

# OCR

## Soutenance 1

Julien BESTARD  
Léa BONET  
Hippolyte PIK  
Benjamin DREYFUS

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Présentation du projet . . . . .	3
1.2	Le groupe du projet . . . . .	3
<b>2</b>	<b>Répartition des tâches</b>	<b>4</b>
<b>3</b>	<b>Réseau de neurones</b>	<b>5</b>
3.1	Structure globale . . . . .	5
3.1.1	Front Propagation . . . . .	5
3.1.2	Back Propagation . . . . .	5
3.2	Le xor . . . . .	6
<b>4</b>	<b>Traitement de l'image</b>	<b>7</b>
4.1	Réduction de la taille . . . . .	8
4.2	Ajustement des couleurs . . . . .	8
4.2.1	Gamma . . . . .	8
4.2.2	Contrast . . . . .	9
4.2.3	Niveau de gris . . . . .	10
4.2.4	Gaussian Blur . . . . .	10
4.2.5	Méthode d'Otsu . . . . .	11
4.3	Rotation de l'image . . . . .	12
<b>5</b>	<b>Detection des bords</b>	<b>13</b>
5.1	L'opérateur Sobel . . . . .	13
<b>6</b>	<b>Résolution d'une grille</b>	<b>14</b>
6.1	Le Solveur . . . . .	14
6.1.1	Le principe . . . . .	14
6.1.2	Utilisation du programme . . . . .	14
<b>7</b>	<b>Ce qui sera implémenté</b>	<b>15</b>
7.1	Grilles et cases . . . . .	15
7.2	Interface Graphique . . . . .	15
<b>8</b>	<b>Conclusion</b>	<b>15</b>

# 1 Introduction

## 1.1 Présentation du projet

Voici le rapport de soutenance pour notre projet de S3. Il présente l'avancement actuel de notre OCR ainsi que les prochaines étapes à implémenter pour la version finale du projet.

Notre groupe *Les Princesses* est composé de 4 étudiants en E1.

## 1.2 Le groupe du projet

- **Julien Bestard (Chef de Projet)**

Julien est un génie de l'informatique originaire de la région 91, comme le célèbre groupe PNL, et mène une vie tranquille dans sa ville de campagne. Depuis le lycée, il est attiré par les nouvelles technologies et s'intéresse de plus en plus à ce domaine.

Étudiant en 2ème année à EPITA, il est prêt à se lancer corps et âme dans ce nouveau projet. Grâce à ses compétences informatiques et son sens de l'humour inégalé, il est le garant du bon déroulement du projet et du maintien d'une bonne ambiance entre nous. Julien est une valeur sûre pour ce groupe !

- **Hippolyte Pik**

Hippolyte Pik est un élève de la E1. C'est la première fois qu'il développe un réseau de neurone c'est donc une totale découverte à faire. Passionné tout particulièrement par l'intelligence artificielle ce projet est pour lui une super opportunité pour la suite de son projet de vie.

- **Léa Bonet**

Léa Bonet est une étudiante en deuxième année à l'EPITA. Elle a beaucoup aimé faire le projet de S2 car il lui a appris beaucoup de compétences en C# mais également à travailler en groupe. Léa avait hâte de démarrer ce projet pour renouveler cette expérience. Elle adore résoudre des sudokus et c'est donc occupée de la partie solver.

- **Benjamin Dreyfus**

Passionné de logique et de programmation, Benjamin (lui aussi élève de la très bonne classe E1) est touche-à-tout et s'intéresse à toute sorte de programme informatique. Le projet de S2 étant une expérience plutôt satisfaisante pour lui, il est prêt à mener un autre projet de ce genre à terme. Dans celui-ci, il s'occupera principalement de l'interface graphique.

## 2 Répartition des tâches

Tasks	Julien	Léa	Hippolyte	Benjamin
Traitement d'image				
Detection de la grille				
Sudoku Solver				
Interface				
Réseau de Neurones				
Site Web (Bonus)				

### 3 Réseau de neurones

#### 3.1 Structure globale

##### 3.1.1 Front Propagation

La propagation vers l'avant (ou en anglais *front propagation*) est une méthode dite de base dans la résolution du problème du réseau de neurones.

