Projet OCR

Rapport de Projet

Grégoire Le Bihan, Dardan Bytyqi
Léo Menaldo, Nathan Fontaine

Table des matières

1 Prélude

Aujourd'hui, en ouvrant un journal ou un magazine, on trouve toujours une ou deux pages remplies de casses-tête que ce soit des mots croisés, des mot fléchés... Mais on y trouve aussi des grilles de sudoku, un jeu très simple par abord, mais qui est pourtant extrêmement populaire, en effet, pas besoin d'avoir la bosse des maths pour terminer une grille à 100%, il suffit simplement d'avoir de la logique.
Dans les années 2000, les sudokus sont apparus dans les journaux, et ont amené le jeu sous un nouveau jour. Depuis, des informaticiens se sont penchés sur le sujet et ont essayé de résoudre des sudokus à l'aide d'un programme informatique, ils ont donc créé plusieurs algorithmes, certains se rapprochant de la façon dont un joueur résout la grille, d'autres étant un peu plus abstraits et très calculatoires.
Dans le cadre du projet de troisième semestre à l'EPITA, on nous a confié la mission de devoir être capable de résoudre une grille de sudoku présente sur une image, quelle qu'elle soit, pouvant être prise de n'importe où, n'importe quand.

2 État des lieux

Lors de la précédente soutenance, nous avons présenté l'avancée de notre projet, c'est à dire :

1. **Floutage de l'image**
2. **Binarisation de l'image**
3. **Découpage de l'image**
4. **Un réseau neuronal capable d'apprendre la fonction XOR**
5. **Un solveur de sudoku complet**

En plus d'avoir présenté et réalisé ces points, nous avions remarqué que nous étions en retard sur ce qui était à faire suite à une répartition des tâches trop bâclée. C'est pourquoi nous avons revu cette assignation des tâches et avons mieux réparti le travail nécessaire.

3 Assignation des tâches

Ci-dessous, les tâches assignées à chacun des membres du groupe :

3.1 Dardan Bytyqi

Pour ma part, j'ai travaillé sur tout ce qui concerne le pré-traitement hors rotation ainsi que la reconstruction de l'image finale grâce au résultat du solveur

3.2 Nathan Fontaine

Une fois l'image rendue propre, il faut séparer tous les caractères de la grille pour pouvoir ensuite les faire reconnaître au réseau de neurones

3.3 Léo Menaldo

Un réseau de neurones était implémenté, il était capable d'apprendre la fonction XOR. Il faut l'améliorer afin de lui apprendre à reconnaître des caractères.

3.4 Grégoire Le Bihan

Pour que les utilisateurs de notre projet se sentent merveilleusement biens et ne soient pas perdus en essayant d'utiliser notre travail, une interface graphique sera implémentée

3.5 Récapitulatif

Assignement des tâches général				
Tâches / Membres	Dardan	Nathan	Léo	Grégoire
Image				
Pré-traitement	X	-	-	-
Découpage de l'image	-	X	-	-
Rotation (Manuelle + Auto)	-	-	-	X
Sudoku				
Solveur	-	-	-	X
Reconstruction de l'image	X	-	-	-
Autre				
Reconnaissance de caractères	-	-	X	-
Interface graphique	-	-	-	X

