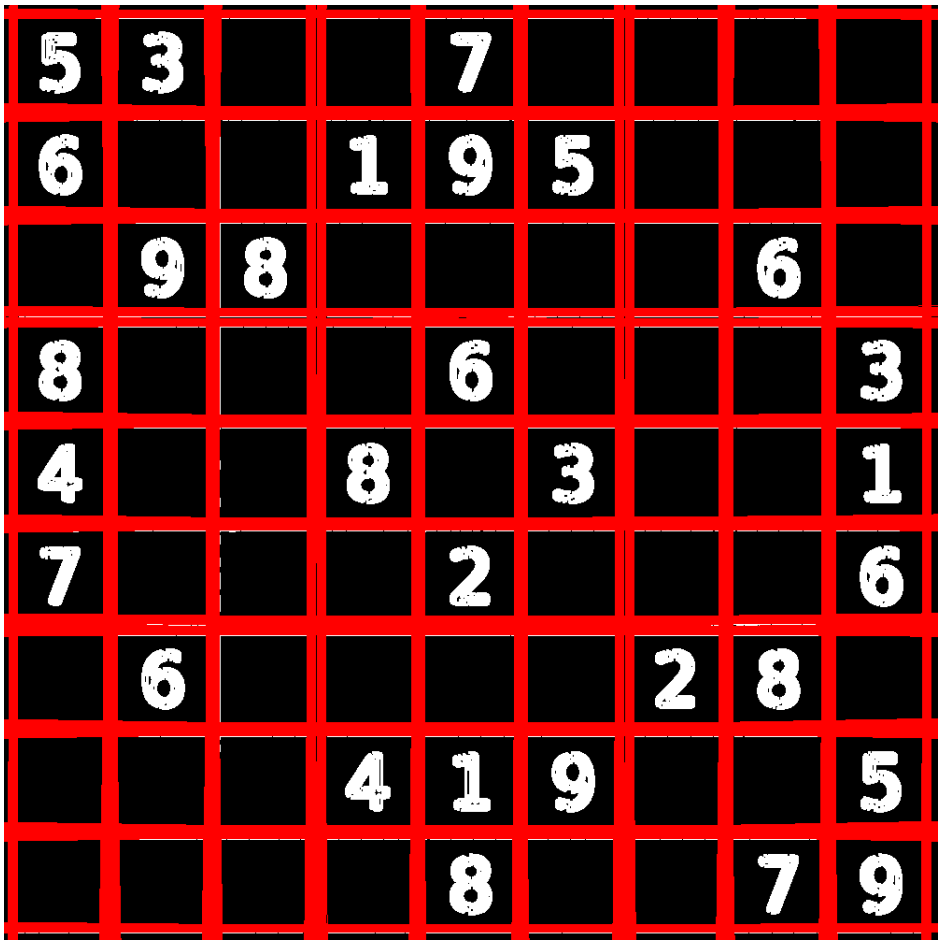
OCR - Rapport 2

Soutenance de Projet

Décembre 2023



Erwan LE GRAND - Alexis LATOURNERIE
Thomas GRAVELINE-MERCIER - Nathan CHAMPAGNE

Table des matières

1 Introduction

1.1 Présentation du groupe

Notre groupe est composé des 4 mêmes membres que lors du projet du S2. Ce choix est dû à la cohésion de groupe et le sérieux dont nous avions tous fait part. Il était donc naturel de se remettre ensemble pour ce projet d'OCR.

Nous nous sommes répartis les tâches en fonction de nos préférences pour ce projet, mais aussi de nos capacités. Il était important de nous voir régulièrement et de travailler dans un même espace pour que l'on puisse avancer de manière coordonnée, c'est pourquoi nous nous sommes rejoints de nombreuses fois à EPITA pour continuer l'avancement du projet.

1.2 Description d'un OCR et du projet

Un OCR, ou Reconnaissance Optique de Caractères, est une technique qui, à partir d'un procédé optique, permet à un système informatique de lire et de stocker de façon automatique du texte dactylographié, imprimé ou manuscrit sans qu'on ait à retaper ce dernier.

Ce procédé permet donc de numériser un document, c'est-à-dire de le posséder physiquement et de le retrouver numériquement exactement conforme à celui physique. Cette technologie repose sur des algorithmes de traitement d'image et de reconnaissance de caractères, qui sont conçus pour identifier les caractères individuels présents dans l'image et les convertir en une forme de texte codé.

Nous utilisons de nombreux OCR dans la vie de tous les jours, car dès que, par exemple on scanne ou numérise un livre, la technologie sous-jacente à ces fonctionnalités est un OCR.

Le projet consiste à réaliser un OCR sur une image de grille de sudoku afin de reconnaître les chiffres, de résoudre le sudoku et d'afficher le résultat pour l'utilisateur.

2 Répartition des charges

2.1 Erwan Le Grand

La première grande étape pour détecter les lignes de la grille de sudoku de manière fiable est d'appliquer des traitements adaptés sur l'image et de réduire au maximum l'impact des défauts des images sur la détection des lignes.

2.2 Nathan Champagne

Pour réaliser un OCR, il faut développer une intelligence artificielle capable de détecter et reconnaître des chiffres. Cette intelligence artificielle doit pouvoir s'entraîner sur une base de données d'images et pouvoir prédire le chiffre contenu dans une image.

2.3 Alexis Latournerie

Dans un OCR, il faut aussi gérer la rotation et le redimensionnement de l'image. Il est aussi nécessaire de détecter la grille et les cases après le traitement de l'image, la détection des lignes et la simplification de celles-ci. De plus, il a été nécessaire de créer une gestion par threads multiple, afin que les calculs s'exécutent en arrière-plan sans bloquer l'interface de l'utilisateur.

2.4 Thomas Graveline–Mercier

Un autre pré-requis est de réaliser un algorithme qui résout un sudoku, appelé solveur, de manière complète et fonctionnelle, en 9x9 ainsi qu'en hexadécimal (en 16x16).

Il est aussi important de faire une interface graphique pour pouvoir interagir avec tous les éléments créés lors du processus de l'OCR.

3 Finalisation du projet

3.1 Traitement de l'image et détection des lignes

Les différentes étapes de pré-traitement de l'image et de détection des lignes ont été implémentées en totalité :
- chargement d'une image
- suppression des couleurs
- amélioration du contraste
- réduction du bruit
- détection des contours
- détection des lignes
- simplification des lignes

Cependant il reste quelques ajustements à réaliser au niveau de la réduction du bruit et de la détermination des seuils de détection afin que ces derniers soit automatiquement géré en fonction du niveau de bruit de l'image.

3.2 Rotation de l'image

La rotation manuelle de l'image est complètement implémentée. Elle permet d'effectuer une rotation depuis n'importe quel angle saisi par l'utilisateur. Elle s'effectue aussi automatiquement une fois les lignes directrices de l'image détectée.

3.3 Détection de la grille et de la position des cases

La détection de la grille et de la position des cases est implémentée. Nous avons aussi réalisé une correction de la perspective de l'image, en fonction des quatre coins de la grille détectée.

3.4 Réseau de neurones

Nous avons commencé la réalisation du réseau de neurones par comprendre tout le procédé mathématique de ce dernier ainsi que son vocabulaire. Tout de suite, nous avons trouvé très judicieux de travailler avec des matrices pour pouvoir travailler en groupe de neurones, couche après couche de manière très simple au lieu de devoir travailler neurone

par neurone de manière individuelle.

3.5 Algorithme de résolution du sudoku - Solveur

La résolution du sudoku a été conçue depuis le début de la validation des groupes de l'OCR et s'est poursuivie tout du long de la durée jusqu'à la première soutenance. La méthode de résolution a été réfléchie et repensée plusieurs fois pour permettre d'avoir un solveur efficace et optimisé.

Ainsi, nous avons développé plusieurs solveurs fonctionnels, mais qui ne correspondaient pas à nos attentes. Lorsque la méthode finale a été trouvée, une phase de résolution manuscrite a pris place.

Il s'agissait de bien visualiser comment s'y prendre pour réaliser ce solveur et ne pas perdre de temps dessus. Les premiers solveurs n'étant aucunement une perte de temps, car ils ont permis d'avoir des fonctions basiques réutilisables, mais aussi un apprentissage de comment il faut s'y prendre pour résoudre un sudoku.

Notre solveur final à ainsi directement pu etre implémenté pour résoudre un sudoku normal (en 9x9) et en hexadécimal (en 16x16).

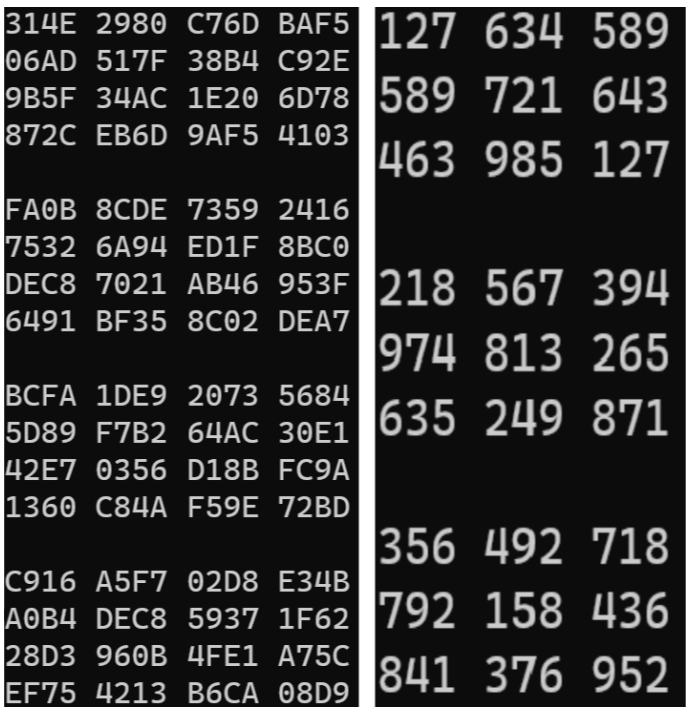


Figure 1 – Différence entre un hexadoku et un sudoku

4 Aspects techniques

4.1 Rotation manuelle de l'image

Il est important de pouvoir effectuer une rotation sur les images en fonction de n'importe quel angle. Pour ce faire nous avons implémenté une rotation par "shearing". Cela consiste à faire un produit entre les coordonnées et les fonctions de trigonométrie. On agrandit l'image pour garder le même nombre de pixels.

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

Mais cette méthode comporte un défaut, pour les angles proches des diagonales de l'image, cela crée beaucoup de crénelage (aliasing). Cela vient du fait de simplifier des cosinus et des sinus en nombre entier pour en faire des coordonnées. À cause de cela beaucoup de points ou de motifs noirs apparaissent à l'image, car aucun pixel n'a été trouvé pour ces coordonnées.

Pour pallier cela, nous avons effectué cette transformation de manière plus précise en appliquant la méthode "3 shears rotation". Cette méthode est identique à la première, mais au lieu de multiplier par une matrice de transformation, on effectue le produit de 3 matrices de transformation pour chaque pixel. Cela permet de ne plus avoir l'effet de crénelage et d'être plus précis.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & -tan(\theta/2) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ sin\theta & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -tan(\theta/2) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Nous pouvons donc grâce à cette méthode, effectuer la rotation depuis n'importe quel angle compris entre -90° et 90°. Afin d'effectuer la rotation pour tous les angles possible, si nous ne sommes pas compris entre ces deux angles, nous retournons d'abord l'image (rotation à 180°) en inversant les coordonnées des pixels puis appliquons la méthode de rotation "3 shears rotation".

Afin de travailler avec une image sans bordure de vide, nous prenons ensuite, le plus grand rectangle possible de l'image où la rotation a été effectuée.

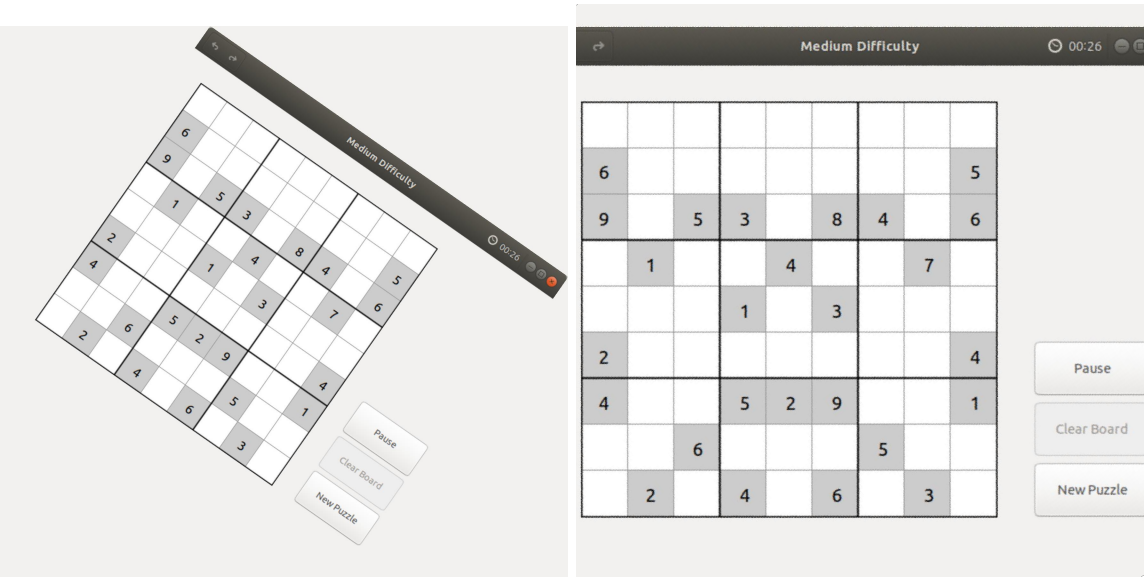


FIGURE 2 – Rotation de l'image suivant un angle de 35°

4.2 Traitement de l'image

Pour le traitement des images, nous avons opté pour la détection de contours de Canny, qui se décompose en plusieurs étapes que nous allons expliquer dans les sections suivantes.