C'est une méthode où l'on ne se déplace que dans un seul sens, des neurones d'entrée en passant par les neurones cachés (le cas échéant) et puis, vers la sortie. Cette méthode consiste à envoyer dans chaque neurone de la première couche chacune des entrées du problème.

La propagation vers l'avant se calcule à l'aide d'une fonction d'activation noté  $g^{(n)}$  (l'indice  $n$  indique la couche sur lequel le neurone se situe, ici la  $n$ -ème), et d'une fonction d'agrégation noté ici  $h_j^{(n)}$  (souvent un produit scalaire entre les poids et les entrées du neurone) (l'indice  $j$  indique le numéro du neurone sur la couche, ici le  $j$ -ème) et des poids synaptiques  $w_{jk}$  entre le neurone  $x_k^{(n-1)}$  et le neurone  $x_j^{(n)}$ .

La notation est alors inversée :  $w_{jk}$  indique bien un poids de  $k$  vers  $j$ .

$$x_j^{(n)} = g^{(n)}(h_j^{(n)}) = g^{(n)}\left(\sum_k w_{jk}^{(n)} x_k^{(n-1)}\right)$$

Ainsi avec cette méthode on obtient finalement sur le(s) dernier(s) neurone(s) le(s) résultat(s) attendu(s)

##### 3.1.2 Back Propagation

La propagation vers l'arrière (en anglais *back propagation*) est une méthode où l'on cherche à faire apprendre notre réseau de neurone. Cette méthode comporte ce que ne comporte pas la première méthode présentée, des boucles et des retours en arrière. Elle prend comme base la méthode présentée au dessus. Nous continuons donc le processus après avoir obtenu un résultat noté  $\vec{y}$ .

On calcule alors l'erreur entre la sortie donnée par le réseau  $\vec{y}$  et le vecteur  $\vec{t}$  désiré à la sortie pour cet échantillon. Pour chaque neurone  $i$  dans la couche de sortie, on calcule ( $g'$  étant la dérivée de  $g$ ):  $e_i^{\text{sortie}} = g'(h_i^{\text{sortie}})(y_i - t_i)$

Ensuite on propage l'erreur vers l'arrière  $e_i^{(n)} \mapsto e_j^{(n-1)} e_i^{(n)} \mapsto e_j^{(n-1)}$  grâce à la formule suivante :  $e_j^{(n-1)} = g'^{(n-1)}(h_j^{(n-1)}) \sum_i w_{ij}^{(n)} e_i^{(n)}$

On note que entre la dernière couche et l'avant dernière  $e_i = e_i^{\text{sortie}}$

Et finalement on met à jour les poids dans toutes les couches :  $w_{ij}^{(l)} = w_{ij}^{(l)} - \lambda e_i^{(l)} x_j^{(l-1)}$  où  $\lambda$  représente le taux d'apprentissage (de faible magnitude et compris entre 0,0 et 1,0).

### 3.2 Le xor

Le xor que l'on a implémenté utilise la méthode de back propagation. Pour ce faire nous avons créé un type *Neuron* auquel on a assigné un tableau d'entrée, un tableau de poids, un coefficient d'erreur, et un biais.

La première étape est d'assigner à chacun de nos neurones un poids aléatoire. Cela étant fait, on met à chacune de nos entrées de chacun de nos neurones les entrées de notre programme.

On calcule ensuite la sortie de notre neurone actuel, à l'aide d'une fonction sigmoïde ce qui donne:  $\frac{1}{1 + e^{\sum w_i x_i}}$  que l'on insère directement dans les entrées des neurones de sortie. Nous calculons finalement le résultat à l'aide de la même fonction sigmoïde.

Une fois le résultat obtenu nous calculons le coefficient d'erreur de la sortie grâce à la formule:  $\frac{y}{1 - y}(t - y)$  avec y le résultat obtenu et t le résultat attendu.

On ajuste ensuite les poids et le biais avec la formule:  $w_i = w_i + e_i x_i$  et  $biais = biais + e_i$ . Une fois la sortie ajustée ne reste plus qu'à ajuster les erreurs, les biais et les poids de chacun des neurones restant.

Les coefficients d'erreur sont modifiés avec la formule:  $e_i = \frac{y_i}{1 - y_i} e^{\text{sortie}} w_i$ , cependant les poids et les biais ont la même formule que le neurone de sortie. Voilà, l'explication en détail de notre xor est finie.