4 Avancement du projet

4.1 Le pré-traitement

4.1.1 Première idée

Pour la première soutenance, notre approche initiale comprenait l'utilisation d'un filtre de dégradation des couleurs en niveaux de gris, l'application d'un flou gaussien, et enfin, une opération de binarisation à double seuil. Ces étapes constituaient les préliminaires de la méthode de détection de bord de Canny. Notre intention initiale était d'implémenter l'intégralité de cette méthode en ajoutant un filtre de Sobel et un algorithme de "non-maximum suppression" à notre processus.
Cependant, suite à divers problèmes techniques et des résultats insatisfaisants, nous avons pris la décision de réviser notre approche de manière significative. Une des alternatives envisagées était l'utilisation de la dilatation et de l'érosion de l'image, combinées avec l'application d'un filtre de Sauvola. Dans cette nouvelle approche, nous avons donc exécuté les étapes suivantes dans cet ordre : conversion en niveaux de gris, dilatation, binarisation, et enfin l'application du filtre de Sobel.
En ce qui concerne l'explication de l'algorithme, le filtre de Sobel est un opérateur de traitement d'image qui permet de mettre en évidence les contours en calculant le gradient de l'image. Il utilise des masques de convolution pour détecter les variations d'intensité dans les différentes directions.
Quant à l'algorithme de "non-maximum suppression", il s'agit d'une technique couramment utilisée après l'application de filtres de détection de contours, tels que le filtre de Sobel. Son objectif est de réduire l'épaisseur des contours détectés en ne conservant que les pixels qui représentent les valeurs maximales locales le long des directions du gradient.
En ce qui concerne la dilatation et l'érosion, ce sont des opérations morphologiques largement utilisées en traitement d'image. La dilatation a pour effet d'élargir les contours des objets dans une image, tandis que l'érosion a l'effet inverse en rétrécissant ces contours.
Enfin, le filtre de Sauvola est spécifiquement conçu pour la binarisation d'images en conditions d'éclairage variables. Il adapte localement le seuil de binarisation en fonction des variations d'intensité dans l'image, ce qui peut améliorer la robustesse de la binarisation.

4.1.2 Implémentation

Cette année, nous avons fait un TP sur SDL2 dans lequel nous devions convertir une image en nuances de gris. Je m'en suis donc inspiré pour la méthode. Je charge donc une image dans une surface, puis crée une texture à partir de cette surface.
Il faut ensuite modifier la valeur des composantes r, g et b de chacun des pixels de la surface. Afin d'obtenir des nuances de gris, il faut que ces trois valeurs soient égales. Je calcule donc une moyenne à partir des trois données. Il faut alors remplacer le pixel par un pixel ayant la moyenne comme r, g et b.
Une fois l'intégralité de la surface remplacée, on crée alors une nouvelle texture à partir de la surface modifiée.
La surface en nuances de gris est sauvegardée comme image au format bitmap.
La seule ambiguïté du programme était la formule du calcul de la moyenne. Dan un premier temps, on peut supposer que le rouge, le vert et le bleu sont aussi sombres les uns que les autres et appliquer la formule :

$$average = 0.299 * Red + 0.587 * Green + 0.114 * Blue$$

FIGURE 1 – Exemple de grayscale d'image

La dilatation est une opération morphologique fondamentale en traitement d'image, utilisée pour modifier la forme ou la structure des objets dans une image. Elle s'applique principalement aux images binaires, mais peut également être adaptée à des images en niveaux de gris. L'opération de dilatation est définie par l'utilisation d'un élément structurant, généralement un petit noyau ou masque, qui se déplace sur toute l'image.
Élément Structurant : L'élément structurant est une petite matrice ou noyau qui définit la forme de la dilatation. Il peut prendre différentes formes, telles qu'un carré, un cercle ou une croix. La taille et la forme de cet élément structurant influent sur le résultat de l'opération de di-

latation.
Parcours de l'image : L'élément structurant est déplacé sur toute l'image, pixel par pixel. À chaque position de l'élément structurant, une opération est effectuée pour déterminer la valeur du pixel correspondant dans l'image dilatée.
Opération de Dilatation : Pour chaque position de l'élément structurant, l'opération de dilatation consiste à examiner la région de l'image sous l'élément structurant et à attribuer une valeur maximale à tous les pixels inclus dans cette région.
Effet sur les Objets : L'effet de la dilatation est de faire croître les objets dans l'image. Si le noyau de l'élément structurant correspond à la forme d'un objet dans l'image, la dilatation élargira cet objet en ajoutant des pixels à sa frontière.
Bordures : Lorsque l'élément structurant chevauche les bords des objets, il peut y avoir un effet de "débordement" où des pixels appartenant à différents objets se chevauchent. Cela peut créer une fusion des objets voisins.
Paramètre de Dilatation : La taille de l'élément structurant détermine l'ampleur de la dilatation. Plus l'élément structurant est grand, plus l'effet de dilatation est important. Cependant, il faut veiller à ne pas choisir un élément structurant trop grand, car cela pourrait conduire à une fusion excessive des objets.
En résumé, la dilatation est une opération qui étend les objets dans une image en attribuant la valeur maximale des pixels dans la région couverte par un élément structurant. C'est une étape essentielle dans de nombreuses applications de traitement d'image, telles que la segmentation d'objets et la détection de contours.