4.2.1 Chargement d'une image et suppression des couleurs

La première étape est de charger une image et de la convertir dans le format souhaité pour le traitement d'image. En l'occurrence, notre traitement d'image utilise le format SDL_PIXELFORMAT_RGBA8888, c'est-à-dire que chaque pixel est encodé sur 32 bits avec ses 4 composantes R, G, B, et A chacune encodée sur 8bits.

Afin que le traitement d'image fonctionne dans un temps raisonnable sur des images de grandes tailles nous avons mis en place un redimensionnement automatique afin de garder des dimensions inférieures ou égales à 1500 pixels en hauteur et en largeur tout en conservant les proportions de l'image bien sûr.

Ensuite, nous convertissons l'image en nuances de gris en calculant pour chaque pixel sa luminance

$$L = 0.2126R + 0.7152V + 0.0722B$$

Puis nous remplaçons les composantes R, G, et B du pixel par sa luminance L et nous mettons la composante A à 255.

4.2.2 Amélioration du contraste

Ensuite, on améliore le contraste de l'image. Pour cela, on génère l'histogramme de l'image qui contient pour chaque intensité $i \in [\![0, 255]\!]$ le nombre de pixels d'intensité i. Notons N_i le nombre de pixels d'intensité i. On cherche alors i_{min} et i_{max} tel que $\forall i \in [\![0, i_{min}[\![\cup]\!]i_{max}, 255]\!], N_i = 0$. On a alors $[\![i_{min}, i_{max}]\!]$ qui correspond à la dynamique de l'image initiale. Ensuite, pour chaque pixel d'intensité i, on applique une fonction de transformation linéaire afin d'obtenir sa nouvelle intensité i'.

$$i' = \frac{255}{i_{max} - i_{min}}(i - i_{min})$$

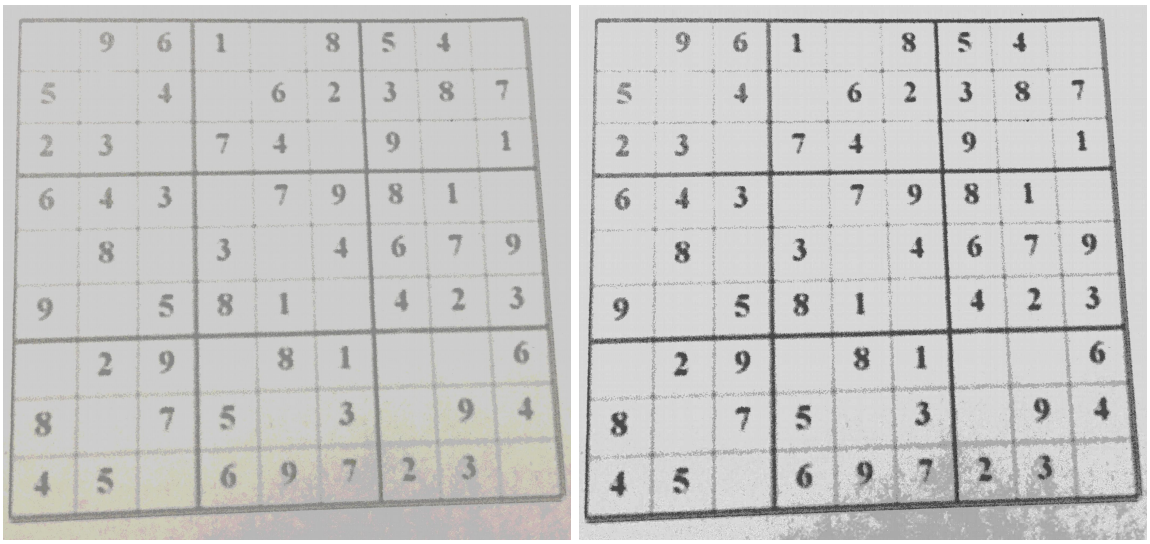


FIGURE 3 – Suppression des couleurs et amélioration du contraste.

4.2.3 Filtre médian

Pour réduire le bruit potentiel de l'image, on applique un filtre médian. Le principe est simple, pour chaque pixel de l'image, on construit une liste qui contient la valeur du pixel courant et les valeurs de ses pixels voisins dans un rayon de 1 pixel. Ensuite, on trie cette liste (avec un algorithme de tri par insertion) et on cherche la valeur médiane qui correspond à la nouvelle valeur du pixel courant.

Par exemple, prenons le voisinage d'un pixel dans un rayon de 1 pixel : $\begin{bmatrix} 3 & 2 & 3 \\ 4 & \boxed{7} & 9 \\ 5 & 1 & 8 \end{bmatrix}$

Le pixel courant est le pixel au centre de la matrice.
On obtient ensuite cette liste triée : $\begin{bmatrix} 1 & 2 & 3 & 3 & \boxed{4} & 5 & 7 & 8 & 9 \end{bmatrix}$
La nouvelle valeur du pixel courant est donc 4.

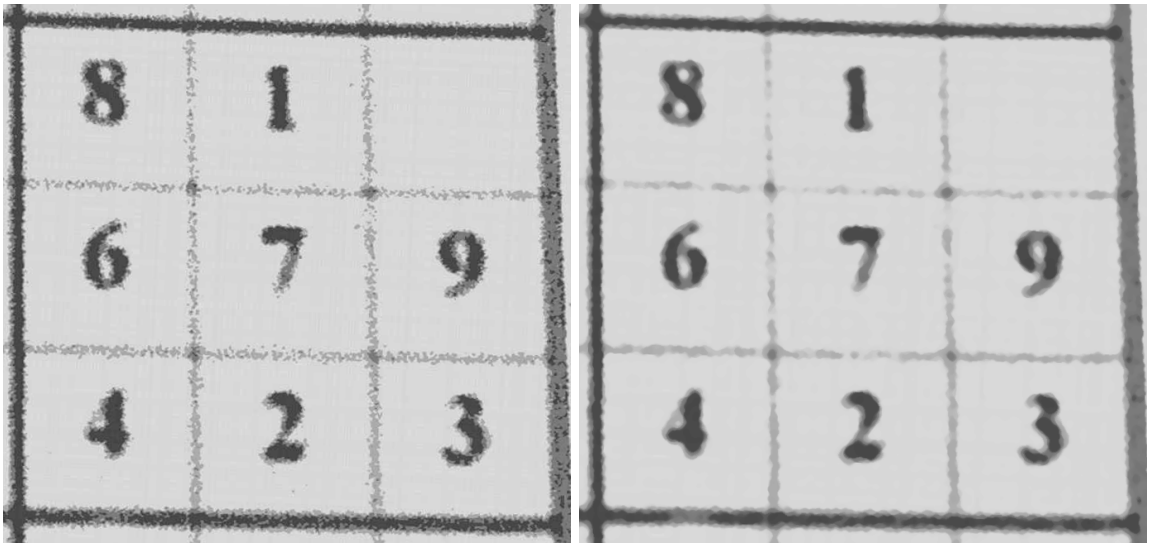


FIGURE 4 – Filtre médian : avant / après

4.3 Filtre Bilatéral

Le filtre bilatéral est une technique de traitement d'image utilisée pour réduire le bruit tout en préservant les contours. Il combine deux composantes principales : la similarité spatiale et la similarité d'intensité.

Le filtre bilatéral agit en filtrant une image I en fonction de la similarité spatiale et de la similarité d'intensité. La formule générale du filtre bilatéral pour chaque pixel de l'image est donnée par :

$$I'(x,y) = \frac{1}{W_p} \sum_{(i,j)\in N} I(i,j) \cdot w(i,j,x,y)$$

Où :
— $I'(x,y)$ est la valeur du pixel filtré,
— W_p est la normalisation,
— N est le voisinage du pixel (x,y),
— $w(i,j,x,y)$ est la fonction de poids basée sur la similarité spatiale et d'intensité.
La fonction de poids w est donnée par :

$$w(i,j,x,y) = \omega_s \cdot \exp\left(-\frac{(i-x)^2+(j-y)^2}{2\sigma_s^2}\right) \cdot \exp\left(-\frac{\|I(i,j)-I(x,y)\|^2}{2\sigma_r^2}\right)$$

Où :
— ω_s est le poids spatial,
— σ_s contrôle la décroissance de la similarité spatiale,
— σ_r contrôle la décroissance de la similarité d'intensité.
Ci-dessous, deux images illustrant l'effet du filtre bilatéral avant et après son application.

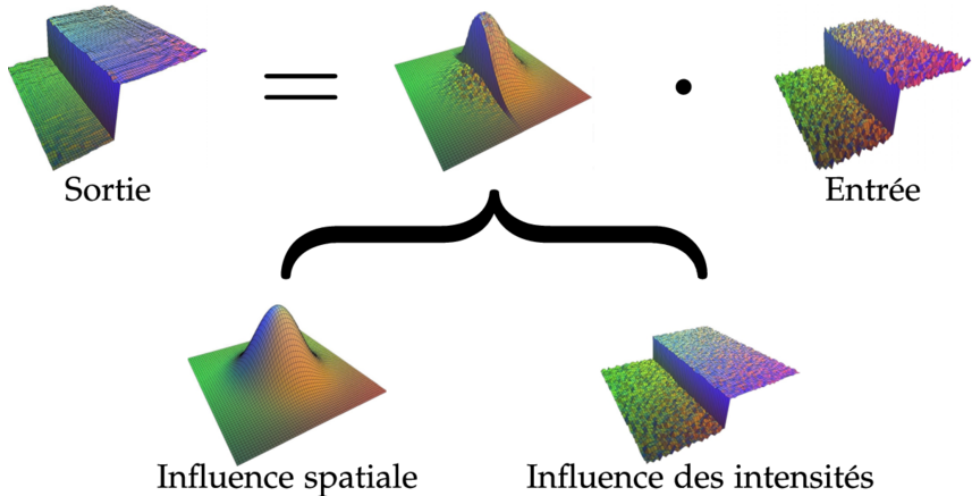


FIGURE 5 – Visualisation de l'influence spatiale et de l'incluence des intensités

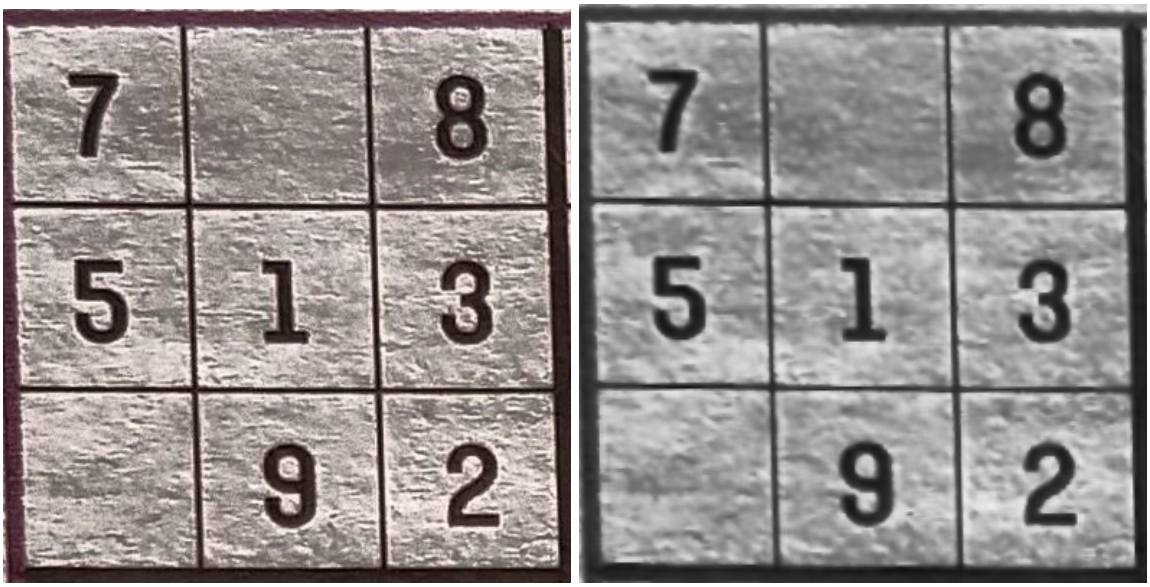


FIGURE 6 – Filtre bilatéral : avant / après

4.3.1 Filtre de Sobel

Le filtre de Sobel permet d'obtenir l'intensité et l'orientation des contours. Pour cela, on génère deux images :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

