	<b>Détection de grille : Clément</b>	<b>15</b>
5.1	Introduction . . . . .	15
5.2	Mise en évidence de la grille . . . . .	15
5.3	Détection des coins . . . . .	17
5.4	Vérifications . . . . .	17
5.5	Rotation automatique . . . . .	18
<b>6</b>	<b>Détection des cases : Clément</b>	<b>19</b>
6.1	Ressenti . . . . .	20
<b>7</b>	<b>Réseau de neurones : Leandro</b>	<b>21</b>
7.1	Conception . . . . .	21
7.2	La reconnaissance de chiffres . . . . .	22
7.3	Ressenti . . . . .	25
<b>8</b>	<b>Interface : Clément</b>	<b>26</b>
8.1	Présentation . . . . .	26
8.2	Reconstruction de la grille . . . . .	29
8.3	Vérification . . . . .	29
8.4	Ressenti . . . . .	29
<b>9</b>	<b>Conclusion</b>	<b>30</b>

# Chapitre 1

## Introduction

### 1.1 Présentation du groupe

Ce projet offrira une opportunité inestimable pour le développement de compétences avancées en programmation en langage C. En outre, il nous permettra d'acquérir une compréhension approfondie du fonctionnement des systèmes d'intelligence artificielle (IA), des techniques de traitement d'image et des principes sous-jacents à la conception d'interfaces graphiques conviviales. Par ailleurs, ce projet revêt une importance significative en ce qu'il renforcera notre capacité à travailler efficacement en groupe tout en perfectionnant notre aptitude à gérer un projet complexe. Cette expérience nous enseignera des compétences essentielles en gestion de projet, telles que la planification, la répartition des tâches, la communication et la collaboration au sein de l'équipe. En somme, ce projet constitue une opportunité d'apprentissage holistique dans le domaine de la technologie et de l'informatique, nous préparant ainsi à des défis professionnels futurs.

**Leandro :** J'appréhendais quelque peu la réalisation de ce projet, car il me semblait très professionnel, exigeant le développement d'un produit abouti. Heureusement, le fait de pouvoir compter sur une équipe motivée a rapidement dissipé ces inquiétudes. J'ai rapidement compris qu'en travaillant correctement, nous pourrions mener à bien ce projet avec brio. Cependant, la détermination seule ne suffit pas ; une bonne organisation était nécessaire. J'ai donc pris le rôle de chef de projet afin de décharger les autres de cette tâche. Finalement, c'est avec beaucoup d'enthousiasme que je me suis lancé dans sa réalisation, étant convaincu d'apprendre énormément sur un sujet qui m'intéresse : l'intelligence artificielle.

**Clément :** Pour ce projet, ma responsabilité inclut le développement du solveur ainsi que la conception de l'interface graphique. Étant donné que je n'ai aucune expérience préalable dans le domaine de l'interface graphique, cette initiative représente une opportunité précieuse pour acquérir une compréhension approfondie de son fonctionnement. Par ailleurs, ce projet dans son ensemble me permettra de renforcer mes compétences en programmation en langage C, tout en m'offrant une expérience significative en gestion de projets en équipe.

**Théo :** Depuis tout petit, j'ai toujours été intéressé par l'informatique, quel que soit le domaine, que ce soit les jeux vidéo, la cybersécurité, le traitement d'images, et ainsi de suite. J'ai tout d'abord découvert l'informatique à travers les jeux vidéo, en utilisant de nombreuses consoles comme la plupart des gens. C'est à partir du moment où j'ai eu mon premier PC que l'informatique a vraiment commencé à m'intéresser. Ainsi, vu mon engouement pour l'informatique, j'ai décidé de m'orienter vers une école d'ingénieur en informatique, et voilà où je suis actuellement. C'est lors du Projet S2 que j'ai pu vraiment voir ce qu'est un projet en informatique, car je me suis occupé de

tout ce qui concerne le multijoueur, mais aussi du développement de certaines mécaniques. Ainsi, durant l'ensemble de ce projet, malgré certains problèmes, j'ai pu y passer de très bons moments et j'en tire une bonne expérience. Avant de commencer ce projet qu'est l'OCR, je ne savais pas dans quoi nous allions être lancés, car avant de commencer, j'avais pu entendre que, comme moi, celui-ci est compliqué, mais surtout que c'est dans un nouveau langage de programmation que l'on n'avait pas encore fait. Mais je pense que je suis bien entouré avec des personnes motivées, ce qui devrait nous permettre de réussir ce projet.

## Chapitre 2

# Répartition des tâches

Pour cette première soutenance, nous avons décidé de nous répartir les tâches comme l'indique le tableau suivant :

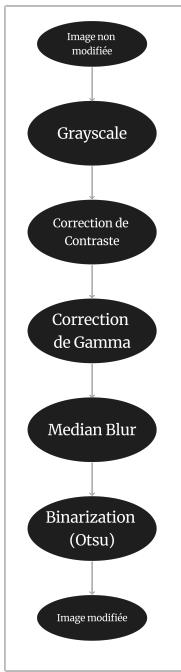
	Théo	Leandro	Clément
Prétraitement	X		
Solver			X
IA		X	
Parseur de l'Image			
Interface			X
Rotation	X		
Détection de la grille			X
Détection et découpage des cases			X

# Chapitre 3

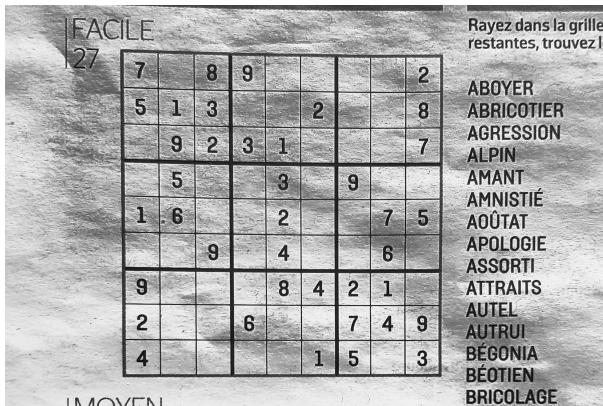
## Prétraitement et rotation : Théo

### 3.1 Prétraitement

Avant de pouvoir détecter les bords du Sudoku et ainsi passer à la phase du solveur, il faut rendre l'image plus lisible pour l'ordinateur et permettre l'application de différents algorithmes par la suite. Pour cette première soutenance, une première partie des filtres a déjà été implémentée, la voici :



Dans un premier temps, il faut appliquer un grayscale pour faciliter l'application des filtres suivants. Le grayscale, comme son nom l'indique, convertit l'image en niveaux de gris. Pour ce faire, il suffit d'appliquer une formule à tous les pixels de l'image :



$$\text{pixel} = 0.3 * \text{red} + 0.59 * \text{green} + 0.11 * \text{blue}$$

Correction de Contraste :

Après avoir appliqué le grayscale, nous devons procéder à une correction de contraste. Dans certains cas, cette correction n'est pas vraiment nécessaire. Il est alors préférable d'opter pour une correction automatique du contraste plutôt qu'en utilisant des valeurs prédéfinies. Pour cela, il faut :

- Rechercher les valeurs minimales et maximales des pixels de l'image afin de calculer un facteur de contraste, puis ajuster en conséquence la valeur de chaque pixel.

Le facteur de contraste se calcule en faisant :

$$\text{contrastFactor} = 255.0 / (\text{maxGray} - \text{minGray});$$

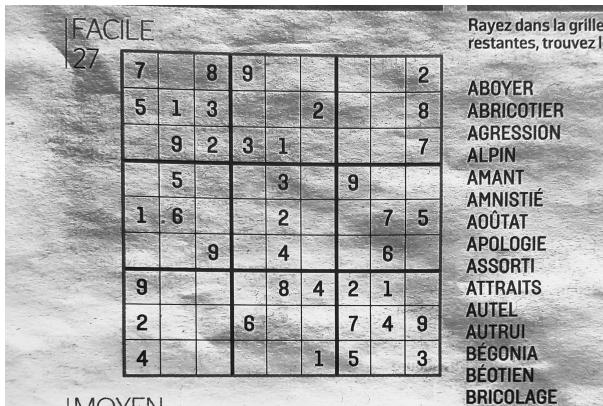
Pour calculer la bonne valeur de chaque pixel, il suffit de procéder comme suit :

$$\text{grayValue} = \text{grayValues}[y * \text{width} + x]; \text{adjustedValue} = (\text{contrastFactor} * (\text{grayValue} - \text{minGray}));$$

Pour éviter un dépassement de valeur dans un intervalle I, qui est ici [0 ; 255], on utilise une fonction de limitation (clamp).