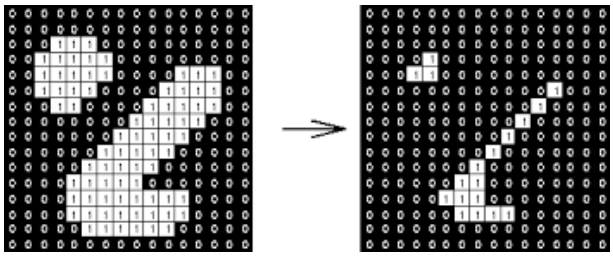

FIGURE 2 – Exemple de dilatation d'image

Le filtre de Sauvola est un algorithme de binarisation adaptative, souvent utilisé dans le domaine du traitement d'image pour améliorer la qualité de la binarisation, en particulier dans des conditions d'éclairage variables. Voici une explication détaillée du filtre de Sauvola :

Calcul de la Moyenne et de l'écart-type local : Le filtre de Sauvola fonctionne en calculant la moyenne locale et l'écart-type local dans une fenêtre glissante autour de chaque pixel de l'image. Cette fenêtre est généralement définie comme un voisinage carré.
Le filtre de Sauvola est un algorithme de binarisation adaptative, souvent utilisé dans le domaine du traitement d'image pour améliorer la qualité de la binarisation, en particulier dans des conditions d'éclairage variables. Voici une explication détaillée du filtre de Sauvola :

1. **Calcul de la Moyenne et de l'écart-type local :** Le filtre de Sauvola fonctionne en calculant la moyenne locale et l'écart-type local dans une fenêtre glissante autour de chaque pixel de l'image. Cette fenêtre est généralement définie comme un voisinage carré.
2. **Formule de Binarisation :** La formule de binarisation de Sauvola utilise la moyenne locale (μ) et l'écart-type local (σ) pour calculer un seuil adaptatif pour chaque pixel. La formule est la suivante :

$$T(x, y) = \mu(x, y) \times (1 + k \times (\frac{\sigma(x, y)}{R} - 1))$$

   — $T(x, y)$ est le seuil calculé pour le pixel à la position (x, y).
   — k est un facteur ajustable qui contrôle la sensibilité du seuil. Une valeur typique est 0.5.
   — R est la plage dynamique maximale des valeurs de pixel dans l'image (généralement la différence entre la valeur maximale et minimale).
3. **Binarisation :** Une fois que le seuil adaptatif est calculé pour chaque pixel, la binarisation est effectuée en comparant la valeur du pixel à son seuil. Si la valeur du pixel est supérieure au seuil, le pixel est considéré comme blanc ; sinon, il est considéré comme noir.
4. **Adaptabilité au Contraste :** L'avantage clé du filtre de Sauvola réside dans son adaptation dynamique au contraste local de l'image. Cela permet de gérer efficacement les variations d'éclairage et de contraste, rendant la binarisation plus robuste dans des conditions d'éclairage changeantes.
5. **Paramètres Ajustables :** Outre le facteur k, d'autres paramètres tels que la taille de la fenêtre de calcul de la moyenne et de l'écart-type peuvent également être ajustés en fonction des besoins spécifiques de l'application.

En résumé, le filtre de Sauvola offre une approche adaptative à la binarisation, permettant de maintenir une performance stable même dans des situations où l'éclairage varie. Son utilisation est courante dans des domaines tels que la reconnaissance de texte, où la qualité de la binarisation peut avoir un impact significatif sur les résultats.