G_x et G_y correspondent respectivement à l'approximation de la dérivée horizontale et verticale de l'image initiale A. Ensuite, on peut obtenir une approximation de la norme/intensité du gradient ainsi que son orientation :

$$G = \sqrt{G_x^2 + G_y^2} \qquad \Theta = \textbf{atan2}(G_y, G_x)$$

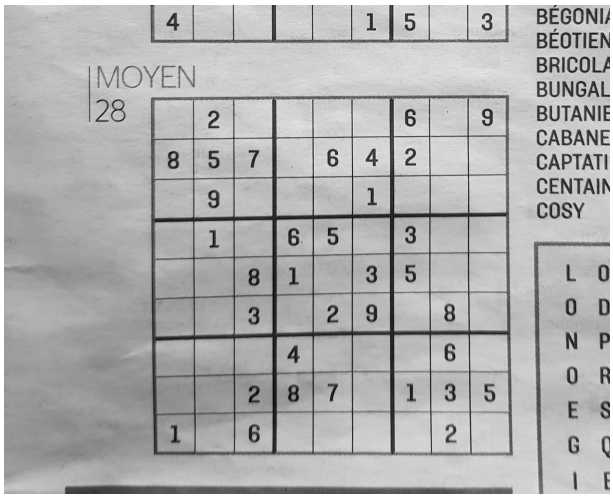


FIGURE 7 – Image d'origine

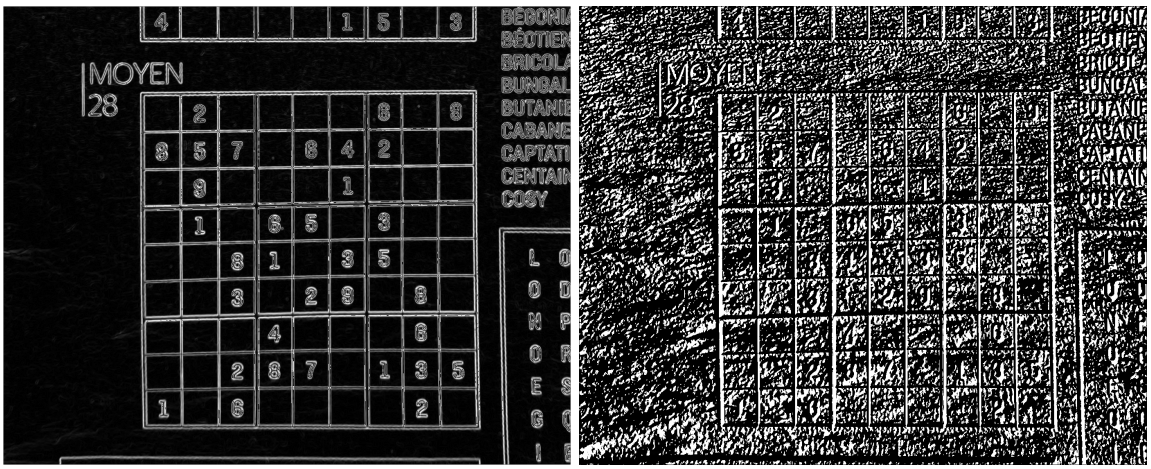


FIGURE 8 – Intensité du gradient / Orientation du gradient

4.3.2 Seuillages

4.3.2.1 Suppression des non-maxima

On ne peut pas déterminer si un point appartient à un contour seulement à partir de l'intensité du gradient en ce point. En effet, une forte intensité nous indique qu'il y a une forte probabilité de présence de contours, mais ce n'est pas suffisant pour se décider. Nous nous aidons donc de l'orientation du gradient calculée au préalable afin de supprimer les non-maxima.

4.3.2.2 Seuillage des contours

Après avoir supprimé les non-maxima, nous utilisons un seuillage à hystérésis qui nécessite de classer les pixels dans trois catégories. On utilise d'abord la méthode d'Otsu afin de trouver un seuil haut s_1 adapté à l'image. Puis, nous définissons le seuil bas

$s_2 = 0.5s_1$. Ensuite, pour chaque pixel dans l'image du gradient d'intensité, on compare son intensité avec les deux seuils. Notons i l'intensité d'un pixel :

— si $i > s_1$ alors le pixel sera catégorisé comme fort et on met la valeur du pixel à 255 (pixel blanc)
— si $s_2 < i \leq s_1$ alors le pixel sera catégorisé comme faible.
— si $i \leq s_2$ alors on met la valeur du pixel à 0 (pixel noir).

Enfin, pour chaque pixel faible, si un de ses voisins (directs ou en diagonale dans un rayon de 1 pixel) est un pixel fort, alors le pixel est mis en blanc sinon il est mis en noir. À l'issue de cette dernière étape du filtre de Canny, nous obtenons une image en noir et blanc.

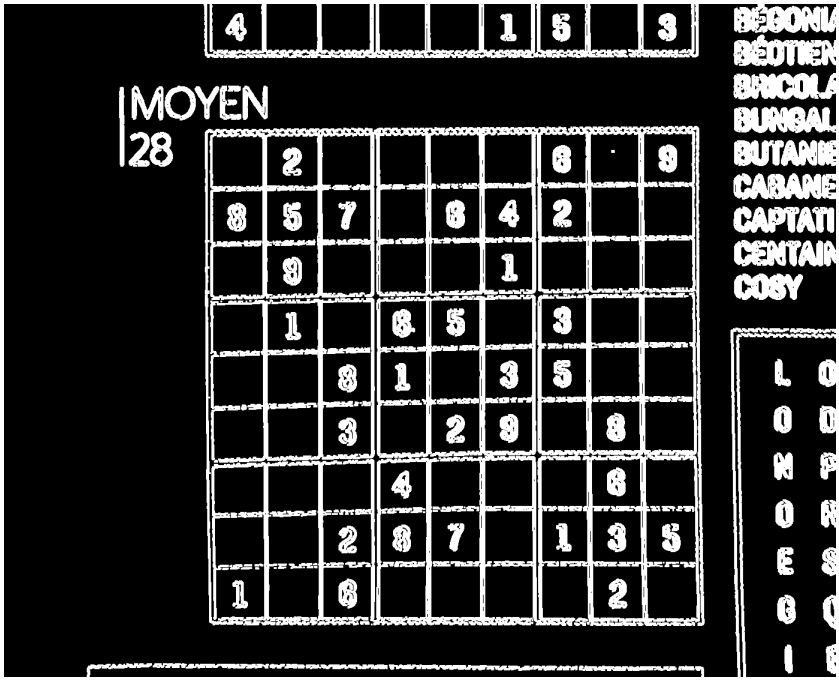


Figure 9 – Seuillage à hystérésis

4.4 Détection de la grille et de la position des cases

4.4.1 Transformée de Hough

Pour détecter les lignes de notre grille de sudoku, nous utilisons la transformée de Hough. Pour en expliquer simplement le principe, chaque droite dans l'image peut être caractérisée par deux paramètres : ρ et θ où ρ est la distance de la droite à l'origine du repère et θ est l'angle que fait la perpendiculaire à la droite avec l'axe x (système de coordonnée polaire).

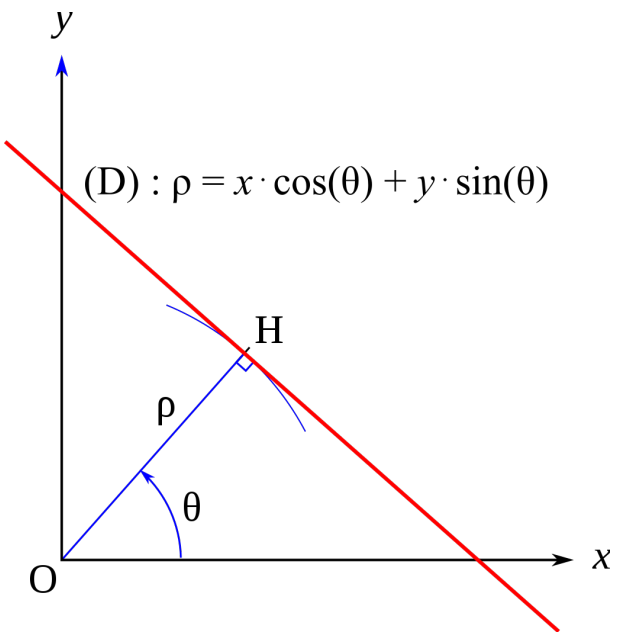


FIGURE 10 – Passage en coordonnée polaire pour les droites

Tout d'abord, on créer un accumulateur (une sorte d'histogramme) qui représente l'espace des paramètres ρ et θ. Une droite dans l'image correspond donc à un point dans l'espace des paramètres. Dans notre image avec les contours détectés préalablement, pour chaque pixel blanc de coordonée (x, y), on cherche toutes les droites passant par ce point. Pour cela, on fait varier θ entre 0 et π et on calcul $\rho = x \times \cos\theta + y \times \sin\theta$ puis on incrémente la valeur de l'accumulateur aux coordonnées (θ, ρ).

Une fois l'accumulateur généré, on le parcours une première fois pour chercher la valeur maximum de l'accumulateur. Puis, on le parcours une seconde fois et on regarde si la valeur courante est supérieur à un certain pourcentage du maximum. Si oui, les coordonnées (θ, ρ) de cette valeur dans l'accumulateur nous permettent d'identifier une droite dans l'image ayant pour équation :

$$y = (-\frac{\cos\theta}{\sin\theta})x + (\frac{\rho}{\sin\theta})$$

Pour dessiner les lignes sur l'image nous avons utilisé l'algorithme de Bresenham.

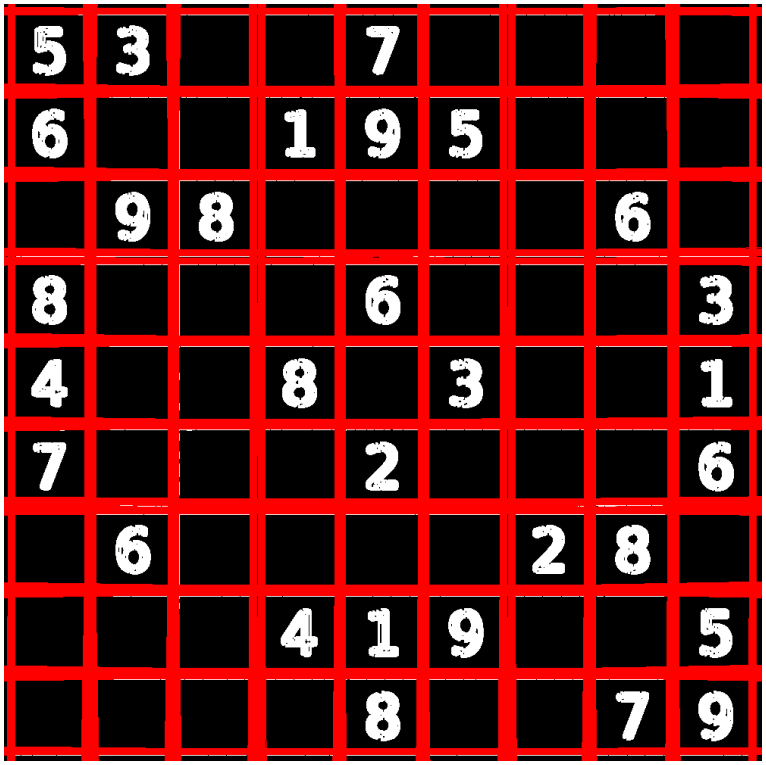


FIGURE 11 – Lignes détectées

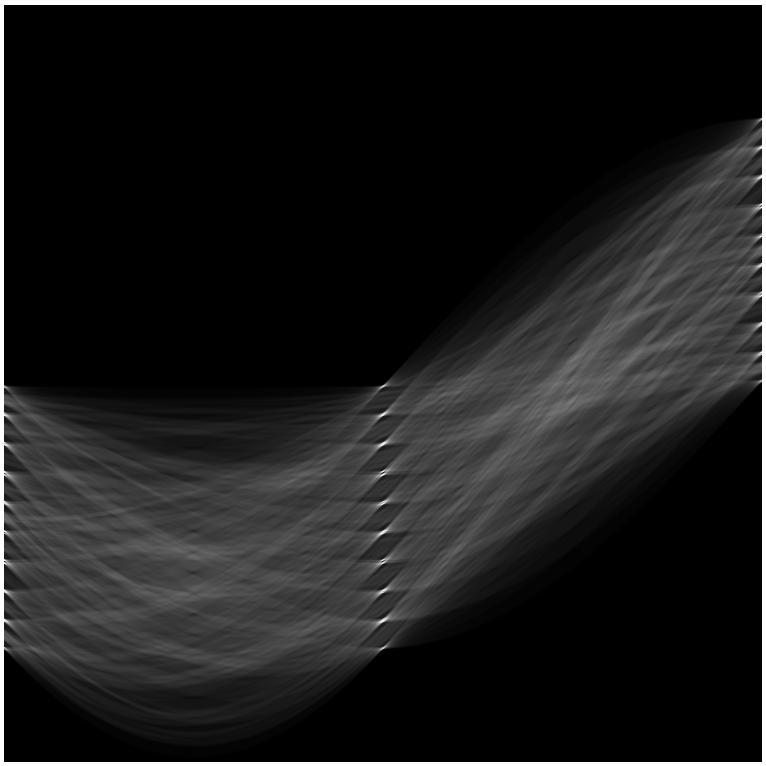


FIGURE 12 – Visualisation de l'accumulateur

4.4.2 Rotation automatique de l'image

La rotation automatique récupère l'angle des lignes directrices détectés auparavant et applique la fonction de rotation détaillé plus haut.