## 4 Traitement de l'image

Actuellement, notre traitement d'image n'est pas parfait, mais il nous permet d'obtenir de bons résultats.

Il y a actuellement 4 étapes :

1. Réduction de la taille
2. Gamma
3. Contraste
4. Le niveau de gris
5. Le flou Gaussien
6. La méthode d'Otsu

Prenons l'image ci-dessous comme exemple :

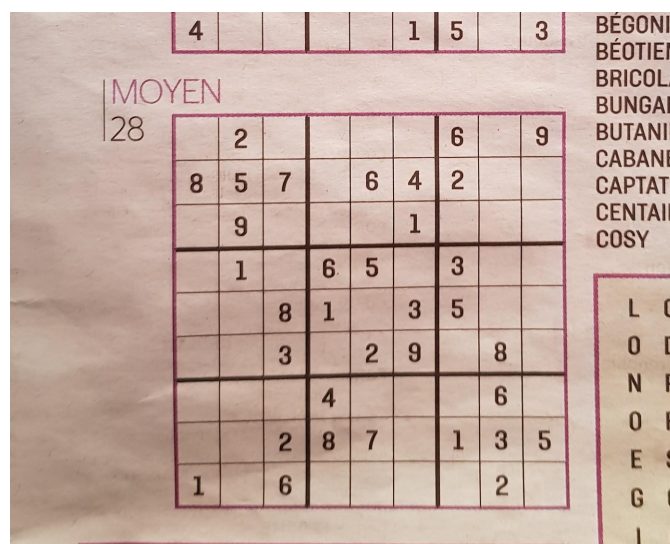


Figure 1: Image 04

## 4.1 Réduction de la taille

Si une image est trop grande, nous réduisons la taille de celle-ci pour le temps de traitement de celle-ci.

## 4.2 Ajustement des couleurs

### 4.2.1 Gamma

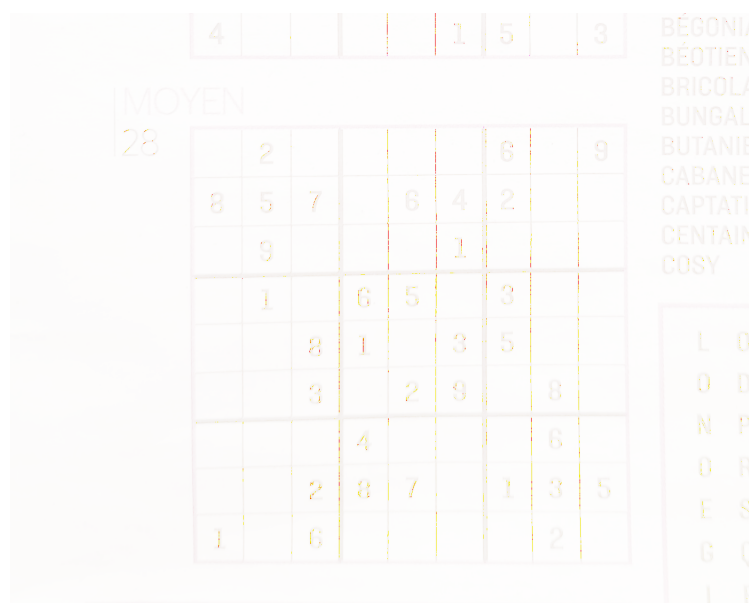


Figure 2: Après l'ajustement du gamma

Après avoir éliminé le bruit, nous appliquons un filtre gamma. Ce filtre, combiné aux précédents, nous permet d'avoir une réduction de bruit efficace, améliorant donc la détection des bords. Le gamma définit la relation entre la valeur des couleurs d'un pixel et sa luminosité actuelle.

Nous calculons sa valeur grâce à cette formule :  $nC = 255 * \frac{C}{255}^\gamma$ , avec ici  $\gamma = \frac{2}{5}$  et  $c$  la valeur du rouge ou vert ou bleu.