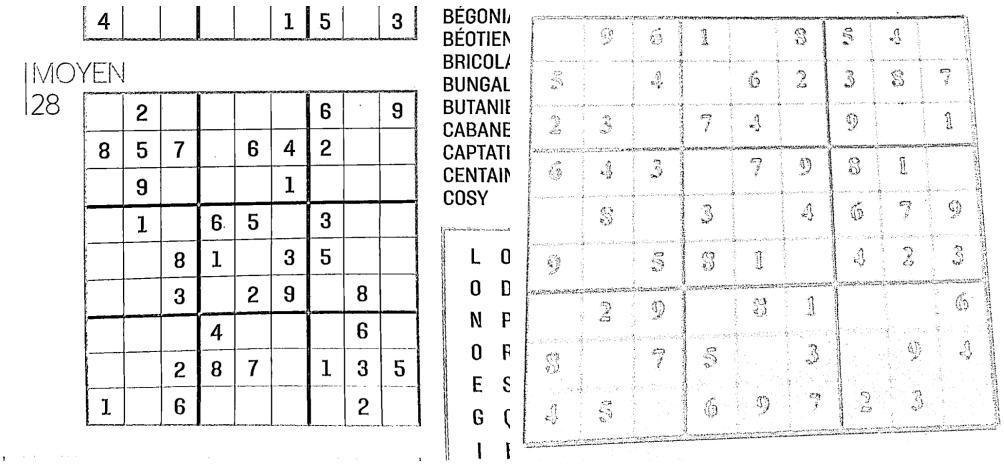

FIGURE 3 – Images 5 et 6 binarisées

Le filtre de Sobel est un opérateur de traitement d'image utilisé pour détecter les contours, en particulier les contours verticaux et horizontaux, dans une image. Il fait partie des filtres de détection de contour basés sur la convolution. La convolution est une opération mathématique qui consiste à combiner deux fonctions pour en produire une troisième. Dans le contexte du filtre de Sobel, la convolution est utilisée pour appliquer un masque de détection de contour à chaque pixel de l'image.
Le filtre de Sobel se compose de deux masques, un pour la détection des contours verticaux et l'autre pour la détection des contours horizontaux. Ces masques sont souvent appelés "noyaux de Sobel". Voici les deux noyaux de Sobel typiques :

Noyau de Sobel pour la détection des contours verticaux (Gx) :

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

Noyau de Sobel pour la détection des contours horizontaux (Gy) :

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

Le filtre de Sobel fonctionne en appliquant ces noyaux à chaque pixel de l'image en effectuant une opération de convolution. Voici comment cela fonctionne en détail pour la détection des contours verticaux (Gx) :

1. Placez le noyau de Sobel Gx sur le pixel central de l'image.
2. Multipliez chaque valeur du noyau par la valeur correspondante du pixel de l'image sous-jacente.
3. Sommez les résultats des multiplications.
4. Répétez ces étapes pour chaque pixel de l'image.

Le résultat de cette opération de convolution donne une nouvelle image, appelée "carte de gradients", qui représente les variations d'intensité des contours verticaux dans l'image d'origine. De manière similaire, on peut utiliser le noyau Gy pour détecter les contours horizontaux.

La carte de gradients résultante peut ensuite être utilisée pour identifier les contours et les bordures dans l'image. Habituellement, on combine les résultats des deux convolutions en calculant la magnitude du gradient comme suit :

$$\text{Gradient} = \sqrt{(Gx^2 + Gy^2)}$$

On peut également calculer l'angle du gradient pour déterminer la direction du contour. En pratique, le filtre de Sobel est souvent utilisé en conjonction avec d'autres techniques de traitement d'image pour améliorer la qualité de la détection des contours.