4.4.3 Simplification des lignes

Au moment du parcours des valeurs de l'accumulateur, si des valeurs sont supérieures au seuil de détection et que leur coordonnées dans l'accumulateur sont proches, nous les fusionnons afin de réduire le nombre de lignes détectées. Une fois qu'il ne reste plus que quelques lignes, et que celles-ci ont la direction des lignes de la grille, il peut en rester quelques-unes qui avaient la même direction à cause d'autres éléments présents dans l'image. Tout d'abord nous trions ces lignes. Afin de les supprimer, tant que nous n'avons pas le bon nombre de lignes (10 horizontales et 10 verticales), nous calculons les distances entre les droites, puis nous supprimons la première ou la dernière en fonction de celle qui est la plus éloignée de la moyenne des distances ou des votes récupérés par la transformé de Hough quand l'écart entre les deux lignes les plus éloignés est trop faible pour prendre une décision.

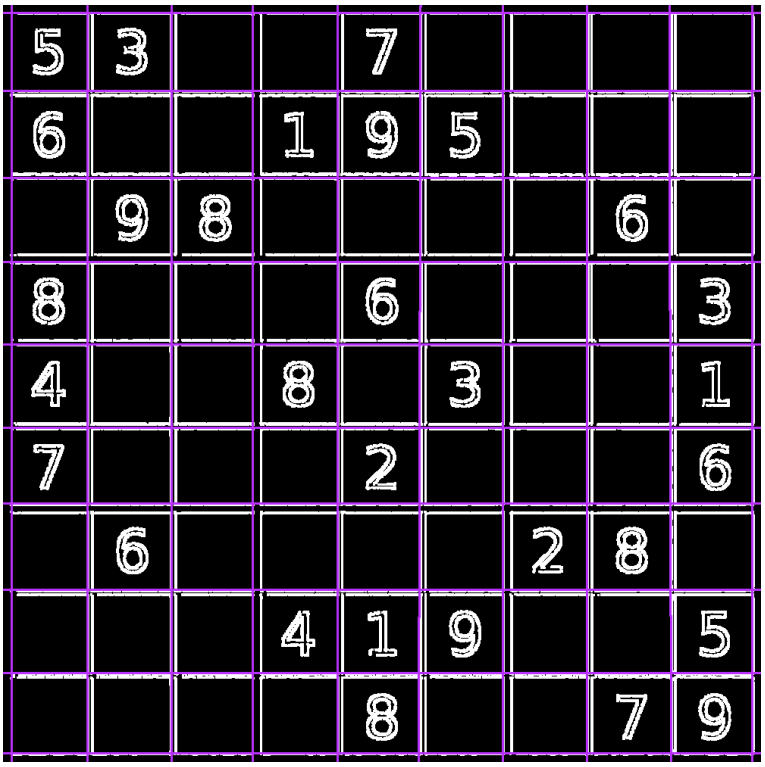


Figure 13 – Lignes simplifiées

4.4.4 Calcul des intersections et d'une matrice d'intersections

Une fois les lignes simplifiées, il reste uniquement 10 lignes horizontales et 10 lignes verticales. Il faut donc calculer les intersections entre ces lignes. Notre première idée était de calculer l'équation des droites afin de résoudre un système d'équation et de trouver l'intersection. Cependant, nous travaillons avec des lignes verticales qui ont donc une pente infinie et résoudre un système d'équation est coûteux. Nos lignes sont représentées

avec deux points, en dehors de l'image, de chaque côté de celle-ci. Nous pouvons donc calculer les intersections grâce à ces points.

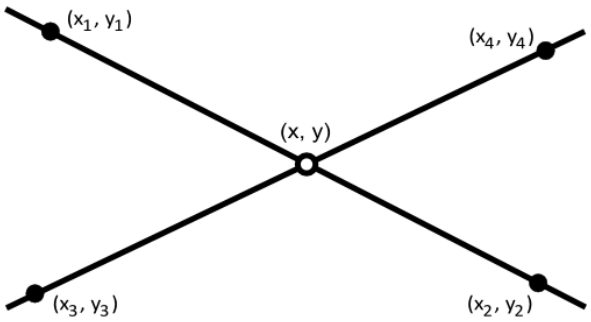


FIGURE 14 – Intersection de deux droites

$$x = \frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3x_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$
$$y = \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

Une fois les insertions calculées, nous les avons triées et mises sous la forme d'une matrice de points 10*10.

4.4.5 Position de la grille et des cases

Grâce à la matrice d'intersection, nous pouvons en déduire la position de la grille et celle des cases. Si on nomme M, la matrice d'intersection. Alors la grille est le quadrilatère formé par les intersections aux quatre coins de la matrice, c'est-à-dire les points $M_{1,1}, M_{1,10}, M_{10,10}, M_{10,1}$.

De même, nous pouvons en déduire les quadrilatères des cases grâce à cette matrice.

$$\forall (i,j) \in [\![1;10]\!]^2, case_{i,j} = \square M_{i,j}M_{i+1,j}M_{i+1,j+1}M_{i,j+1}$$

4.4.6 Correction de la perspective

Pour réaliser la correction de la perspective de l'image, nous avons mis en œuvre une correction de la perspective en utilisant une matrice homographique. Cette matrice, notée H, permet de transformer les coordonnées homogènes $(x, y, 1)$ d'un point de l'image d'origine vers une vue corrigée, compensant ainsi les distorsions liées à la perspective.

La matrice homographique H est définie comme suit :

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$

Les coefficients de cette matrice sont calculés en résolvant un système d'équations linéaires basé sur des correspondances de points entre l'image d'origine et l'image corrigée.

Pour corriger la perspective, nous avons besoin de calculer l'inverse de la matrice homographique, noté H^{-1}. La détermination de l'inverse implique des calculs minutieux basés sur les mineurs, le déterminant et les cofacteurs. Pour commencer, le déterminant de la matrice homographique $\det(H)$ est crucial pour la normalisation de l'inverse. Il se calcule comme le produit des éléments diagonaux moins le produit des termes croisés :

$$\det(H) = h_{11}(h_{22}h_{33} - h_{23}h_{32}) - h_{12}(h_{21}h_{33} - h_{23}h_{31}) + h_{13}(h_{21}h_{32} - h_{22}h_{31})$$

Ensuite, les cofacteurs de chaque élément de la matrice homographique sont utilisés pour former la matrice adjointe $\text{adj}(H)$. Les cofacteurs sont déterminés en alternant les signes des mineurs associés à chaque élément, où le mineur est obtenu en éliminant la ligne et la colonne de l'élément considéré. Par exemple, le cofacteur C_{11} associé à h_{11} est donné par :

$$C_{11} = (-1)^{1+1} \cdot M_{11}$$

où M_{11} est le mineur associé à h_{11}.

La matrice adjointe est alors formée en plaçant les cofacteurs dans une matrice transposée :

$$\text{adj}(H) = \begin{bmatrix} C_{11} & C_{21} & C_{31} \\ C_{12} & C_{22} & C_{32} \\ C_{13} & C_{23} & C_{33} \end{bmatrix}$$

Enfin, l'inverse de la matrice homographique est obtenu en divisant la matrice adjointe par le déterminant :

$$H^{-1} = \frac{1}{\det(H)} \cdot \text{adj}(H)$$

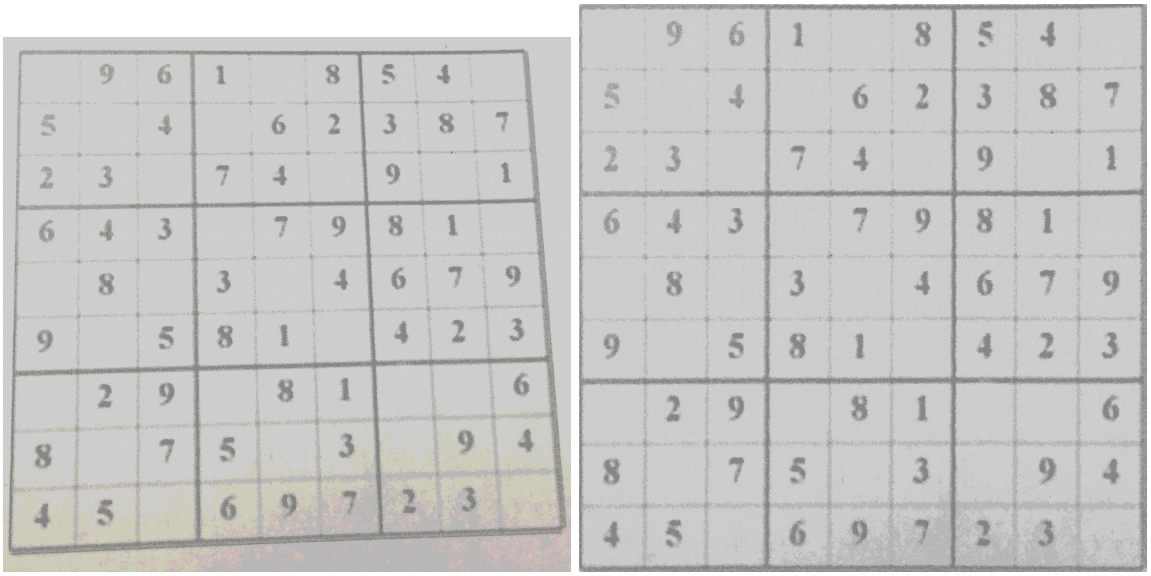


Figure 15 – Correction de la perspective

4.4.7 Traitement des cases

Après avoir obtenu la position des cases, on prend chaque cases de l'image après suppression des couleurs, on applique un filtre négatif et un seuillage par hystérésis avec un seuil haut obtenu grâce à la méthode d'Otsu. Cependant, il peut parfois rester des résidus de lignes apparents sur les cases, ce qui pourrait nuire à la bonne reconnaissance des chiffres par le réseau de neurones. Nous enlevons donc les résidus de lignes restantes en partant des bordures de la case (avec un rayon de 3 pixels car les lignes peuvent parfois ne pas être collées aux bordures de la case) et en supprimant récursivement les pixels blancs et ceux voisins. Notons l'importance du traiment préalable de la case car si des résidus de lignes sont connectés aux chiffres, ces derniers seront effacés et considérés comme des résidus nuisibles.

4.5 Réseau de neurones

4.5.1 Fonctionnement général de l'IA

Nous avons travaillé avec des structures matricielles pour pouvoir simplifier le plus possible les formules mathématiques et ainsi le risque d'erreur.

Les matrices ont été implémentées de cette manière de façon à stocker les dimensions des matrices plus facilement :

```
typedef struct
{
        size_t rows;
        size_t cols;
        double matrix[];
}Matrix;
```

Une IA fonctionne avec des couches constituées de neurones (comme le montrent les images ci-après).

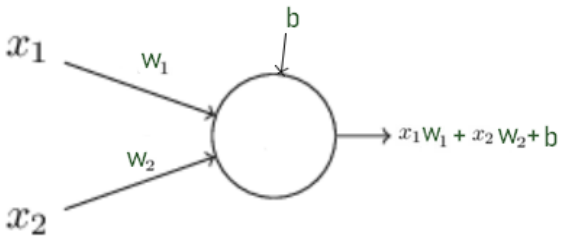


Figure 16 – Neurone IA

Les neurones possèdent des poids notés w_i et un biais noté b. Pour entraîner un modèle, il faut modifier la valeur de ses poids et biais de façon à obtenir les meilleurs résultats possibles.

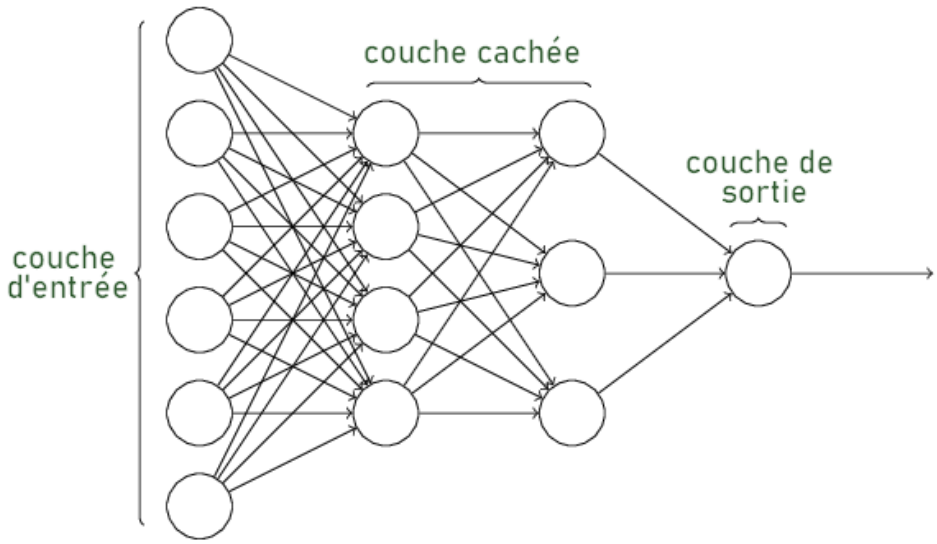


Figure 17 – Schématisation IA

Chaque couche de notre réseau (sauf la couche d'entrée) est constituée des matrices :

W (Matrices des poids des neurones de la couche)
↪ dimensions (nombre neurones couche actuelle, nombre neurones couche précédente)

B (Matrices des biais des neurones de la couche)
↪ dimensions (nombre neurones couche actuelle, 1)

A (Matrices des valeurs de sortie des neurones de la couche)
↪ dimensions (nombre neurones couche actuelle, nombre de données)

La couche d'entrée possède seulement une matrice A pour les données à traiter.