## 4.2.2 Contrast

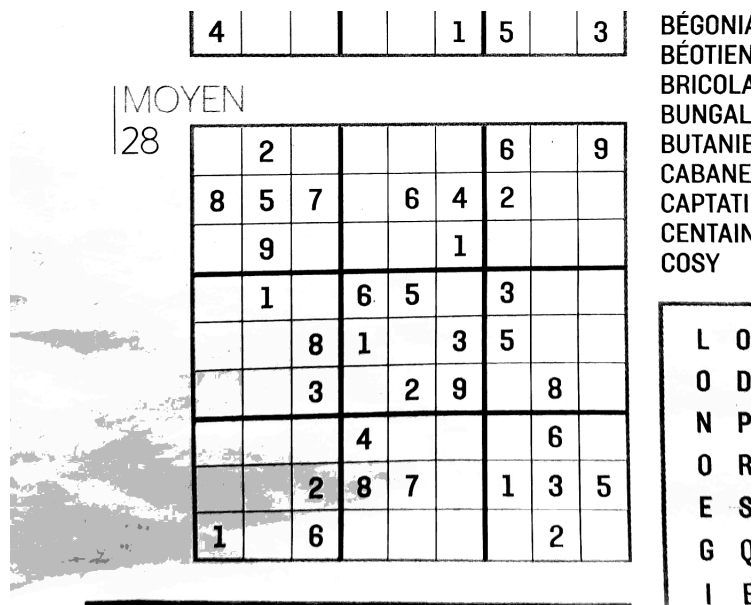


Figure 3: Après l'ajustement du contraste

Nous avons utilisé le contraste ici pour augmenter la lisibilité de la date sur l'image et faire ressortir les contours. Nous utilisons une formule mathématique pour calculer la correction du contraste :  $F = \frac{259 \cdot (C + 255)}{255 \cdot (C - 259)}$

Le  $C$  est le facteur du contraste souhaité . Il sera utilisé pour modifier la valeur du rouge/bleu/vert avec cette formule :  $Color = F \cdot (Color - C) + C$

La combinaison de ces deux procédés nous permet d'obtenir une image plus détaillée, et avec un peu moins de bruit.

### 4.2.3 Niveau de gris

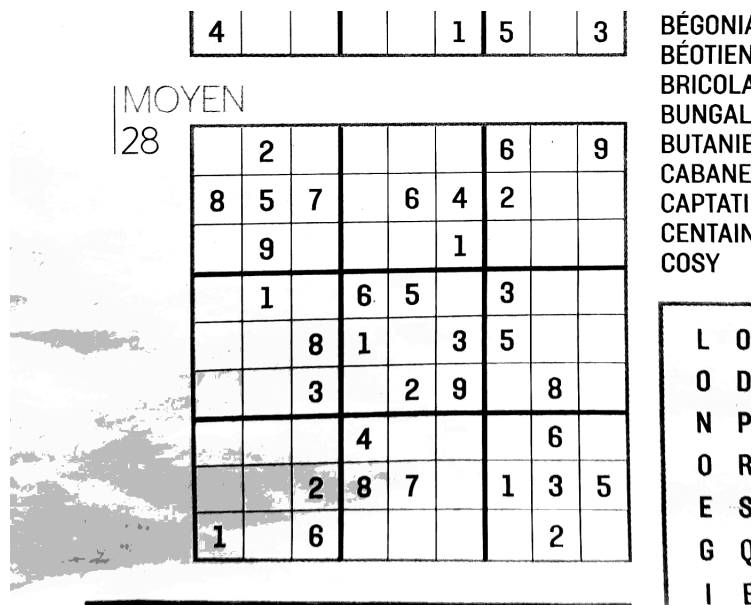


Figure 4: Après le niveau de gris

Le niveau de gris est la première étape de notre traitement d'image. De ce fait, nous n'aurons plus qu'à traiter un seul taux de couleur, et plus du RGB.

Pour le faire on applique cette fonction :  $gris = 0.3 * r + 0.59 * g + 0.11 * b$ .

La valeur "gris" remplacera chaque valeur de notre RGB pour le pixel actuel, qui deviendra donc gris.

Nous utilisons la méthode pondérée car nous sommes sensibles aux lumières dans cet ordre : vert, rouge, bleu.