Ainsi, le filtre se décompose de la manière suivante :

- La fonction "clamp" maintient une valeur à l'intérieur d'une plage spécifique.
- "adjust\_contrast" détermine les valeurs de gris minimale et maximale dans une image.
- En utilisant ces valeurs, elle calcule un facteur de contraste.
- Elle ajuste chaque pixel en fonction du contraste calculé pour améliorer la netteté et le contraste de l'image.



Correction de Gamma :

Une correction gamma sera effectuée après avoir fait de même avec le contraste. Comme pour le contraste, cette correction n'est pas nécessairement la même pour toutes les images et peut être beaucoup moins importante. Il faut alors calculer automatiquement cette valeur et ne pas utiliser de valeurs prédéfinies. Pour cela, il faut :

- Calculer la luminosité moyenne d'une image :  
 $\text{average\_luminosity} = \text{sum} / \text{num\_pixels}$   
 num\_pixels : Taille de l'Image  
 sum : additions de chaque pixel de l'image.
- En fonction de cette luminosité, des corrections gamma spécifiques sont définies.
- Chaque pixel est ajusté selon la correction gamma associée à sa luminosité, améliorant ainsi la qualité visuelle de l'image.

Formule pour la correction :

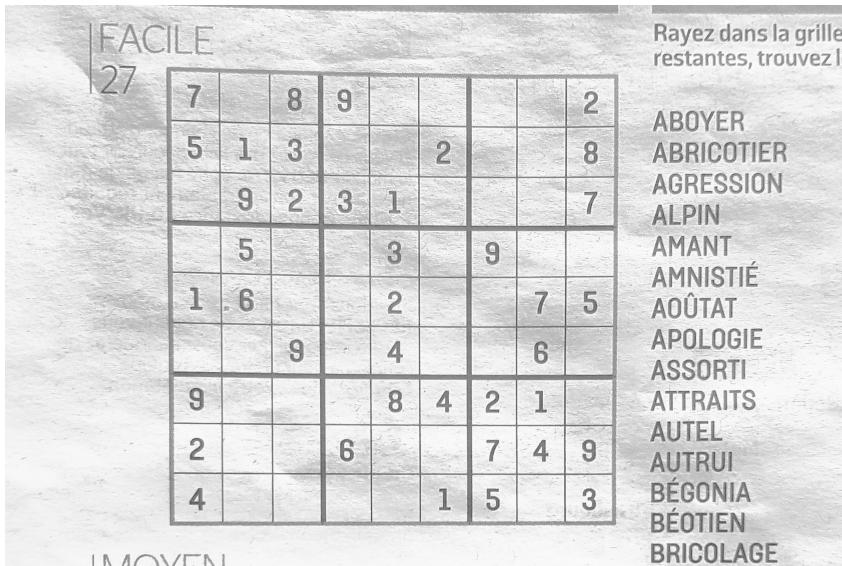
Soit une image représentée par une matrice  $I$ , où  $I_{xy}$  représente l'intensité du pixel à la position  $(x, y)$  de l'image.

La formule pour le filtrage médian peut être exprimée comme suit :

$$I_{\text{med}}(x, y) = \text{median}(I_{\text{voisinage}}(x, y))$$

Où  $I_{\text{voisinage}}(x, y)$  est l'ensemble des valeurs de pixels dans un voisinage entourant le pixel à la position  $(x, y)$ , et  $I_{\text{med}}(x, y)$  est la valeur du pixel médian à cette position.

La fonction médiane (median) calcule la valeur médiane à partir de l'ensemble des valeurs de pixels dans le voisinage.



Pour finir, Il faut passer par une binarization de l'image donc à un passage en noir et blanc. Pour cela, on utilise l'algorithme d'Otsu. Le principe de cette algorithme est de calculer un seuil globale pour l'image, si la valeur d'un pixel est supérieur au seuil, alors le pixel devient noir sinon blanc.  
On peut donc le diviser de cette manière :

1. Calcul de l'histogramme des niveaux de gris de l'image  $H(i)$  :

$$H(i) = \frac{\text{Nombre de pixels avec une intensité de niveau } i}{\text{Nombre total de pixels dans l'image}}$$

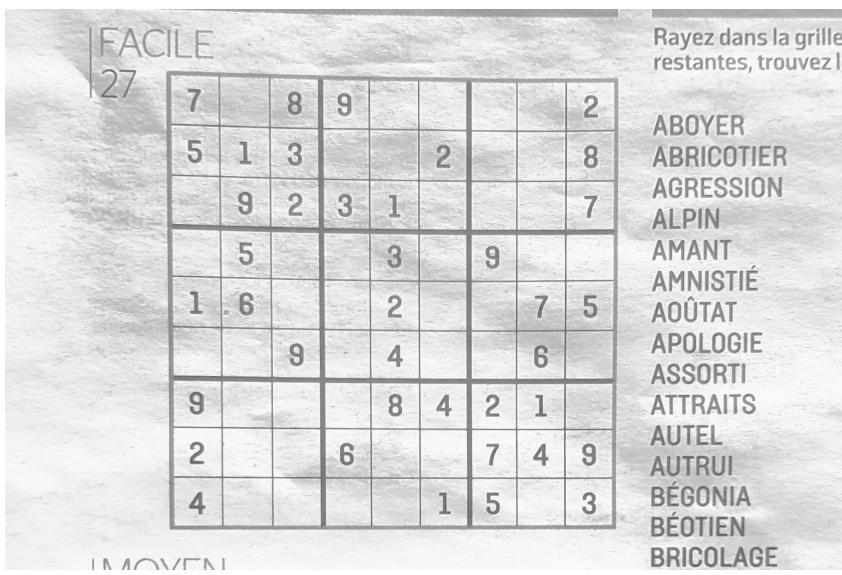
2. Calcul de la moyenne globale (mean\_total) des niveaux de gris de l'image :

$$\text{mean\_total} = \sum_{i=0}^{255} i \times H(i)$$

3. Parcourir tous les niveaux de gris possibles ( $t$ ) et calculer la variance interclasse pour chaque seuil  $t$  :

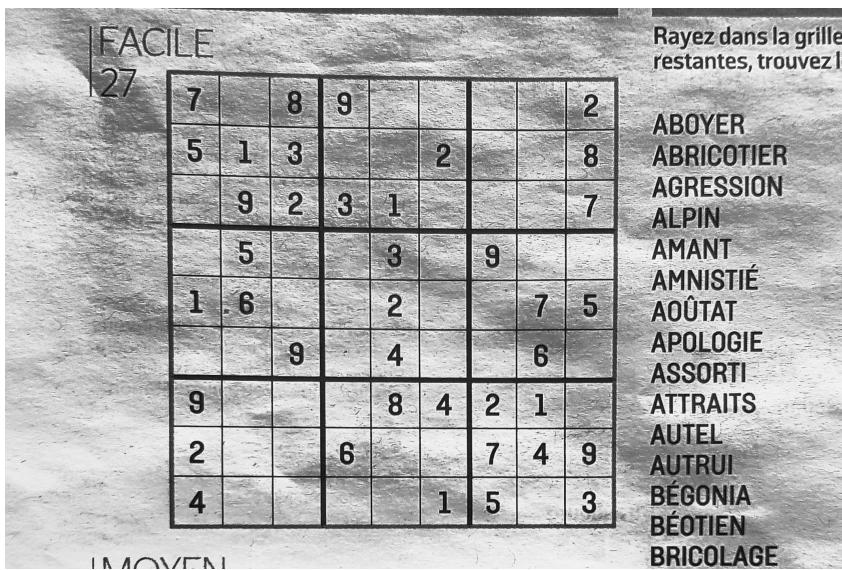
$$\text{variance\_inter}(t) = \text{BackG} \times \text{ForeG} \times (\text{mean\_BackG} - \text{mean\_ForeG})^2$$

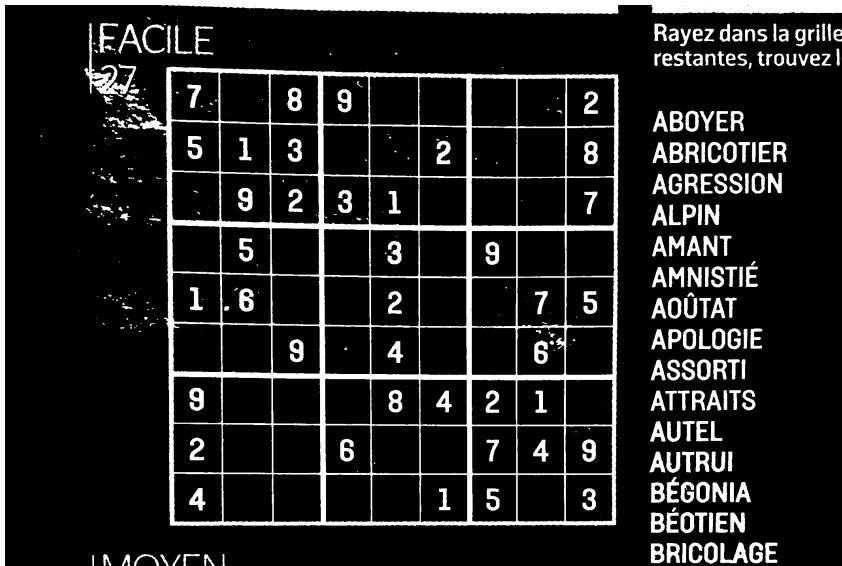
4. Sélectionner le seuil (threshold)



Résultat :

Après applications successives de ces différents filtres/algorithmes, cela nous permet d'avoir des résultats exploitables pour la détection de grille et donc par la même occasion pour le réseau de neurones.