4.5.1.1 Mécanisme de notre réseau de neurones

Notre réseau de neurones est un réseau pouvant être multicouche. Par exemple pour le modèle du XOR, un réseau avec une couche d'entrée de deux neurones, une couche cachée de trois neurones et une couche de sortie d'un neurone est totalement adapté et suffisant. Alors que pour un modèle plus conséquent comme celui des images de chiffres en 28 par 28 pixels, avoir deux couches cachées peut s'avérer utile.

Enfin chaque couche possède une fonction d'activation qui permet d'uniformiser les sorties. Dans notre cas, nous utilisons des fonctions sigmoïde et éventuellement softmax pour la couche de sortie dans le cas d'une classification de plus de deux éléments (c'est le cas pour les images de chiffres où il peut y avoir neuf cas pour les chiffres de 1 à 9).

Pour pouvoir utiliser un réseau de neurones et faire des prédictions, il faut l'entraîner. Pour ce faire, on fixe un nombre d'époques qui correspond au nombre d'itérations des processus de propagations avant/arrière (définis ci-après). Une fois les époques terminées, il faut tester notre modèle sur des données et calculer sa précision (qui correspond au nombre de bonnes prédictions divisé par le nombre de prédictions totales).

Voici le principe mathématique simplifié des deux étapes majeures de l'entraînement de notre réseau de neurones et de la fonction coût choisie.

4.5.1.2 Propagation avant

Lors de cette phase, le réseau de neurones calcule les sorties de chaque couche les unes après les autres.
On peut utiliser cette formule appliquée de la couche 2 jusqu'à la dernière couche :
$[C]$: couche actuelle, $[C-1]$ couche précédente.

$$A[C] = \sigma(W[C] \cdot A[C-1])$$

σ étant une fonction d'activation (sigmoïde ou softmax)

4.5.1.3 Propagation arrière

Lors de cette phase, on cherche à mettre à jour les poids et biais de notre modèle pour obtenir de meilleures prédictions. Pour ce faire il faut préalablement choisir une fonction coût notée L qui calcule l'erreur de notre modèle. Nous avons choisi la fonction Log Loss pour faire cela plutôt que la fonction d'erreur quadratique qui est moins performante.

$$L = -\frac{1}{m}\sum(Y \cdot \log(A[f]) + (1-Y) \cdot \log(1-A[f]))$$

avec Y la sortie attendue et $A[f]$ la sortie d'activation de la dernière couche du réseau.

Une fois cette fonction définie, on calcule une matrice δ pour chaque couche (en partant de la dernière jusqu'à la première) qui servira au calcul des dérivées partielles des poids et des biais de la couche.

On définit la matrice δ de la couche de sortie par

$$\delta[C] = A[f] - Y$$

avec Y la sortie attendue et $A[f]$ la sortie de la dernière couche de notre réseau de neurones

Ensuite, on peut définir récursivement jusqu'à la première couche avec L la fonction coût

$$\begin{cases} \frac{\partial L}{\partial W[C]} &= A[C-1] \cdot \delta[C] \\ \frac{\partial L}{\partial B[C]} &= sommeParLigne(\delta[C]) \\ \delta[C-1] &= W[C]^t \cdot \delta[C] \odot \sigma'(A[C-1]) \end{cases}$$

$\odot$ est le produit d'Hadamard et $sommeParLigne$ une fonction qui calcule la somme de chaque ligne de la matrice et retourne la matrice résultante.

Enfin, il suffit de mettre à jour les paramètres de l'IA avec ces formules à chaque itération d'entraînement de notre réseau et pour chaque lot de données

$$W[C] = W[C] - \mu \times \frac{\partial L}{\partial W[C]}$$

$$B[C] = B[C] - \mu \times \frac{\partial L}{\partial B[C]}$$

μ étant la vitesse d'apprentissage qui est une valeur définie par l'utilisateur et qui dépend du modèle.

Il est aussi intéressant de remarquer que l'utilisation de matrices permet très facilement d'entraîner plusieurs données d'un seul coup. En effet, les dimensions des matrices W et B qui caractérise notre réseau de neurones ne dépendent que du nombre de neurones et non du nombre de données. Ainsi en choisissant m pour le nombre de colonnes de la matrice A, les formules restent toujours vraies et il sera possible d'entraîner m données simultanément. Ce m est appelé "batch size" ou "taille du lot" en français. Plus il est élevé plus le modèle va réussir à généraliser en prenant une moyenne des m données, mais à l'inverse s'il est trop grand le modèle ne va plus réussir à voir les spécificités et ne sera plus précis. Nous avons aussi mis en place un mélange aléatoire des données entre chaque époque ce qui permet de ne pas prendre toujours les mêmes lots et par conséquent d'augmenter la capacité de généralisation de notre modèle.

4.5.2 Mise en forme des bases de données

Pour charger en mémoire, il est essentiel d'allouer en mémoire toutes les données. Mais pour cela, il faut connaître la taille/le nombre de données à allouer. C'est le rôle des entêtes que nous avons développées qui se trouvent juste au-dessus des données à traiter (en orange dans l'image ci-dessous).

Par exemple, le "nb_data" renseigne le nombre de données d'entraînement contenu dans ce fichier, "nb_input_param" contient le nombre de données pour la première couche du réseau (c'est aussi le nombre de neurones de la première couche), respectivement avec "nb_output_param" qui est aussi le nombre de neurones de la couche de sortie.

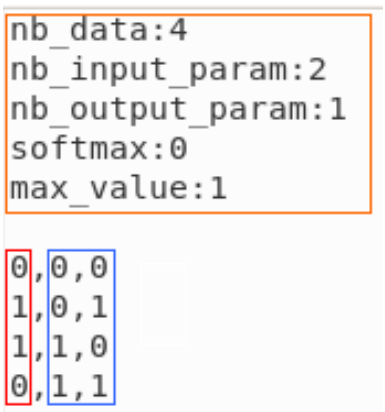


FIGURE 18 – Exemple de la base de données du XOR

Une fois que les en-têtes ont été pris en compte, on peut stocker les données d'entrée (en bleue) et de sortie (en rouge) du modèle. Et on reconnaît dans la première ligne que $0 \oplus 0 = 0$ et qu'à la ligne suivante $0 \oplus 1 = 1$ et ainsi de suite.

La même logique a été adoptée avec la base de données d'images, mais sur 28*28 données d'entrée (une entrée pour chaque pixel d'une image de 28 par 28).

De plus pour optimiser la mémoire, au lieu de stocker ces données de manière contiguë dans une liste de liste d'entiers ou de flottants, nous avons préféré les stocker sous la forme de listes de pointeurs vers les données de façons individuelles.

4.5.3 Utilisation de notre réseau de neurones

Notre réseau de neurones peut s'utiliser très simplement grâce à des arguments dans la phase d'exécution. Par exemple "-m" choisi le modèle voulu (xor ou images) "-l" change le pas d'apprentissage de l'IA, "-e" change le nombre d'itérations lors de l'entraînement ou encore "-train"/"-test" qui fixe le nombre de données pour entraîner et tester notre modèle. Enfin le "-b" sert à changer le nombre de données à passer en simultané (le batch size). Il existe d'autres arguments pour charger un modèle déjà entraîné, changer les bases de données...

```
./network -m images -train 2000 -test 3000 -l 0.5 -b 32 -e 3
```

FIGURE 19 – Changement des paramètres de l'IA

Une fois notre IA lancée, toutes les étapes sont clairement affichées pour informer en temps réel d'utilisateur de l'avancée. Dans le cas ci-dessous, la première étape est de charger les arguments donnés dans la console (ceux non renseignés sont pris par défaut selon le modèle)

L'étape d'initialisation charge en mémoire l'ensemble de la base de données d'entraînement (sous forme de liste de pointeurs vers chaque donnée pour éviter un trop gros stockage contigu en mémoire).

L'étape d'entraînement va appliquer les processus de propagation avant et arrière en boucle pour ajuster le plus possible et de la meilleure des manières les poids et les biais de chaque neurone.

Une fois entraîné, il faut tester la précision de notre modèle, c'est le rôle de la fin de notre programme. Après avoir libéré en mémoire les données d'entraînement, il charge les données de test et fait une prédiction avec les poids et biais trouvés avec l'entraînement. La précision de notre modèle est calculée finalement en faisant le nombre de bonnes prédictions divisé par le nombre de données de tests.

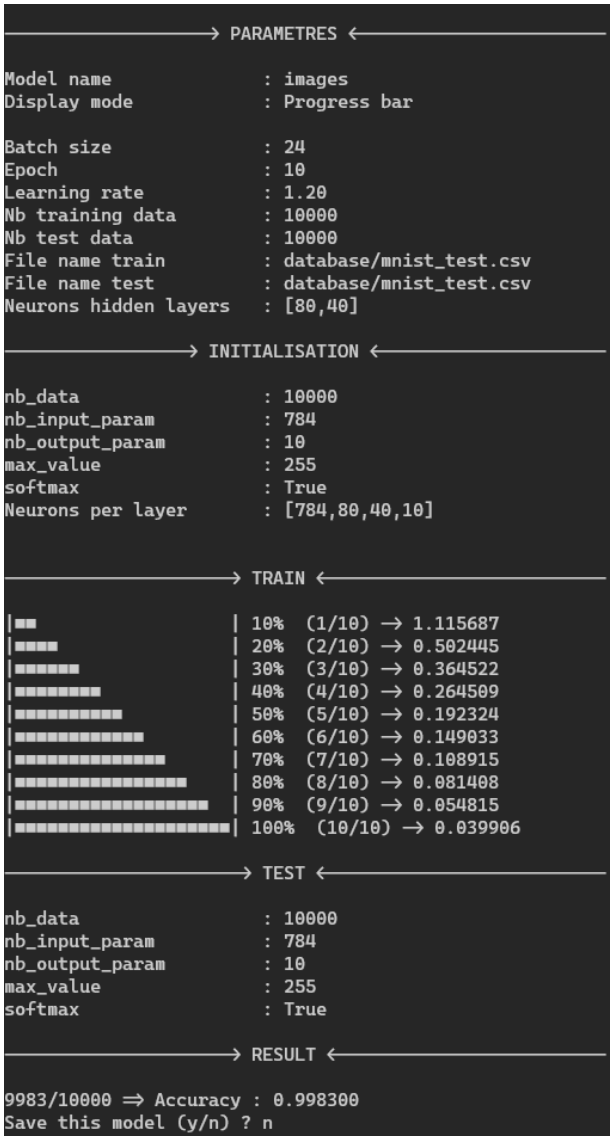


FIGURE 20 – Aperçu de l'IA dans la console

4.5.4 Système de sauvegarde et de chargement de modèle

Une chose importante pour une intelligence artificielle est de pouvoir charger un modèle déjà pré-entraîné. Cela permet de ne pas devoir le ré-entraîner à chaque fois que l'on veut prédire par exemple un chiffre.

Nous avons ainsi mis en place un système de sauvegarde qui, après entraînement, stocke tous les poids/biais de chaque couche de notre modèle dans un fichier. Et réciproquement, un système pour charger ce fichier en remplissant les matrices W et B de chaque couche.

À la fin de chaque entraînement, il est demandé si nous voulons sauvegarder ou non ce modèle. Si nous voulons sauvegarder, un nom du fichier de sauvegarde sera demandé puis un fichier sera créé. Il est aussi possible de charger ce modèle avec l'argument '-load' suivi du nom renseigné lors de la sauvegarde.

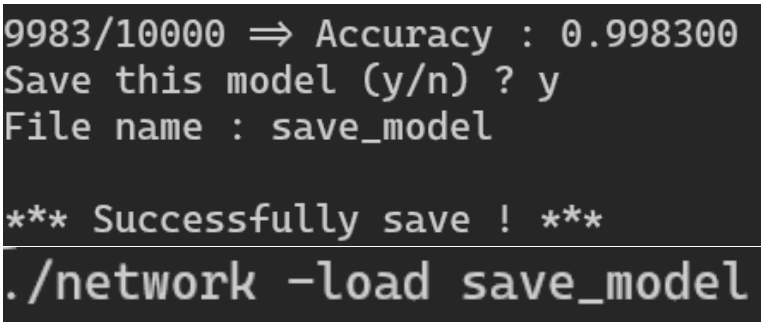


FIGURE 21 – Exemple de procédure pour sauvegarder (en haut) et charger (en bas) un modèle

Cette fonctionnalité peut s'avérer très pratique pour sauvegarder un modèle de reconnaissance de chiffre sans devoir le ré-entraîner de zéro pour faire une prédiction.

4.5.5 XOR

Notre réseau de neurones est capable de très bien prédire un XOR avec une précision de 100% après un entraînement très rapide. Une prédiction est considérée comme un 1 si sa valeur est > 0.5, 0 sinon

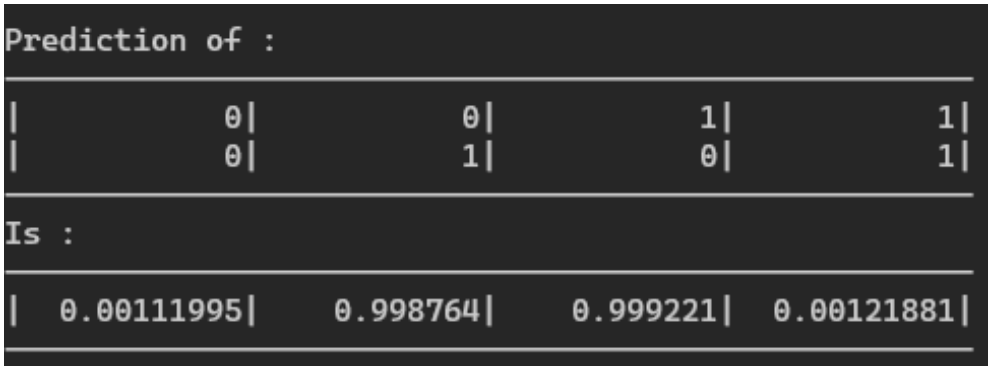


FIGURE 22 – Prédiction de XOR après un entraînement

Le graphique ci-dessous montre l'évolution de la fonction coût en fonction du nombre d'époques.À partir de 2000 époques l'erreur de notre modèle n'évolue quasiment plus et nous pouvons remarquer l'apparition d'une asymptote horizontale, signe que le modèle est déjà assez entraîné et ne pourra plus beaucoup s'améliorer avec ces paramètres d'entraînement (vitesse d'apprentissage, configuration des couches, taille des lots, ...).