### 4.2.4 Gaussian Blur

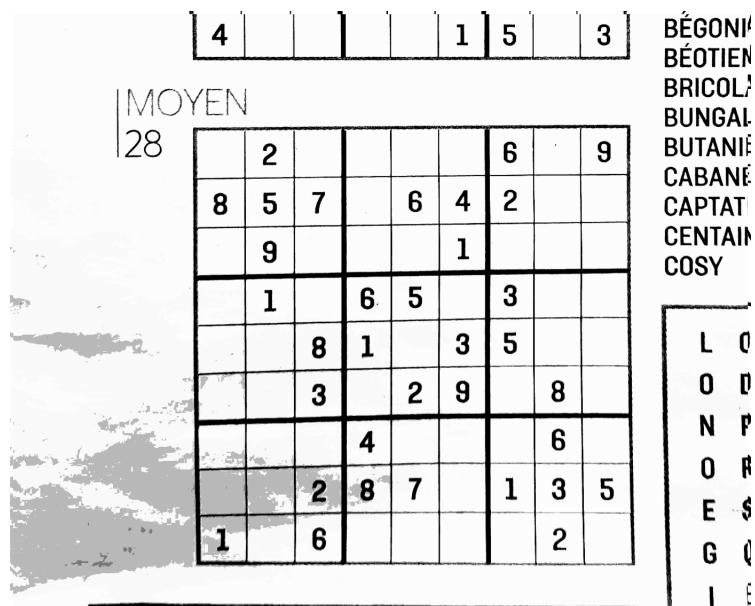


Figure 5: Après le flou

Après avoir fait le niveau de gris, nous appliquons le flou Gaussien, aussi connu sous le

nom de lissage gaussien. Le flou Gaussien, qui tire son nom du mathématicien Carl Friedrich Gauss, est l'application d'une fonction mathématique à une image pour la flouter. Ce filtre de type passe-bas, lisse les valeurs inégales des pixels d'une image en supprimant les valeurs aberrantes extrêmes.

Ce filtre permet, pour une photo prise avec une luminosité faible et qui présente beaucoup de bruit, d'atténuer ceux-ci. Si vous voulez ajouter du texte sur une image, un flou gaussien permettra d'adoucir celle-ci pour le faire ressortir avec plus de netteté.

Ici il nous est particulièrement utile pour éliminer le bruit de l'image et l'adoucir un peu.

Pour calculer la transformation à appliquer à chaque pixel de l'image nous utilisons la formule pour une image à 2 dimensions :  $G(x, y) = \frac{1}{2 \cdot \pi \cdot \sigma^2} e^{-\frac{x^2 + y^2}{2 \cdot \sigma^2}}$

Où  $x$  est la distance par rapport à l'origine sur l'axe horizontal,  $y$  est la distance par rapport à l'origine sur l'axe vertical, et  $\sigma$  est l'écart type de la distribution gaussienne. Les indices de cette distribution sont utilisés pour construire une matrice de convolution qui est appliquée à l'image originale.

Ce processus de convolution est illustré visuellement dans la figure de droite. La nouvelle valeur de chaque pixel est définie comme une moyenne pondérée du voisinage de ce pixel. La valeur du pixel d'origine reçoit la pondération la plus forte (ayant la valeur gaussienne la plus élevée) et les pixels voisins reçoivent des pondérations plus faibles à mesure que leur distance au pixel d'origine augmente. Le résultat est un flou qui préserve mieux les limites et les bords que d'autres filtres de flou plus uniformes.