### 3.2 Parseur de l'image

Afin de convertir une image en fichier texte pour permettre l'utilisation de l'interface du solveur par la suite, les trois parties majeures du projet sont nécessaires : le réseau de neurones, le prétraitement et la détection de la grille/cases.

On doit alors diviser le tout en 4 parties :

- Chargement et prétraitement d'images :  
Lire une image JPEG et convertir ses pixels en valeurs numériques comprises entre 0 et 1.
- Détection de chiffres :  
Déetecter les chiffres dans une image à l'aide d'un réseau de neurones entraîné. Elle renvoie l'index du chiffre détecté.
- Estimation du contenu des images :  
Estimer si une région spécifique de l'image est vide ou non en comptant les pixels blancs dans des zones définies.

### 3.3 Ressentis

Dans un premier temps, je suis fier de ce que j'ai pu accomplir ainsi que l'ensemble du groupe, malgré les différents problèmes que nous avons rencontrés chacun. Certaines parties du prétraitement, comme notamment la bonne binarisation, ont été complexes à comprendre et à implémenter initialement. Ce projet nous a permis à chacun d'apprendre de nouvelles choses et d'améliorer notre compréhension du langage C.

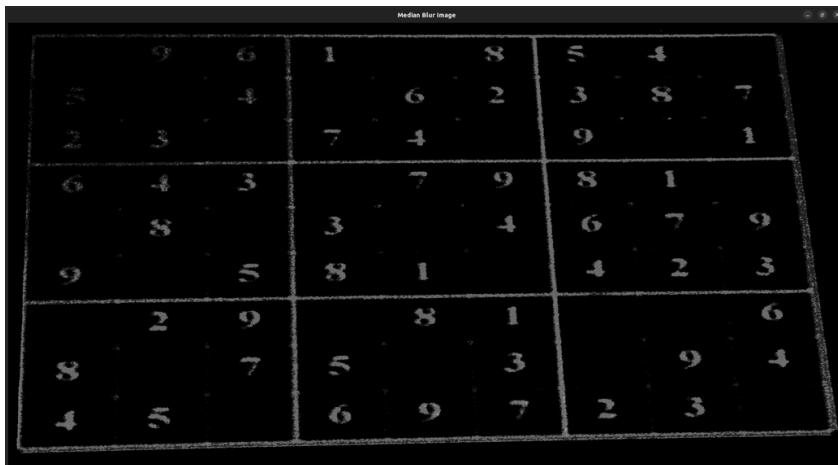
Filtre Gaussien ou Flou Gaussien :

Le filtre gaussien est utilisé pour réduire le bruit de l'image et égaliser les couleurs afin de faciliter le traitement. Le processus de ce filtre est simple. Pour chaque pixel de l'image, il faut calculer une nouvelle valeur à partir de la somme du produit des pixels voisins et d'une matrice gaussienne. Cette matrice est générée à l'aide d'une fonction gaussienne, qui est de la forme suivante :

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Gamma Le réglage de la partie Gamma de l'image permet d'accentuer les contrastes entre les zones claires et sombres de l'image. Ainsi, la formule à utiliser sera :

```
double factor = (259 * (contrast + 255)) / (255 * (259 - contrast));
double new_r = factor * (r - 128) + 128;
double new_g = factor * (g - 128) + 128;
double new_b = factor * (b - 128) + 128;
```

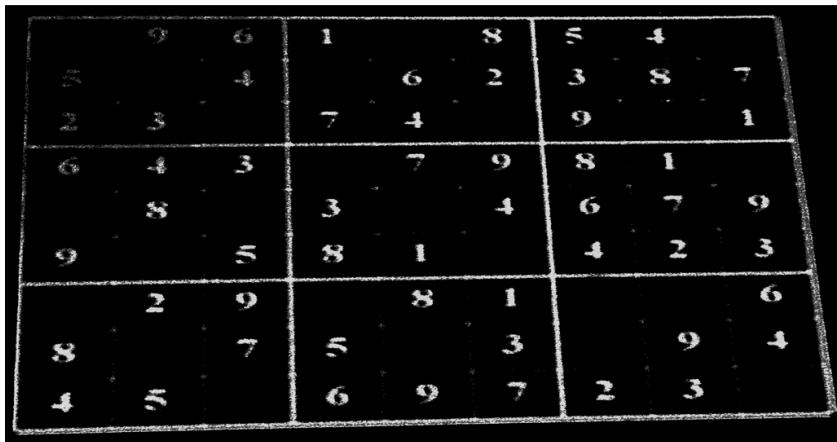


Correction du Contraste :

La correction du contraste de l'image permet d'améliorer la visibilité de certains détails de l'image et donc d'améliorer sa qualité.

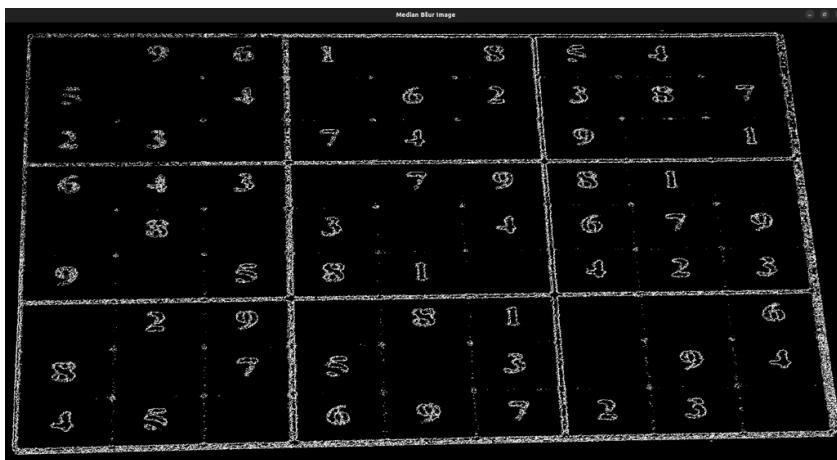
Dans notre cas ici on augmente juste la valeur de chaque pixel par 2 après l'avoir passé en niveaux de gris.

```
red = red * 2;
green = green * 2;
blue = blue * 2;
```



Filtre de Canny :

Le filtre de Canny permet de détecter les bords d'une image, en l'occurrence notre Sudoku. Il regroupe en lui plusieurs autres filtres qui, une fois assemblés, mettent en valeur les bords de l'image. Il est composé d'une réduction du bruit, d'un calcul du gradient d'intensité, de la détermination de la direction des contours, de la suppression des non-maxima, et enfin du seuillage des contours.



## Chapitre 4

# Parseur de l'image : Théo

Afin de convertir une image en fichier texte pour permettre l'utilisation de l'interface du solveur par la suite, les trois parties majeures du projet sont nécessaires : le réseau de neurones, le prétraitement et la détection de la grille/cases.

On doit alors diviser le tout en 4 parties :

Chargement et prétraitement d'images :

- Lire une image au format JPEG et convertir ses pixels en valeurs numériques comprises entre 0 et 1 constituent une étape cruciale. Cette étape rend l'image lisible par le réseau de neurones. Pour ce faire, il suffit de parcourir l'intégralité de l'image, de prendre la valeur de chaque pixel et de la diviser par 255. Cette opération permet d'obtenir un nombre compris entre 0 et 1.

Cette méthode fonctionne étant donné qu'elle intervient après le prétraitement complet de l'image, qui est alors en noir et blanc."