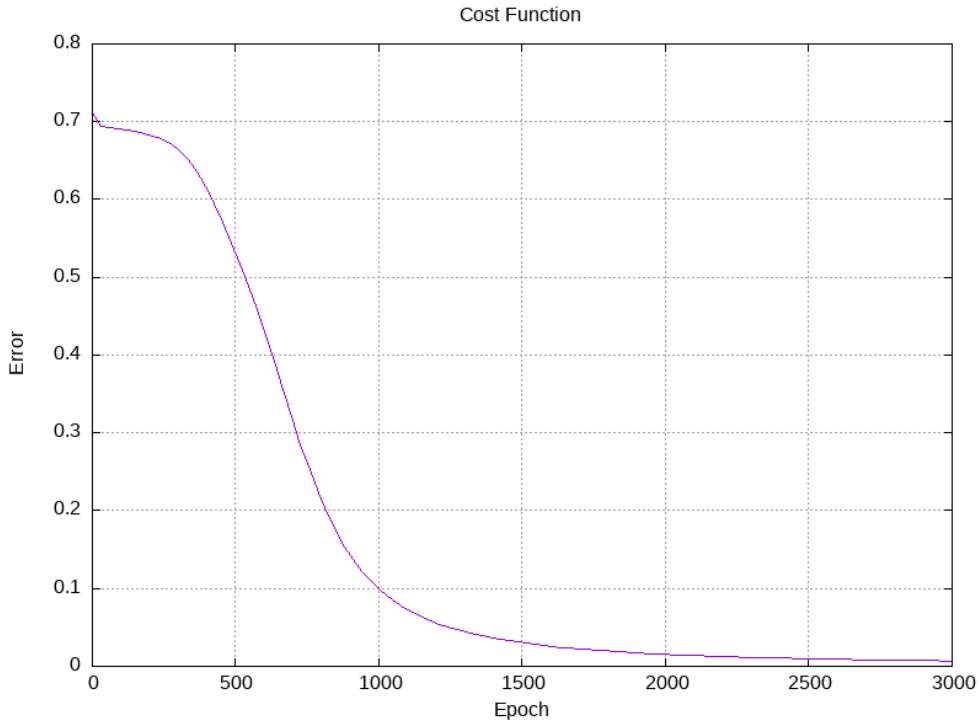


FIGURE 23 – Évolution de la fonction coût en fonction des époques

4.5.6 Images

Enfin notre réseau de neurones fonctionne aussi sur des images de chiffres. Nous avons par exemple essayé sur les bases de données mnist/emnist qui contiennent plusieurs centaines de milliers d'exemples d'images de chiffres. Avec ces derniers, il a été possible d'atteindre des précisions de 97/98% sur des bases de données de test différentes de celles d'entraînement (que le modèle n'avait encore jamais vu). Tandis que sur la même base de données pour tester, il est possible de frôler les 100% de précision.

Par la suite, nous avons créé notre propre base de données à l'aide d'un script python qui a partir de police d'écriture nous donne des images de 1 à 9. Nous avons téléchargé environ 2000 polices en prenant soin d'enlever les polices fantaisistes qui serait un frein à l'apprentissage de notre IA

Pour augmenter la généralisation de l'IA, nous avons choisi d'effectuer les mêmes traitements que sur les cases sur notre base de donnée en commençant par exemple par la binarisation des images pour avoir seulement du blanc ou du noir et non des nuances de gris. Ensuite, face au bruit que pouvait avoir certaines images ou aux déformations causées par le filtre canny, nous avons essayé de récréer un semblant d'aléatoire. Nous avons décidé d'une probabilité d'apparition de pixels blanc/noir dans l'image à des positions aléatoires. Le but étant que l'ia comprenne que l'important dans l'image est le chiffre et non tous les résidus qui peuvent être autour.

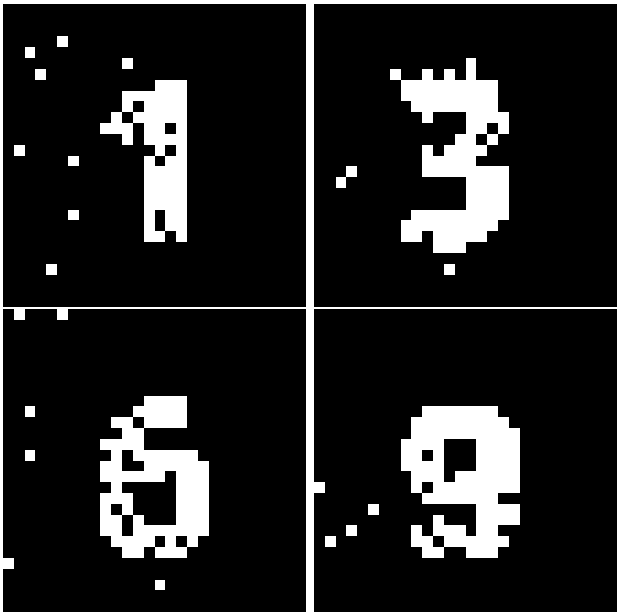


FIGURE 24 – Exemple de chiffres dans notre base de données

En plus de ces déformations, nous avons la possibilité de répété chaque chiffre plusieurs fois, mais à des positions différentes, car les chiffres ne seront pas toujours exactement au milieu. Et comme notre IA mélange les données lors de l'entrainement, il n'y a pas de risque qu'elles se retrouvent toujours dans le même lot, ce qui pourrait fausser les résultats.

Avec toutes ces images, nous avons pu créer des bases de données de 20 000, 40 000, voire 200 000 images très simplement.

Mais nous avons oublié un point majeur, les base de données comme mnist proposé directement leur donnée sous forme de csv ou en format binaire directement. Or ici, nous disposons seulement d'image au format png. Il a donc fallu créer différente fonction pour convertir des images png en matrice pour ensuite pouvoir écrire dans un seul même fichier toutes les données dans le même format que décrit précédemment avec la base de données du XOR.

4.5.7 Intégration dans le processus de résolution

Maintenant que nous avons une IA fonctionnelle, il faut qu'elle puisse faire des prédictions en adéquation avec ce que l'IA reçoit (les cases) et avec ce que le solveur a besoin, une liste de chiffres et -1 en cas de vide.

Pour ce faire, au lieu de devoir faire un appel différent pour chaque case, nous avons créé une unique fonction qui prend une liste de *SDL_Surface** qui les convertis en matrice puis effectue la prédiction sur chacune d'entre elles et retourne une liste de chiffres.

Cette implémentation permet très facilement d'intégrer l'IA dans le processus de résolution

4.6 Algorithme de résolution du sudoku - Solveur

L'algorithme de résolution du sudoku, solveur, que nous avons implémenté peut gérer des sudokus dits normaux (en 9x9) ainsi que des hexadokus (en 16x16).

Nous avons créé une fonction qui permet de charger une grille depuis un fichier et une autre qui sauvegarde une grille dans un fichier pour y inscrire le sudoku résolu. La fonction de chargement permet aussi de vérifier si le format de la grille donné n'est pas bon ou si le fichier est inexistant, auxquels cas, le programme renvoie une erreur.
La grille donnée ressemble à :

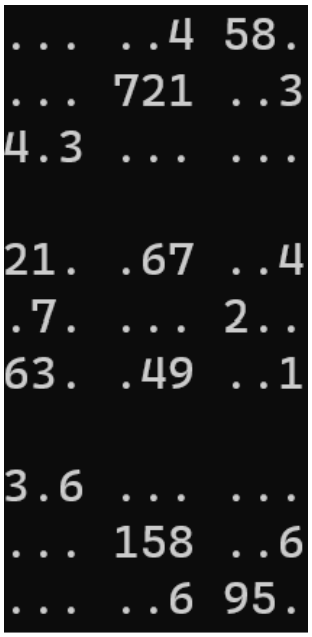


Figure 25 – Sudoku non résolu

Nous avons ensuite créé une fonction qui vérifie si une valeur peut être mise dans une cellule. Or notre solveur doit pouvoir résoudre des sudokus et hexadoku qui n'ont pas les mêmes contraintes. Les chiffres du sudoku vont de 1 à 9 alors que l'hexadoku de 0 à 9 puis de A à F. Il a fallu faire des cas pour bien discerner ces cas de figures.

Après avoir créé les principales fonctions annexes qui pouvaient nous servir, nous sommes passés à l'une des fonctions les plus importantes de notre programme, la conversion en matrice où chaque case contient l'ensemble des possibilités de chiffre de la case associé. Ainsi, elle récupère chacune des cases vides de notre sudoku et crée une liste contenant les possibilités des chiffres en fonction des règles du sudoku (ligne, colonne, sous-carré).

Après avoir créé cette matrice de listes, nous l'avons ordonnée de manière croissante, de la case ayant le moins de possibilités à celle en ayant le plus de possibilités de chiffres.

Nous avons par ailleurs, fait en sorte de gérer les principaux cas d'erreurs qui peuvent survenir lors de la résolution d'un sudoku. Nous avons donc créé une fonction qui regarde si un élément est présent 2 fois dans une ligne, colonne ou sous-carré, auquel cas le sudoku est invalide. Le problème principal lorsque l'on ne traite pas cette erreur est que notre

solveur résout quand même le sudoku, mais il est totalement faux, car dès le départ, il n'est pas solvable.

Nous avons désormais tous les éléments nécessaires pour appliquer la fonction récursive principale du solveur, une méthode qui est appelée "backtracking". Cette fonction remplie d'abord les cellules du sudoku qui ont le moins de possibilités au plus de possibilité (grâce à la liste que l'on a triée auparavant). Lorsque le programme arrive sur une case qu'il ne peut pas remplir dû au pré-remplissage des cellules précédentes, il enlève les valeurs ajoutées aux cellules précédentes jusqu'à là où il est nécessaire d'aller pour ne plus avoir de problème sur la case qui nous posait soucis, puis il remplit encore une fois. Il s'arrête lorsqu'il a fini de remplir tout le sudoku ou qu'aucune possibilité n'existe.

La dernière étape consiste à afficher le résultat dans un nouveau fichier. Pour ce faire, nous créons un nouveau fichier avec pour extension ".result". Notre solveur est désormais fonctionnel. Voici ce que notre sudoku (figure 21) est devenu :

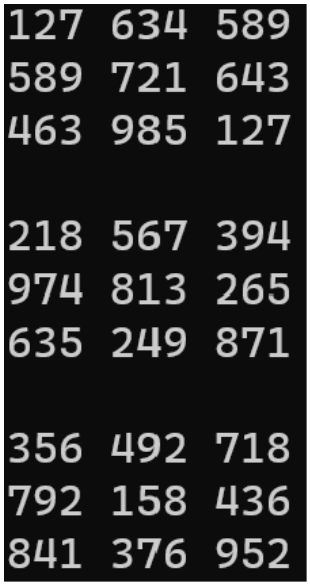


Figure 26 – Sudoku résolu

Nous avons fourni dans le repo différentes grilles de sudoku à remplir. Il y a plusieurs sudokus valide, un sudoku non valide, et un hexadoku valide.

Figure 27 – Les différentes grilles de test

4.7 Interface graphique

Pour produire l'interface graphique de notre OCR - Sudoku solveur, nous avons utilisé glade. Pour préciser, Glade est un outil de conception d'interface graphique (GUI) utilisé principalement dans le développement de logiciels pour les environnements de bureau Linux. Il offre un environnement graphique convivial qui permet aux développeurs de créer des interfaces utilisateur visuelles pour leurs applications sans avoir à écrire manuellement le code source.