#### 4.2.5 Méthode d'Otsu

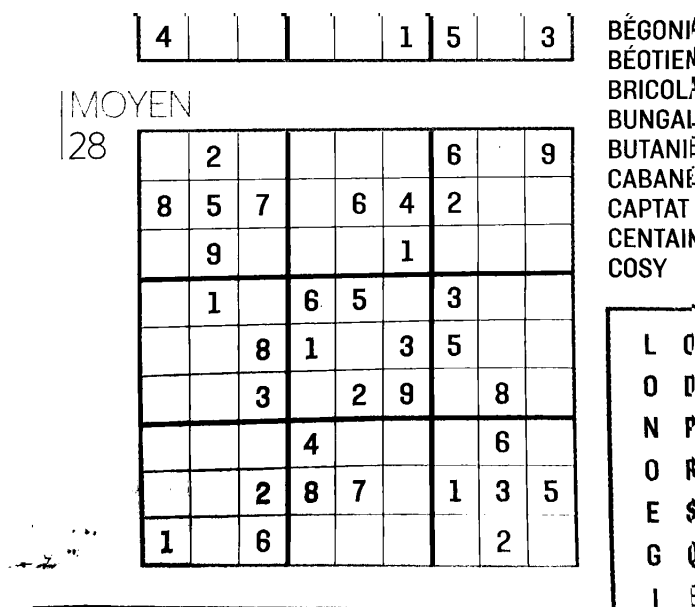


Figure 6: Après OTSU

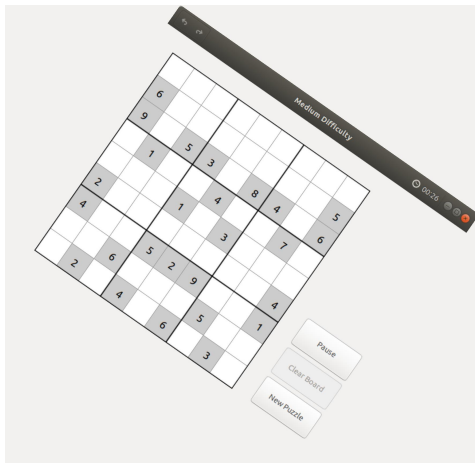
Afin de transformer notre image en noir et blanc, nous utilisons la méthode d'Otsu. La méthode d'Otsu est utilisée pour effectuer un seuillage automatique à partir de la forme de l'histogramme de l'image, ou la réduction d'une image à niveaux de gris en une image binaire. L'algorithme suppose alors que l'image à binariser ne contient que deux classes de pixels, (c'est-à-dire le premier plan et l'arrière-plan). L'algorithme s'exécute de cette façon : il recherche

itérativement le seuil qui minimise la variance intra-classe, définie comme la somme pondérée des variances de deux classes (arrière-plan et premier plan). Les couleurs en niveaux de gris sont généralement comprises entre 0 et 255. Ainsi, si nous choisissons un seuil de 150, tous les pixels dont la valeur est inférieure à 150 deviennent l'arrière-plan et tous les pixels dont la valeur est supérieure ou égale à 150 deviennent le premier plan.

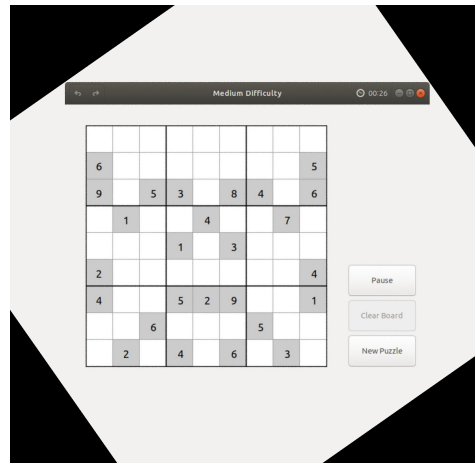
### 4.3 Rotation de l'image

Pour faire pivoter une image, nous avons décidé d'utiliser une matrice de rotation. Cette méthode est rapide et précise, elle nous donne des résultats sans bords irréguliers et nous permet d'effectuer des rotations avec des angles très précis. La matrice de rotation s'exprime par les formules suivantes pour trouver les coordonnées des pixels tournés  $(x, y)$  par l'angle  $\theta$  sur l'image originale  $(x', y')$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \cdot \cos \theta - y \cdot \sin \theta \\ x \cdot \sin \theta + y \cdot \cos \theta \end{pmatrix}$$