Détection de chiffres :

- Déetecter les chiffres dans une image à l'aide d'un réseau de neurones entraîné. Elle renvoie l'index du chiffre détecté.

Pour permettre la détection de chaque chiffre sur l'image, nous utilisons le réseau de neurones mentionné précédemment. Ce processus se divise en deux étapes :

1. Propagation en Avant(feedForward)
2. Determination du chiffre prédit

La première étape implique une propagation avant, où les pixels de l'image servent d'entrée au réseau neuronal via la fonction "feedForward". Cette fonction calcule la sortie du réseau en se basant sur les poids et les biais appris lors de l'entraînement.

La seconde étape consiste à déterminer le chiffre prédit par le réseau de neurones. À cette fin, le réseau retourne une sortie pour chaque neurone dans la dernière couche. Ces sorties représentent la confiance du réseau dans la prédiction de chaque chiffre.

Le code parcourt ces sorties pour identifier celle qui est maximale (la plus probable). Le neurone avec la sortie la plus élevée est considéré comme la prédiction du réseau pour le chiffre représenté dans l'image. Cependant, comme ces sorties représentent des prédictions, des erreurs peuvent se produire lors du processus de détermination. Ces erreurs pourraient entraîner des problèmes dans la résolution du Sudoku.

Estimation du contenu des images :

- Estimer si une région spécifique de l'image est vide ou non en comptant les pixels blancs dans des zones définies.

Pour permettre une détection des cases sans problème, deux fonctions similaires mais explorant des parties différentes de l'image ont été mises en place.

La première fonction examine si les pixels autour du centre de l'image, dans une plage donnée, sont

principalement blancs. La seconde fonction réalise une tâche similaire, mais cette fois, elle inspecte la totalité de l'image.

On pourrait se demander pourquoi deux fonctions de ce type sont nécessaires plutôt que d'examiner simplement la zone centrale ou l'intégralité de l'image. La raison réside dans la possibilité de présence de bruit dans l'image ou du chiffre ne se trouvant pas au centre.

Dans de tels cas, une seule fonction serait moins efficace pour détecter correctement la présence d'un chiffre.

Ainsi, une case est considérée comme non vide dans deux cas spécifiques :

```
if (( !estVide(surface) estVide2(surface)) —— (estVide(surface) !estVide2(surface)))  
ou
```

```
if(estVide2(surface))
```

estVide : fonction qui parcourt et vérifie une grande partie de la case.

estVide2 : fonction qui parcourt et vérifie seulement le centre de la case.

surface : L case que l'on parcours.

Si aucune de ces deux conditions n'est remplie, nous supposons alors que la case est vide, c'est-à-dire qu'elle ne contient pas de chiffre. Dans ce cas, nous inscrivons un point dans le fichier requis pour résoudre ultérieurement le Sudoku, puis nous supprimons l'image du dossier des cases. Seules les cases contenant un chiffre sont conservées. Cependant, si l'une des deux conditions est remplie, cela signifie que la case pourrait potentiellement contenir un chiffre. Nous procérons alors à un appel au réseau de neurones pour déterminer et inscrire le chiffre présent dans la grille à l'emplacement spécifié.

Fonctions Utiles :

Pour faciliter la manipulation des pixels de l'image, j'ai créé deux fonctions : getPixel et setPixel.

getPixel : Cette fonction permet de récupérer la valeur d'un pixel aux coordonnées x et y dans une image. Son fonctionnement est divisé en trois parties :

- Localisation du pixel.
- Détermination de la taille du pixel.
- Lecture de la valeur du pixel.

La manière dont le pixel est lu diffère selon sa taille, d'où l'importance de cette étape dans le processus. Cette lecture varie en fonction de la façon dont les données des pixels sont stockées en mémoire et de la manière dont elles doivent être interprétées pour obtenir la valeur correcte du pixel. setPixel : Cette fonction repose sur le même principe que getPixel. L'écriture de chaque pixel dépend de la manière dont il est stocké en mémoire.

# Chapitre 5

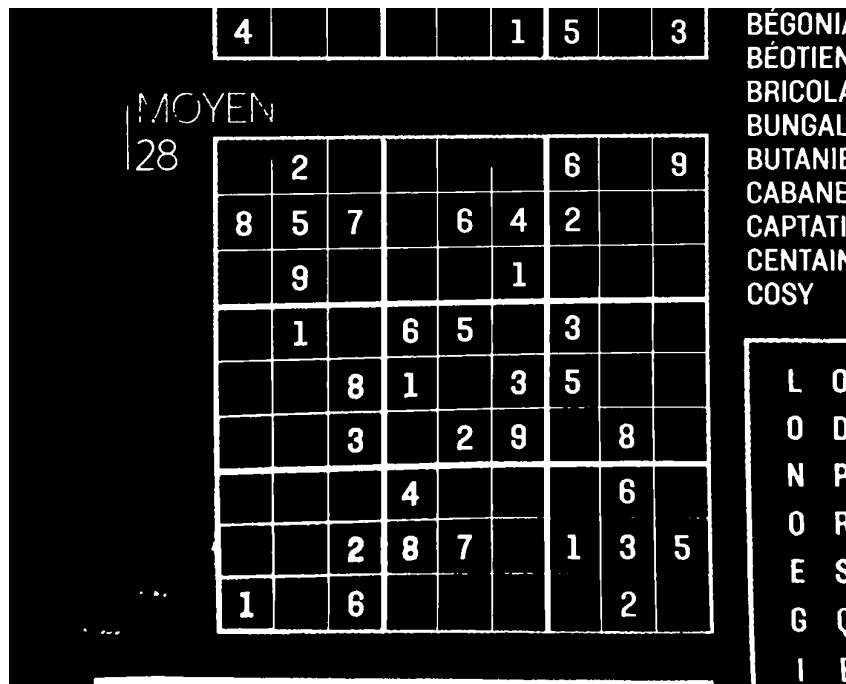
## Détection de grille : Clément

### 5.1 Introduction

Pour cette section, il est impératif de remédier au retard accumulé avant la première soutenance. En effet, à la suite de cette dernière, l'analyse de la grille n'avait pas encore été amorcée. Contrairement à ce qui avait été énoncé lors du premier rapport de soutenance, j'ai décidé de ne pas recourir à la transformée de Hough. À la place, j'ai opté pour une solution qui m'a semblé plus aisée à mettre en œuvre.

### 5.2 Mise en évidence de la grille

Pour détecter la grille, j'ai entrepris de mettre en évidence tous les traits continus présents sur l'image. Une fois le prétraitement effectué, l'image est convertie en nuances de gris.



Le programme crée un tableau de la taille de l'image, où chaque case correspond à un pixel de l'image. Ensuite, chaque pixel de l'image est parcouru. Si le pixel est blanc, il est considéré comme un trait, et la case du tableau qui lui est associée prend la valeur de 1. Les pixels noirs sont considérés comme le

fond de l'image, et la valeur du tableau est donc mise à 0. En appliquant ce processus à chaque pixel, on obtient un tableau de la taille de l'image où chaque trait est représenté par une suite continue de cases 1.

Une fois le tableau obtenu, il est possible de le parcourir afin de séparer les différents traits. À cette fin, j'ai employé un algorithme récursif qui s'inspire du principe d'un parcours en profondeur. Le tableau est parcouru tant que les cases sont égales à 0.

```
int cpt = 2; //nombre de traits différents
for (int y = 0; y < h ; y++)
{
    for (int x = 0; x < w ; x++)
    {
        if (tab[y][x] == 1) //pixel de contour: appel de la fonction recursive;
        {
            recursive(y, x, cpt);
            cpt+= 1;
        }
    }
}
```

Dès qu'une case à la valeur 1 est rencontrée, l'algorithme entre dans une fonction récursive, prenant en paramètre les coordonnées de la case détectée. Cette fonction vérifie si la case est effectivement un 1 ; dans le cas contraire, la fonction se termine sans valeur de retour. Si la case est un 1, la valeur du tableau est modifiée pour un nombre autre que 0 et 1, et la fonction est rappelée sur les 8 cases voisines de cette case. Ainsi, toutes les cases liées à la première case 1 sont assignées à une autre valeur, permettant d'identifier un trait continu.

```
void recursive(int y, int x, int cpt)
{
    // if(x < 2 || x > w-3 || y < 2 || y > h-3) return; //pixel hors de l'image
    if(x < 3 || y < 3) return; //pixel hors de l'image
    if(y > h-3 || x > w-3) return;
    if (tab[y][x] == 0) return; //le pixel appartient au fond et on arrete le parcours;
    if (tab[y][x] == 1) //le pixel est un bord et n'a jamais été visité, on le marque comme appartenant au trait numero cpt
        //on rappelle la fonction sur ses 8 voisins
    {
        tab[y][x] = cpt;
        recursive(y+1, x+1, cpt);
        recursive(y+1, x, cpt);
        recursive(y+1, x-1, cpt);
        recursive(y, x+1, cpt);
        recursive(y, x-1, cpt);
        recursive(y-1, x+1, cpt);
        recursive(y-1, x, cpt);
        recursive(y-1, x-1, cpt);
    }
    return;
}
```

Une fois que la fonction récursive se termine, le parcours du tableau est repris. Tant que les cases sont égales à 0, on passe à la case suivante et on réexécute la fonction récursive sur les cases dont la valeur est 1.