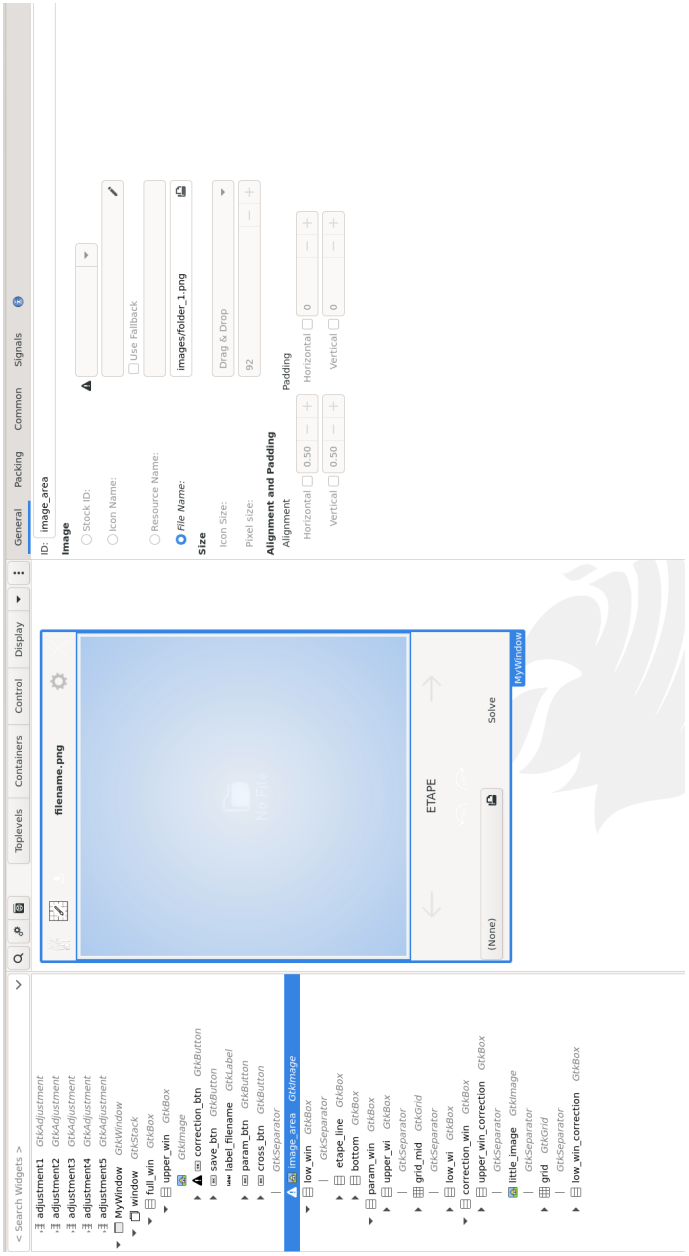


FIGURE 28 – Aperçu du logiciel Glade

L'interface graphique de notre OCR à été conçue de telle manière que nous pouvons voir les différentes étapes du processus mais aussi changer des paramètres spécifiques. Pour cela nous avons choisi de répartir les informations sur trois pages différents pour que l'utilisateur est une interface épurée et claire. Pour améliorer l'expérience utilisateur nous avons aussi modifié les couleurs présentes de base dans glade pour mettre un thème sombre. Ce thème sombre se caractérise par un gris foncé pour le haut et bas de la page, ceux qui contiennent les boutons prinicipaux. Pour permettre une différence entre chaque partie, la partie centrale de l'écran est d'un gris plus clair.

Ci-dessous sont présentées les interfaces utilisateurs en fonction des différents pages.

4.7.1 Page Principale

Lorsque nous lançons notre application, nous avons accès à une page principale. Cette page constitue l'essentiel de notre interface graphique car c'est ici que le traitement de l'image sera effectué. Nous disposons de plusieurs boutons que nous allons détailler plus précisément ci-dessous. Au milieu de l'écran nous avons un champ libre avec marqué "No File" cet endroit est fait pour voir l'image de sudoku qui sera traitée.

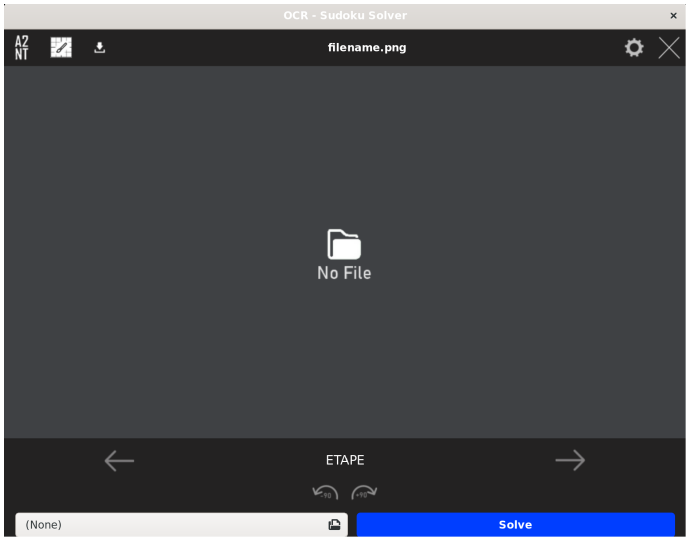


Figure 29 – Page principale vide

Pour importer cette image, nous disposons en bas à gauche de la fenetre d'un bouton qui lorsqu'on clic dessus nous amène dans un explorateur de fichier et nous permet de choisir une image. Cette image ne peux avoir comme extension que ".png", ".jpg" et ".jpeg". Ce choix est fait pour éviter d'introduire des fichiers qui ne sont pas des images dans notre interface.

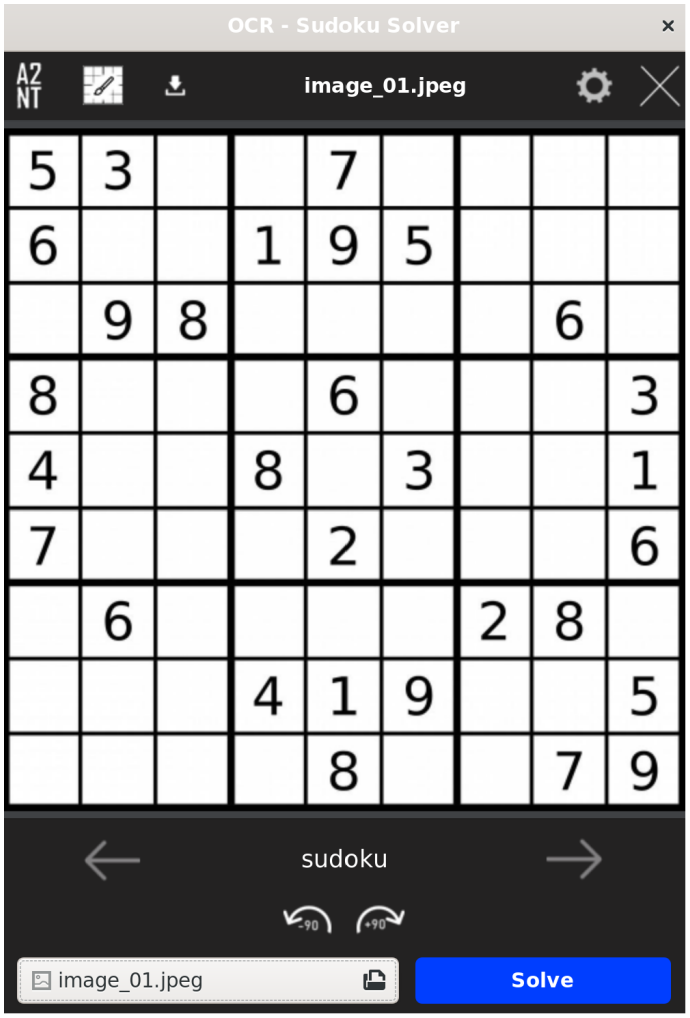


FIGURE 30 – Page principale avec image

Lorsque l'image est choisie, elle s'affiche à la place du "No File" vu précedemment. Il est désormais possible de résoudre notre sudoku. Pour cela, le bouton "Solve" est présent en bas à droite de l'interface. Il est en bleu foncé et ressort donc de notre charte graphique principale car c'est le bouton le plus important de l'interface graphique. Il change aussi de couleur lorsque nous passons la souris dessus, pour avoir une teinte de bleu plus claire.

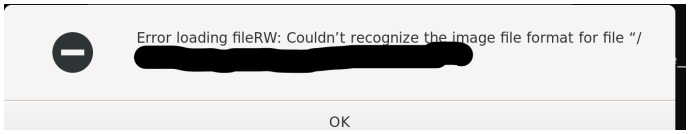


FIGURE 31 – Erreur d'extension du fichier

Appuyer sur ce bouton permet de résoudre le sudoku présent mais il existe tout de meme plusieurs cas possibles lors de la résolution. Le sudoku peut tout d'abord etre résolu ainsi nous verrons les images s'afficher au fur et à mesure qu'elles seront réussies. Il y a dans l'ordre : le sudoku de base, le grayscale, le contraste, le bilatéral, le sobel, hough trandform, la rotation automatique et la perspective corrigée puis le sudoku résolu.

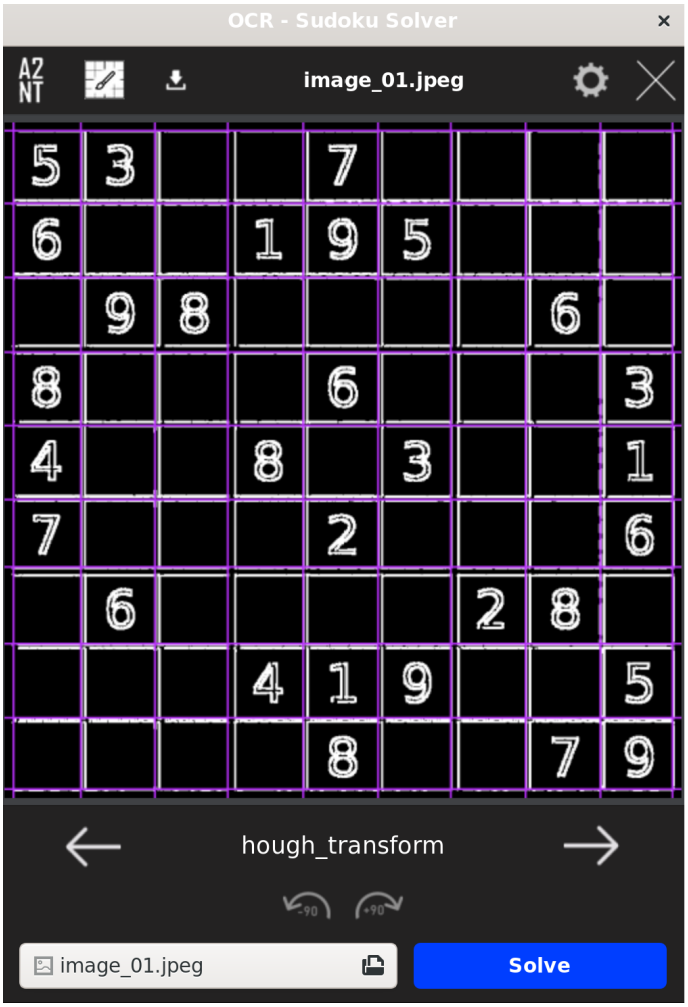


FIGURE 32 – Exemple d'affichage durant le traitement (Etape - Hough)

Le sudoku peux aussi rencontrer différents problèmes tel que la non détection des cases qui renvoi un "problème dans la détection des lignes" ou encore un "impossible de résoudre le sudoku" si l'IA s'est trompée lors de la reconnaissance ne serait-ce que d'un chiffre.

Pour connaitre le fichier pris en charge, il y a tout en haut le nom du fichier mis en gras. Pour ce qui est de l'étape actuelle, elle est notée en fonction de l'image affichée en bas de l'image. Pour pouvoir revisualiser les différentes images, nous avons disposé autour de l'étape actuelle deux boutons de flèches qui permettent de revenir en avant puis en arrière. Ces boutons ne sont activés que lorsque le bouton "solve" est appuyé avec une image chargée.

Nous avons aussi disposé deux boutons de rotation de 90 degrés de l'image qui ne sont aps activables lorsqu'il n'existe pas de fichier inséré dans l'interface, ces boutons sont aussi désactivés dès lors que le bouton solve est appuyé, cela permet d'éviter de casser le traitement en cours.

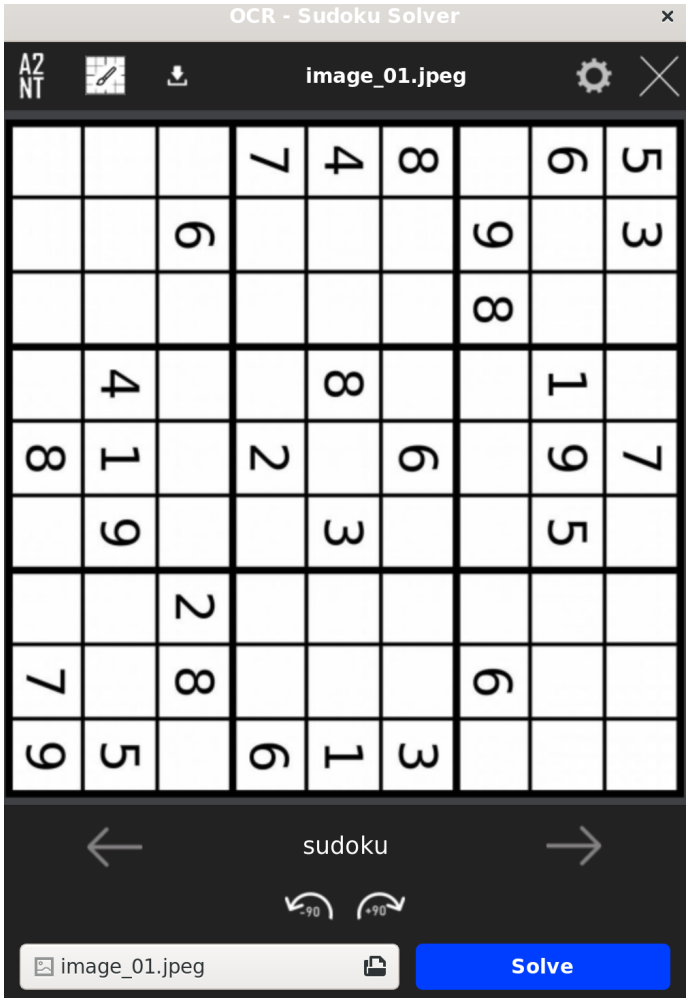


FIGURE 33 – Rotation de la grille

Pour le reste des informations de cette page, nous avons notre logo qui est présent tout en haut à gauche de la page, en haut à droite se trouve un bouton croix qui permet de quitter le programme avec un appui sur bouton.