(a) Avant rotation



(b) Après rotation de 35

## 5 Detection des bords

### 5.1 L'opérateur Sobel

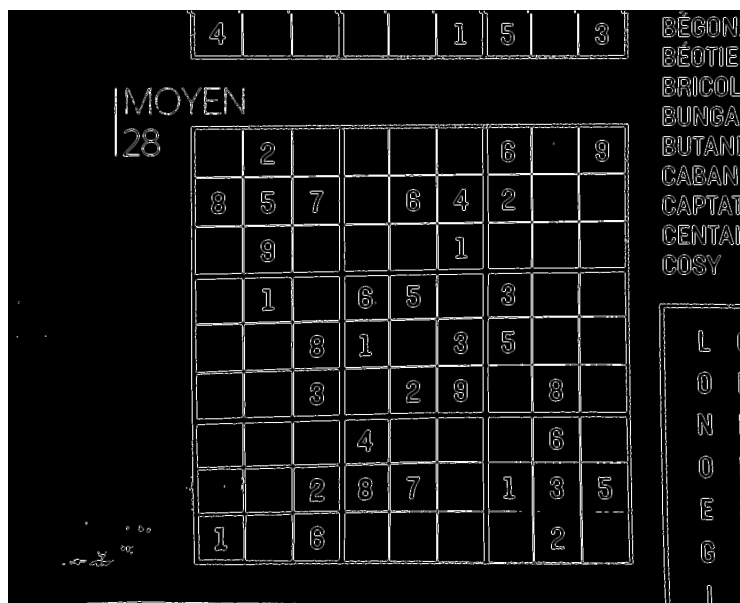


Figure 8: Après l'opérateur Sobel

Le première chose que nous ferons pour la détection de bords sera d'utiliser l'opérateur Sobel. L'opérateur de Sobel utilise deux noyaux (un pour chaque direction) :

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ et } K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

On calcule la convolution entre l'image (convertie en noir et blanc) et les deux noyaux séparément. Cela nous donne, pour chaque pixel, les valeurs  $mag_x$  et  $mag_y$ . La valeur du pixel courant est fixée à  $\sqrt{(mag_x)^2 + (mag_y)^2}$ .

## 6 Résolution d'une grille

### 6.1 Le Solveur

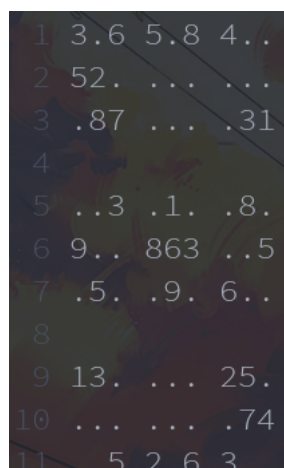
#### 6.1.1 Le principe

Pour résoudre une grille de sudoku on va utiliser la méthode du *backtracking* (retour sur trace). Cette méthode consiste à revenir en arrière sur les décisions prises afin de sortir d'un blocage. Dans le cas du sudoku il s'agit de tester toutes les valeurs pour chaque case vide en vérifiant si la grille reste valide. Dès qu'on arrive sur une valeur où la grille n'est pas valide on revient en arrière. Pour rappel, chaque lignes, colonnes et carrées de 3x3 doit contenir un chiffre entre 1 et 9 mais ne doit pas contenir deux fois la même valeur.

Dans le programme, nous allons d'abord vérifié que la valeur que l'on souhaite ajouter n'est pas déjà présente sur la ligne, la colonne et le carrée de 3x3. Puis dans la fonction principale, on va énumérer les valeurs possibles et les tester. On va appeler notre fonction récursive pour pouvoir retourner en arrière si la valeur est incorrecte.

#### 6.1.2 Utilisation du programme

L'utilisation du solveur est simple, celui ci prends en paramètre un fichier contenant une grille disposée sous cette forme (les points représentant les cases à combler):



```

1 3.6 5.8 4..
2 52. ... ..
3 .87 ... .31
4
5 ..3 .1. .8.
6 9.. 863 ..5
7 .5. .9. 6..
8
9 13. ... 25.
10 ... ... .74
11 ..5 2.6 3..
  
```

Figure 9: Gille 00 non résolue

Une fois fini, le programme écrit dans un fichier `<nom_de_la_grille>.result`, la grille résolue.

## 7 Ce qui sera implémenté

Voici ce qui n'a pas encore été implémenté mais le sera une fois le projet terminé :

### 7.1 Grilles et cases

La détection des bords étant implémentée par l'opérateur Sobel, il nous reste plus qu'à trouver la grille et découper en cases de 28x28 pour ensuite envoyer les datas au réseau de neurones. Pour ce faire nous nous sommes pas encore décidé sur quel algorithme nous allons partir, nos choix tendent plus vers *hough transform* ou le *connect component labelling*.

### 7.2 Interface Graphique

N'ayant pas terminé l'interface graphique, elle n'est pas prête pour cette soutenance, mais le sera pour la soutenance finale une fois le projet terminé. Cette interface sera implémentée grâce au logiciel Glade.

## 8 Conclusion

En conclusion, le projet ne cesse de progresser à un bon rythme, les principaux aspects du projet sont déjà présents et la plupart sont terminés.

Une bonne synergie de groupe se met en place, ce qui nous permettra de terminer le projet correctement avant la date butoir. Nous espérons que notre travail vous plaira.