Grâce à ce tableau, il devient possible d'assigner une couleur à chaque trait, permettant ainsi de différencier visuellement chaque trait en couleur distincte.

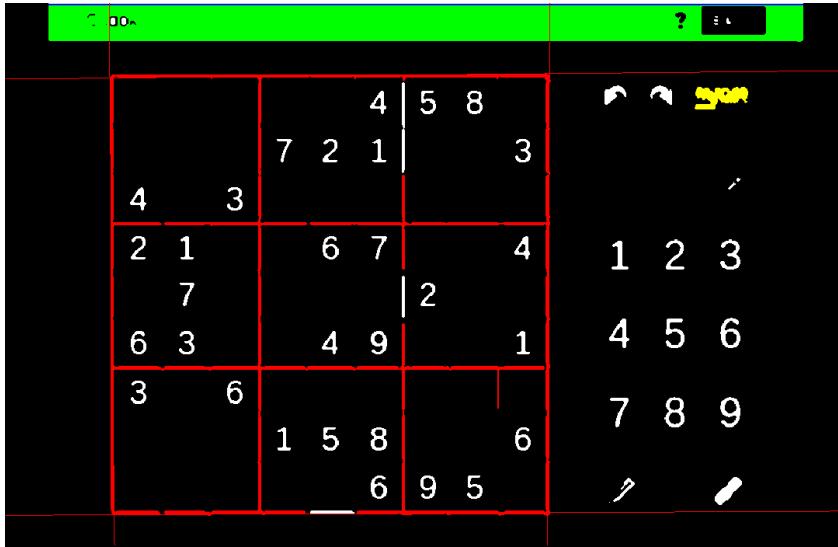
Il est à noter que l'image résultante n'est pas construite au cours du processus de détection de la grille, mais elle sert à visualiser le fonctionnement du programme.

Ensuite, nous créons un nouveau tableau ayant pour dimension le nombre de traits distincts présents dans l'image. Dans la case numéro n de ce tableau, nous stockons le nombre de pixels appartenant au n-ième trait. À partir de ce nouveau tableau, nous pouvons déterminer l'occurrence la plus élevée parmi les pixels. Nous considérons cette occurrence maximale comme représentative de la grille, en supposant que l'utilisateur fournira une photo où le sudoku constituera probablement l'élément prédominant. Nous reviendrons sur cet aspect ultérieurement dans ce rapport.

À partir de ce tableau, nous sommes en mesure de colorer sur l'image filtrée l'ensemble des pixels appartenant à l'occurrence la plus élevée du tableau en rouge.

### 5.3 Détection des coins

Afin de détecter les coins de la grille, nous procérons de la manière suivante : nous parcourons l'image de gauche à droite, de haut en bas, jusqu'à ce que nous rencontrions un pixel rouge. Dès que cela se produit, le parcours s'interrompt. Ensuite, nous vérifions si le pixel se trouve dans la partie gauche ou droite de l'image. S'il est dans la partie gauche, nous nous déplaçons de 100 pixels vers la droite. Ensuite, nous parcourons l'image en hauteur jusqu'à ce que nous trouvions un pixel appartenant à la grille. À partir de ce pixel, nous traçons la droite qui relie les deux points appartenant à la droite. Si le premier pixel de l'image est dans la partie droite, nous effectuons les mêmes étapes en nous déplaçant de 100 pixels vers la gauche. Ce processus est répété en commençant à parcourir l'image à partir du bas, à partir de la gauche et de la droite. Nous obtenons ainsi quatre droites qui coïncident avec les quatre côtés de la grille.



Nous pouvons ensuite effectuer le parcours de ces lignes afin de déterminer leur intersection. Ces intersections sont donc les coordonnées des quatre coins de la grille.

### 5.4 Vérifications

Une fois ces coordonnées obtenues, nous effectuons des vérifications. Nous partons du principe que la plus grande ligne continue est la grille, mais cela n'est pas toujours le cas. Par exemple, sur l'image ci-dessus, le trait le plus long est vert et ne correspond pas à la grille.

Nous vérifions que les droites du haut et du bas sont parallèles en comparant les pentes des deux droites. Si les pentes diffèrent de plus de 0,05 (environ un décalage de 15 à 20 pixels sur la largeur de l'image), nous vérifions le nombre de pixels détectés pour déterminer la droite. Si l'une des deux droites compte moins de 100 pixels(\*), nous calculons une nouvelle droite en prenant les deux droites perpendiculaires de gauche et de droite, sur lesquelles nous appliquons le nombre de pixels de 10 ou 11 pour déterminer les nouvelles coordonnées. Par exemple, si la droite du haut est mal définie (10 < 100) et les pentes sont différentes, et si la longueur de la droite du bas (l1) est de 500 pixels, nous calculons la distance de 500 pixels avec l'équation de la droite de gauche à partir de l'intersection des droites du bas et de la gauche. Cela nous permet de déterminer  $(x_0, y_0)$ . Ensuite, nous calculons la distance de 500 pixels avec l'équation de la droite de droite à partir de l'intersection des droites du bas et de la droite, ce qui nous donne  $(x_1, y_1)$ . Les points  $(x_0, y_0)$  et  $(x_1, y_1)$  permettent de calculer la nouvelle droite.

Sur une image de 700 pixels de largeur et de hauteur, on estime qu'au moins la moitié de l'écran contient la grille de Sudoku, soit un minimum de 350 pixels. En considérant que 100 pixels correspondent à environ 2 cases sur 9, détecter moins de 2 cases est sujet à doute pour justifier la validité de la droite.

(\*) Note : La valeur de seuil de 100 pixels est un exemple, elle peut être ajustée en fonction des spécificités

de l'image et des besoins de détection.

Nous vérifions ensuite que les droites de gauche et de droite sont parallèles en s'assurant que les pentes sont supérieures à 30 ( $+/-30$  pixels en  $y$  pour 1 pixel en  $x$ ). Si la pente est inférieure à 30 pour l'une des deux droites, nous calculons une nouvelle droite en appliquant le même raisonnement que pour le haut et le bas.

Ensuite, nous vérifions que les quatre longueurs de lignes,  $l_0$  à  $l_3$ , sont cohérentes (différence de moins de 20%). Si les longueurs sont incohérentes, tous les tests sont recommandés à partir du second trait le plus long de l'image. Ce processus peut être répété jusqu'à quatre fois. Si au bout du quatrième essai, la grille n'est pas détectée, le programme s'arrête avec un code d'erreur.

Ces vérifications semblent être assez restrictives pour détecter la grille sur toutes les images que nous avons testées.

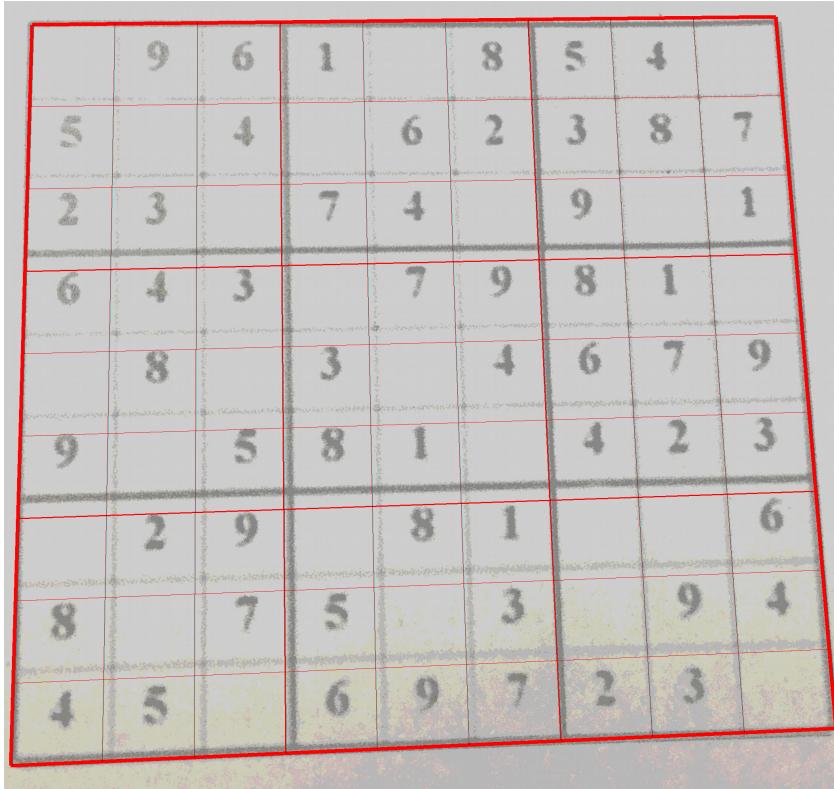
## 5.5 Rotation automatique

Une fois les coordonnées des quatre coins détectées, nous pouvons procéder à la rotation de l'image. Étant donné que nous disposons des droites associées aux quatre côtés de la grille, nous pouvons calculer l'angle formé avec l'horizontale. Si cet angle est supérieur à 4 degrés, la grille est redressée grâce au programme de rotation. Une fois la grille pivotée, nous relançons le programme de détection des coins, car leurs coordonnées ont été modifiées suite à la rotation.