Les derniers boutons sont : L'engrenage qui amène vers la page de paramètres (voir sous-section ci-dessous, "Page de paramètres"), le pinceau amène vers la page des corrections (voir sous-section ci-dessous, "Page de corrections") et enfin, il y a un bouton pour sauvegarder l'image actuelle sur l'interface.

Ce bouton ne fonctionne que si une image est présente, on peu ensuite choisir le nom du fichier ainsi que l'emplacement où l'on veut le sauvegarder et il ne reste plus qu'à appuyer sur "ok". Il est aussi possible de "cancel" l'action et de ne pas sauvegarder l'image.

FIGURE 34 – Sauvegarder l'image

4.7.2 Page de Paramètres

La page de paramètres comporte des boutons qui permettent de changer le traitement de l'image. Tous les boutons sont maximisés à une valeur de 1000. Les kernels ne sont que des entiers, les 3 autres peuvent avoir 2 chiffres après la virgule. Nous pouvons ainsi changer pour la hauteur et largeur du kernel. Nous pouvons aussi modifier le treshold de Hough, le bilatéral brightness ainsi que le spatial. Nous pouvons toujours modifier les valeurs avec une insertion numérique ou en appuyant sur des boutons "+" ou "-". Les boutons sont initialisés avec des valeurs par défaut.

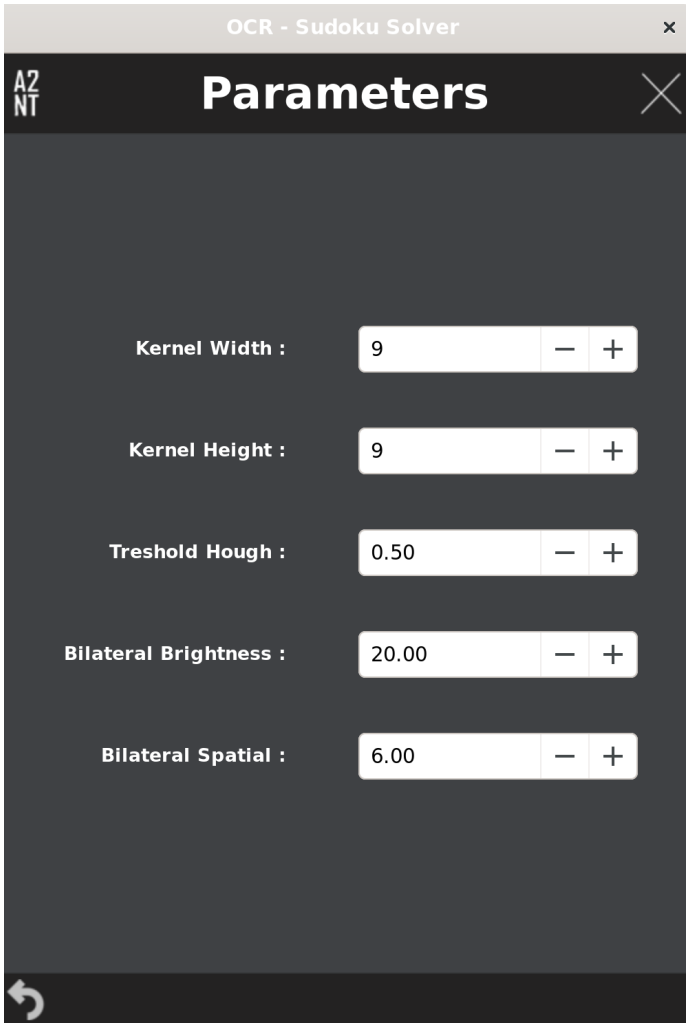


FIGURE 35 – Page de paramètres

Le logo ainsi que le bouton pour quitter l'application sont toujours présentes. Tout en bas de l'interface, il y a un bouton retour qui permet de revenir à la page principale évoquée auparavant.

FIGURE 36 – Changement des valeurs des paramètres

4.7.3 Page de Corrections

La page de correction comporte aussi le même en-tête que les autres pages avec le logo et la croix pour quitter la page. Le centre de l'interface est un sudoku qui est composé de 81 boutons. Il y a aussi en plus petit au-dessus l'image qui est affichée dans la fenêtre principale. Les cases sont remplies après avoir résolu l'image. Cette page permet de changer manuellement si l'image n'a pas pu être résolue, car l'IA aura mal reconnu les nombres. Il s'agit alors de remettre les bons chiffres aux bons endroits.

Tout en bas de la page, il y a le bouton retour qui ramène vers la page principale ainsi qu'un bouton solve. Le solve permet de résoudre la grille qui n'avait jusqu'alors pas été résolue.

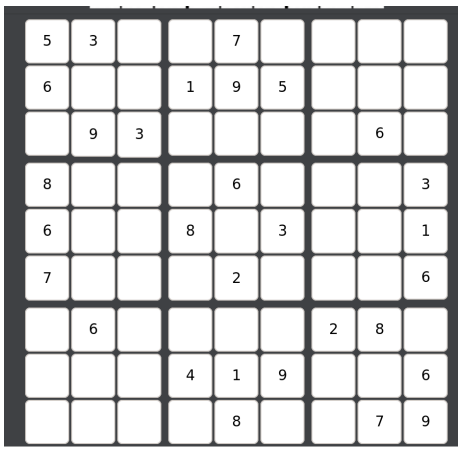


FIGURE 37 – Avant d'avoir appuyé sur solve

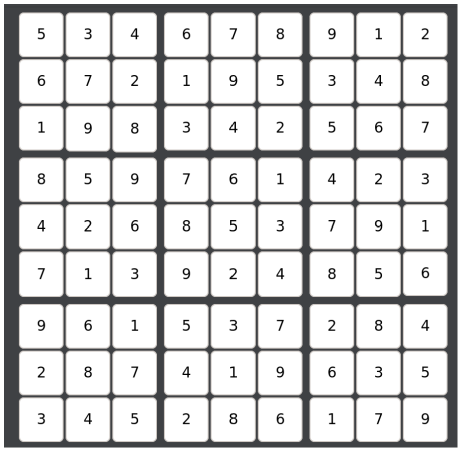


FIGURE 38 – Après avoir appuyé sur solve

4.7.4 Multithreading

Nous avons séparés dans deux threads, donc dans deux fils d'exécutions différents la partie interface graphique de la partie calcul. Pour ce faire, grâce à des écritures sur des variables en faisant attention à la synchronisation, nous envoyons à chaque étapes réalisés toutes les informations à l'interface graphique afin que celle-ci puisse afficher le résultat de chaque étape à l'utilisateur.

4.7.5 Affichage de la solution

Si tout le processus a fonctionné correctement, notre OCR renvoie une image semblable à l'image d'origine (avec rotation et correction de perspectives si besoin) comme montré ci-dessous.

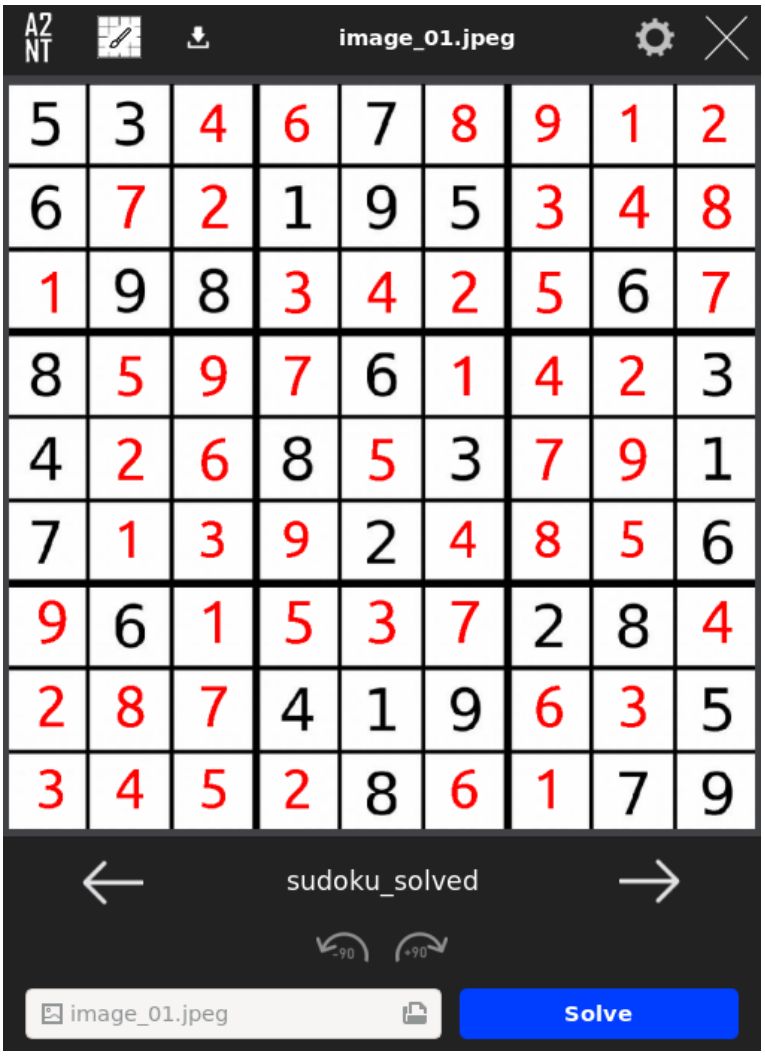


FIGURE 39 – Affichage de la solution pour la grille n°1

Pour réaliser ces chiffres, nous avons opté pour l'utilisation de `SDL_ttf` qui est une bibliothèque d'extension qui permet à partir d'une police d'écriture d'écrire sur une image. La police doit être en format TTF (TrueType)

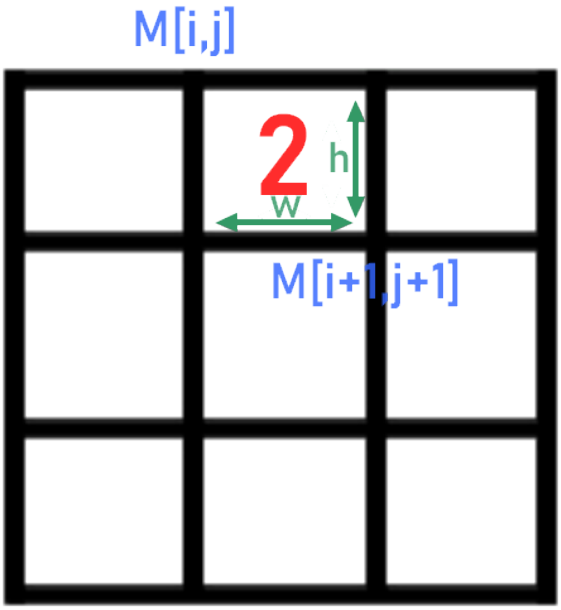


FIGURE 40 – Calcul de la position des chiffres

Le calcul de la position est assez simple et peut s'adapter à n'importe quel type de case. Le calcul de la position (x, y) du chiffre à afficher dans la case (i, j) est le suivant.

$$w = M[i+1, j+1].X - M[i, j].X \tag{1}$$
$$h = M[i+1, j+1].Y - M[i, j].Y \tag{2}$$
$$x = M[i, j].X + 0.35 \times w \tag{3}$$
$$y = M[i, j].Y + 0.1 \times h \tag{4}$$
$$taillePolice = 0.7 \times h \tag{5}$$
$$\tag{6}$$

Le problème est que `SDL_ttf` ne peut pas directement afficher un chiffre sur une image. Nous avons donc dû créer une fonction qui fusionne deux `SDL_surface` une étant le fond donc l'image de sudoku et l'autre le chiffre. Nous parcourons chaque pixel du chiffre et si la couche alpha est > 0 alors, nous affichons le chiffre dans la couleur souhaité (ici en rouge : #FF0000FF)

5 Conclusion

Notre projet OCR a avancé comme nous le souhaitions. Nous pensions que la principale difficulté du projet serait l'IA, mais ce qui nous a demandé le plus d'efforts et de réflexion a été le traitement d'image, car les étapes sont très différentes les unes des autres.

Pour autant, l'avancement a été optimal dans de nombreux domaines, notre solveur répond aux exigences de la soutenance finale ainsi qu'au bonus. Il en va de même pour notre intelligence artificielle qui a pour autant eu quelques adaptations nécessaires pour savoir quelle était la meilleure manière de lui faire apprendre. Nous avons en fin de compte opté pour un entrainement précis et moins généraliste mais nous avons toujours accès au second entrainement plus géénral.. Concernant le traitement de l'image, la totalité des prérequis pour cette soutenance ont été atteints et nous avons pu produire tout ce que nous voulions. Pour rassembler tout cela, nous avons créé une interface graphique qui englobe toutes les fonctionnalités que nous avons implémentées.

Ce projet d'OCR de sudoku aura été très compliqué, car la charge de travail était très importante. Cependant, ce projet nous aura permis de consolider notre travail en équipe et d'encore plus communiquer nos avancées que lors du projet du S2, car nous étions dépendants les uns des autres.