## Chapitre 6

# Détection des cases : Clément

Comme nous avons les coordonnées des quatre coins ainsi que la longueur des quatre côtés de la grille, nous pouvons détecter les cases en effectuant une division par 9. Nous prenons le côté gauche, multiplions sa longueur par un neuvième, puis répétons le processus avec le côté droit, reliant ces deux coordonnées. Nous continuons jusqu'à huit neuvièmes pour obtenir les huit lignes intermédiaires du sudoku. Nous répétons le même processus avec les lignes du haut et du bas pour obtenir les 81 cases du sudoku. Cette méthode de détection des cases n'est pas optimale, et en cas de déformation de l'image, la précision de la détection peut être affectée. Cependant, pour l'image 6 qui est déformée, les résultats nous ont paru suffisamment satisfaisants pour ne pas nécessiter une détection plus précise. De plus, le retard a fait que nous n'avons pas eu le temps d'améliorer ce point spécifique. Enfin comme nous avons toutes les informations nécessaires, nous pouvons découper les 81 cases et les mettre dans un dossier cases



## **6.1 Ressenti**

Ces deux parties ont constitué un défi substantiel. J'étais initialement sous-estimé la complexité de ces tâches. De plus, le fait que ces sections étaient censées être achevées lors de la première soutenance a ajouté à la pression, d'autant plus que je devais simultanément travailler sur une autre partie du projet.

# Chapitre 7

## Réseau de neurones : Leandro

### 7.1 Conception

Pour commencer, j'ai essayé d'implémenter les neurones, les couches et le réseau neuronnel à la manière de la Programmation Orienté Objet du C étudié l'année dernière. En effet, les structures C et le mot clé "typedef" m'ont permis de créer des propres types, reliés à des structures et donc des attributs.

```
1 typedef struct Neuron
2 {
3     int nbr_weight;
4     double *weight;
5     double *end;
6     double bias;
7     double error;
8     double output;
9 } Neuron;
10
11 typedef struct Layer
12 {
13     int nbr_neurons;
14     Neuron *neuron;
15     Neuron *end;
16 } Layer;
17
18 typedef struct NeuralNetwork
19 {
20     int nbr_layers;
21     int nbr_inputs;
22     Layer *layer;
23     Layer *end;
24 } NeuralNetwork;
```

A partir de là, je devais réussir à initialiser chacun de ces attributs, notamment les poids et les biais. Pour ça j'ai implémenté une fonction qui génère un double suivant une distribution Gaussienne, grâce à la formule suivante :

$$\sqrt{-2 \cdot \log(u_1)} \cdot \cos(2\pi u_2)$$

où  $u_1$  et  $u_2$  sont compris entre 0 et 1.

En manipulant les attributs avec des pointeurs et des tableaux, j'ai pu initialiser correctement mon réseau de neurones. Maintenant il me restait à implémenter les fonctions pour le faire évoluer.

### Evolution

Grace à l'ebook fournit dans le sujet et la documentation, j'ai implémenté la fonction simoïde et sa dérivée, un algorithme de backpropagation, et de la fonction feedforward pour tester les résultats du réseau. En suivant les instructions et en manipulant correctement mes structures de données, l'a pouvait maintenant apprendre à partir d'exemples.

Les deux algorithmes principaux sont :

- Front propagation

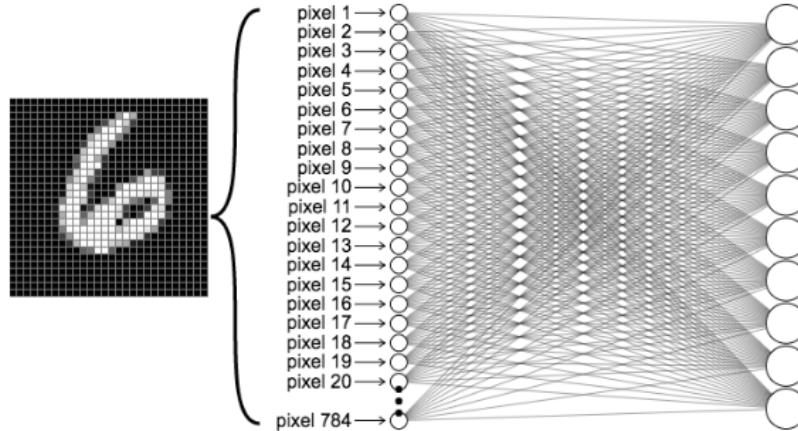
La fonction d'activation de notre front propagation est la fonction sigmoïde comme ci-contre :

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

- Back propagation

La fonction d'apprentissage pour permettre au réseau de comprendre ses erreurs est la fonction d'érivée de la sigmoïde :  $f'(x) = x \cdot (1 - x)$

- La mise à jour des poids de chaque neurones : grâce aux deltas de chaque neurones (différences entre la sortie attendue et la sortie réelle) calculés avec la backpropagation, il fallait mettre à jour les poids pour réduire au maximum l'erreur (gradient descendant).



Représentation de l'architecture du réseau

## 7.2 La reconnaissance de chiffres

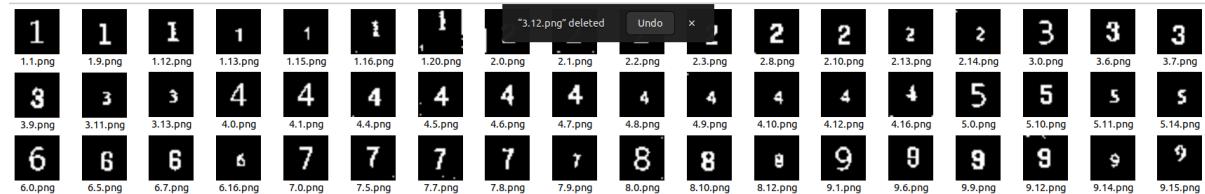
Depuis la dernière soutenance et l'exemple concret de l'apprentissage d'une fonction xor par le réseau, il fallait maintenant que je m'attelle à une fonction bien plus complexe : la reconnaissance de chiffres. Le premier problème, qui allait grandement influencer la performance de mon intelligence artificielle, était de trouver une database riche et qui répondait aux attentes du projet.

Database

Après beaucoup de recherche sur internet, j'ai trouvé une database correspondant à mes attentes : elle a été publiée par un utilisateur anonyme et s'inspire grandement de la base de donnée de MNIST, sauf que au lieu de contenir des chiffres écrit à la main, elle contient des chiffres de différentes polices d'écriture. Elle contient un large choix de données (29000 images), des images déjà à la bonne dimension (28x28 pixels) et est au format .csv, ce qui facilitera beaucoup le parsing du fichier. Ci-dessous la représentation de la database :

On peut voir que c'est un tableau qui contient 786 colonnes : la première étant réservée au nom de la police d'écriture, la deuxième au chiffre représenté par l'image, et les 784 autres pour la valeur entre 0 et 255 de chaque pixel de l'image. Elle dispose également de 29000 lignes, représentant chacune une image.

Et ici une représentation de quelques chiffres de la database de manière visuelle :



## Entrainement et tests

Pour rendre le réseau de neurones opérationnel, j'ai procédé de la façon suivante : pour chaque génération, la database était divisée en :

- 28000 images d’entraînement, c’est-à-dire qu’il y avait un cycle feedforward – backpropagation-mise à jour des poids pour chaque image
  - 1000 images de tests : ces images n’avaient jamais été rencontrées par le réseau et servaient de témoin de l’avancement de l’apprentissage du réseau, et ne contribuaient pas à son entraînement.

En répétant ces opérations un nombre significatif de fois, on obtient les résultats suivant :

```

Training a digit-recognition neural network...

Generation 0 :
Testing...
Results of the testing : 924 / 1000 found. (92.400000%)

Generation 1 :
Testing...
Results of the testing : 943 / 1000 found. (94.300000%)

Generation 2 :
Testing...
Results of the testing : 946 / 1000 found. (94.600000%)

Generation 3 :
Testing...
Results of the testing : 949 / 1000 found. (94.900000%)

Generation 4 :
Testing...
Results of the testing : 953 / 1000 found. (95.300000%)

Generation 5 :
Testing...
Results of the testing : 949 / 1000 found. (94.900000%)

Generation 6 :
Testing...
Results of the testing : 950 / 1000 found. (95.000000%)

Generation 7 :
Testing...
Results of the testing : 952 / 1000 found. (95.200000%)

Generation 8 :
Testing...
Results of the testing : 957 / 1000 found. (95.700000%)

```

Après la première génération et donc le premier entraînement avec 28000 images, on observe déjà un résultat assez satisfaisant de 92%, et avec les générations suivantes, on arrive à maximiser le résultats.

En jouant sur l'architecture de réseau (nombre de couches, nombre de neurones par couche...), sur le nombre d'entrainements et sur le coefficient d'apprentissage, on peut encore optimiser les sorties et après de nombreux tests, j'ai aboutit à un réseau avec plus de 98% d'efficacité

#### Sauvegarder le réseau

La dernière étape avant de pouvoir mettre l'IA en œuvre dans des cas concrets et la lier avec le reste du projet était de la sauvegarder, et d'être en mesure de charger rapidement un réseau déjà entraîné de façon optimale sans avoir à en créer un nouveau à chaque fois. Pour cela, j'écris simplement dans un fichier texte l'architecture du réseau, ainsi que tous ses poids et ses biais.

```

1 784
2 3
3 15
4 25
5 10
6 1.669901
7 0.115239
8 0.210954
9 0.149852
10 0.0171448
11 -1.509442
12 -0.838721
13 -1.415144
14 0.270503
15 -0.194212
16 -1.547212
17 0.128199
18 0.665406
19 1.554250

```

En quelques dixièmes de secondes, je peux maintenant charger un réseau performant :

```
Loaded results : 28462 / 29000 found.  
(98.144828%)  
lxdroot@lxdro-XPS-9315:~/enits/OCR/enits>
```

Conclusion :

Après des mois de travail, nous sommes heureux du résultat final du projet et pouvons être fier d'avoir beaucoup appris et d'avoir su se surpasser. Malgré ça, cela n'a pas été une bonne expérience sur tous les points, notamment à cause des relations aux sein du groupe à cause d'un manque d'implication. Finalement le projet a été intégralement développé par 3 membres, ce qui a beaucoup augmenté la charge de travail de chacun. Ce projet a été éprouvant pour tous, mais réussi avec brio.

### 7.3 Ressenti

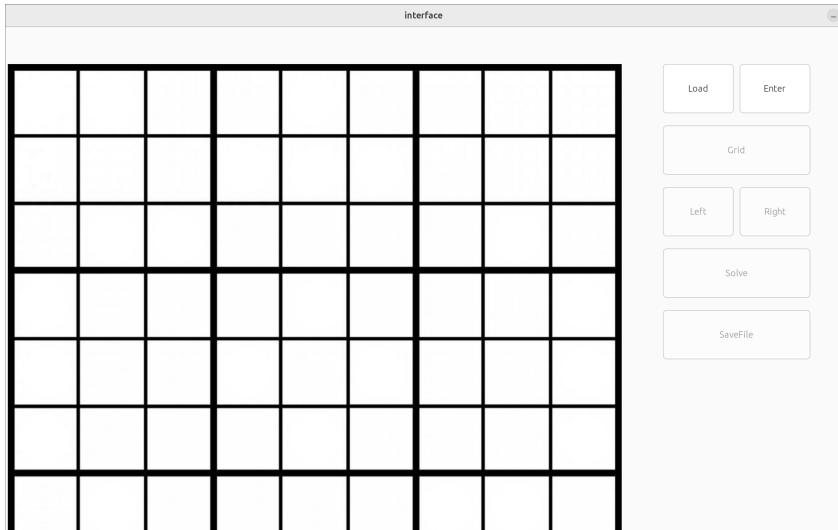
Chercher une database qui correspondait à mes critères s'est avéré une tâche plus complexe que ce à quoi je m'attendais, en effet c'est seulement après plusieurs essais peu fructueux que j'ai réussi à en trouver une bonne. J'ai réussi à implémenter un réseau performant grâce à un bon avancement dès la première soutenance et je suis content de ce que j'ai produit. De plus, cette partie du projet m'a beaucoup plu car l'intelligence artificielle m'intéresse, et j'ai pu apprendre beaucoup à ce sujet

# Chapitre 8

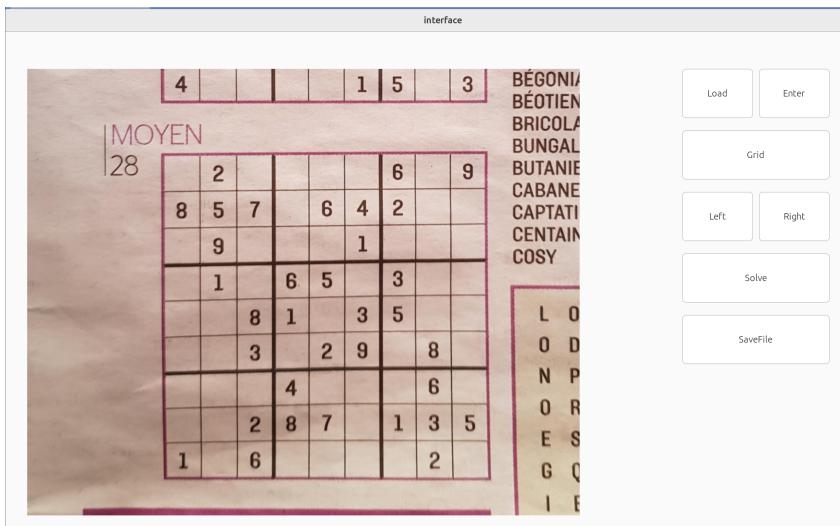
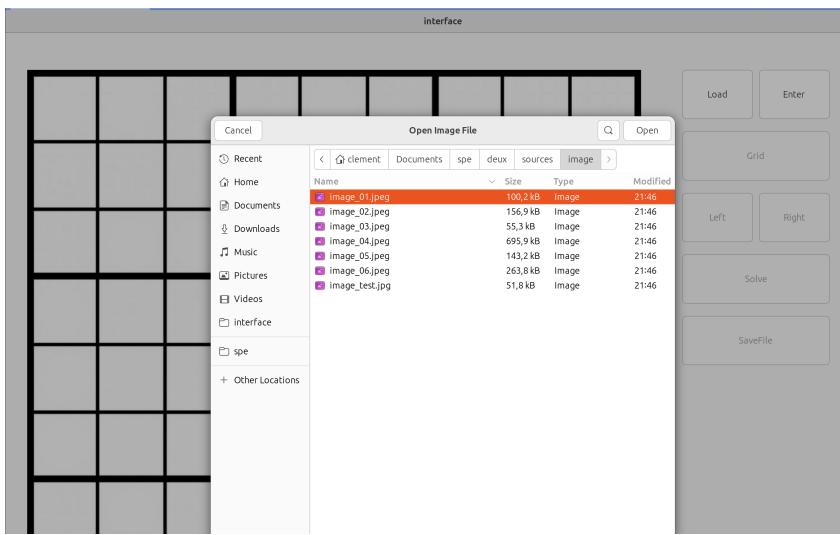
## Interface : Clément

### 8.1 Présentation

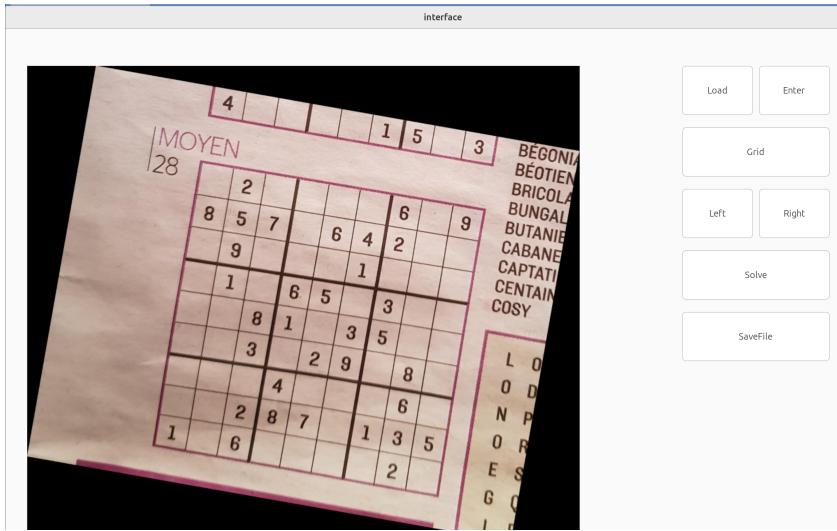
À l'origine, ma principale contribution devait se concentrer sur l'interface. Cependant, j'ai également dû prendre en charge la détection de la grille, ce qui explique la simplicité de l'interface.



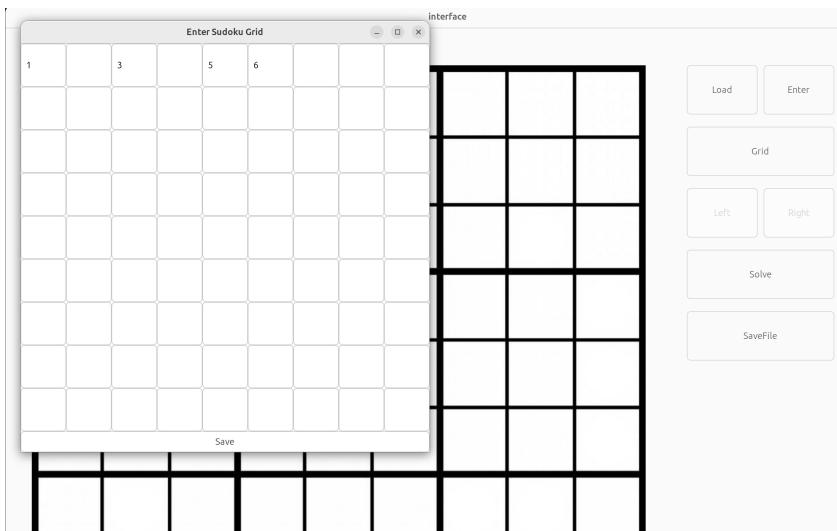
Lorsqu'on lance l'application, l'interface se présente avec plusieurs boutons. Le premier bouton permet de charger une image, une fonctionnalité qui était déjà présente lors de la première soutenance. Lorsqu'une image est chargée, elle s'affiche à l'emplacement de la grille vide sur l'interface.



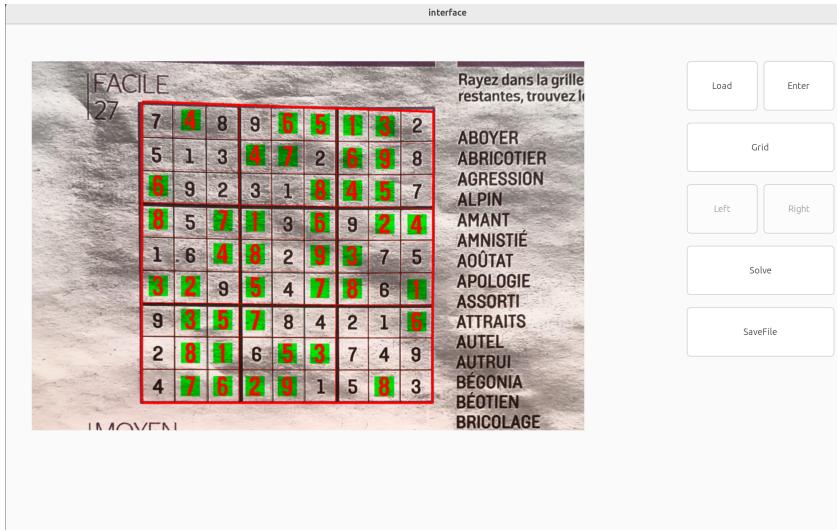
Les boutons "left" et "right" étaient également présents lors de la première soutenance, bien que leur fonctionnalité ne soit pas encore opérationnelle. Ils sont conçus pour effectuer une rotation manuelle de l'image, permettant ainsi de la redresser selon les besoins de l'utilisateur.  
Ces deux boutons sont simplement des appels à la fonction de rotation, prenant l'image affichée à l'écran comme argument.



Le bouton "Enter" permet à l'utilisateur d'entrer manuellement les valeurs de la grille. En activant ce bouton, une nouvelle fenêtre apparaît, composée de 81 cases dans lesquelles l'utilisateur peut saisir manuellement le contenu des cases. Une fois que l'utilisateur a rempli la grille, il peut appuyer sur le bouton "Save" pour sauvegarder la grille. Bien que la grille ne s'affiche pas à l'écran, lorsque le bouton "Solve" est pressé, la grille est résolue et son contenu s'affiche dans l'interface. Notons toutefois que l'affichage ne se réalise pas correctement dans la grille, un problème que je n'ai pas eu le temps de résoudre.



Le bouton "Solve" a pour fonction de résoudre la grille et d'afficher la solution dans l'interface. Ce bouton est un appel aux fonctions de prétraitement, de détection de la grille, de résolution, de reconstruction de la grille, et enfin, d'affichage de la nouvelle grille résolue. Ces fonctions ont déjà été détaillées ou seront détaillées par la suite.



Enfin, il est à noter que les boutons "SaveFile" et "Grid" ne sont pas fonctionnels pour le moment. Cependant, il est important de souligner que l'image de la grille résolue est toujours enregistrée dans le dossier "interface" sous le nom "original.png".

## 8.2 Reconstruction de la grille

Une fois la grille et les cases détectées, nous pouvons soumettre les images au réseau de neurones et construire le fichier de sortie qui contient la grille de l'utilisateur. Une fois cette grille résolue, il est nécessaire de reconstruire la grille de sortie afin de fournir la solution à l'utilisateur. Pour ce faire, nous disposons d'images représentant les chiffres de 1 à 9. Étant donné que nous connaissons les coordonnées des cases, nous pouvons parcourir le fichier de sortie qui contient la grille de l'utilisateur ainsi que le fichier output.result qui contient la solution. Si la case de la grille de l'utilisateur comporte déjà un chiffre, nous ne le réécrivons pas par-dessus. En revanche, si la case est vide, nous ajoutons le chiffre correspondant. Nous pouvons ensuite récupérer l'image ainsi modifiée et l'afficher dans l'interface. Si l'utilisateur rentre la grille manuellement, nous disposons également d'une image de grille vide que nous pouvons remplir. Dans ce cas, toutes les cases seront remplies.

## 8.3 Vérification

Au cours du processus, il est possible que l'intelligence artificielle ne reconnaisse pas correctement le caractère ou même qu'elle ne le détecte pas du tout. Ce problème entraîne généralement une impossibilité de résolution de la grille. Dans ce cas, l'interface affiche une boîte de dialogue pour informer l'utilisateur qu'une erreur s'est produite. L'utilisateur peut alors confirmer dans la boîte de dialogue qu'il a bien lu son contenu. Ensuite, une fenêtre Emacs s'ouvre avec le fichier de sortie, que l'utilisateur peut modifier pour corriger les caractères incorrects. Le processus de résolution peut alors se poursuivre même si l'IA s'est trompée sur certains caractères. Cela s'applique, par exemple, à l'image 6 où deux caractères sont incorrects.

## 8.4 Ressenti

Malgré la relative facilité de la conception de l'interface, j'ai rencontré des difficultés pour maîtriser la bibliothèque GTK. De plus, avec l'architecture de notre projet, l'interconnexion de tous les programmes s'est avérée parfois complexe, entraînant plusieurs erreurs de type "segmentation fault" qui m'ont demandé beaucoup de temps pour en identifier l'origine. Néanmoins, je suis fier du résultat que j'ai réussi à produire.

# **Chapitre 9**

## **Conclusion**

Pour ce rendu de projet, nous sommes très fiers de notre application. Nous avons dû surmonter plusieurs difficultés, telles que des résultats divergents d'un ordinateur à l'autre. De plus, nous avons perdu un membre du groupe en cours de route. En effet, nous avons dû réaliser ce projet à trois, en rattrapant le retard engendré. Malgré ces challenges, nous avons tout de même réussi à produire une application complète et fonctionnelle dans la plupart des cas.