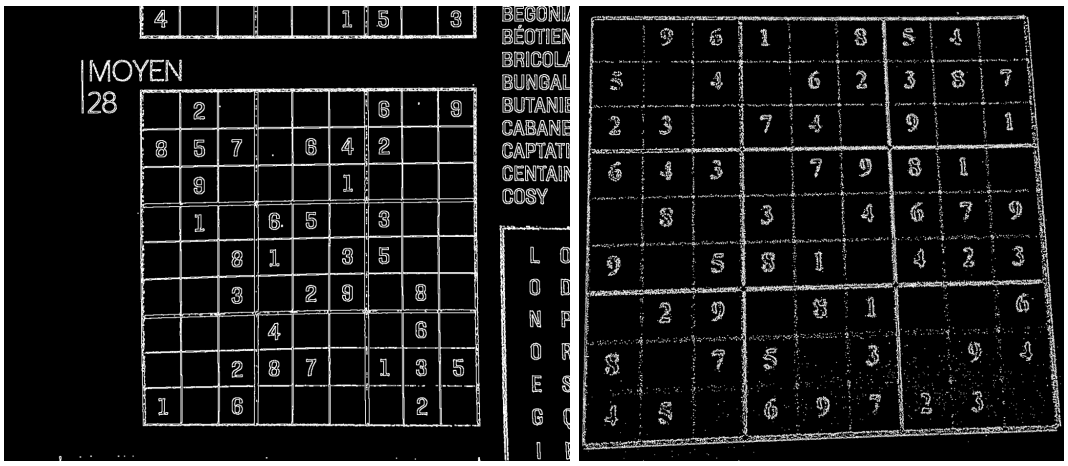

FIGURE 4 – Images 5 et 6 après application du filtre de Sobel

4.2 Reconstruction de l'image résultante

Le résultat généré par notre solveur est consigné dans un fichier texte au format spécifique, nécessitant ainsi la mise en place d'un parseur dédié pour son interprétation. De surcroît, il a été impératif d'effectuer une distinction entre l'ancienne grille et la nouvelle, afin de déterminer quels chiffres appartenaient à chacune et ainsi les afficher correctement à l'écran.
Ma méthode de reconstruction repose sur la création d'une image vide à partir de laquelle j'ai enregistré chaque chiffre en utilisant à la fois le rouge et le noir comme couleurs distinctives. Par la suite, je procède à la réintégration de chaque chiffre dans la grille en m'appuyant sur des images préexistantes.
Ainsi, le résultat final présente une composition visuelle où les anciens chiffres sont représentés en noir tandis que les nouveaux sont mis en évidence en rouge, offrant une visualisation claire et différenciée des deux ensembles de chiffres.

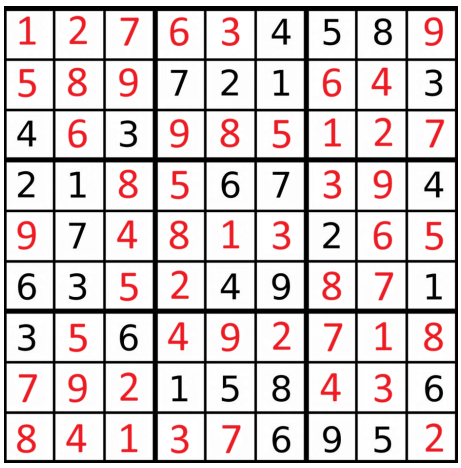

FIGURE 5 – Exemple d'image résultante

4.3 Le traitement de l'image

4.3.1 Détection des lignes

La question de la détection des lignes sur une image est un problème difficile. En effet il faut tout d'abord utiliser une image monochrome composée essentiellement de points, de plus ces images seront parfois incomplètes, en effet, selon la qualité de l'image d'origine, les images seront composées de plus ou moins de points et de bruits, pour faire face à ce problème, nous avons utilisé une technique appelée "transformée de Hough". Il existe deux procédés différents :

— Une méthode avec un accumulateur contenant deux paramètres (m, c). Ce qui équivaut à avoir un point de coordonnées (x, y) quelconques, qui peut être contenu dans une droite de fonction affine $y = mx + c$ avec m un multiplicateur et c une constante.

— Une autre méthode qui utilise elle aussi un accumulateur, mais cette fois de paramètres (θ, ρ). Ce qui équivaut à avoir un point de coordonnées (x, y) quelconques, qui peut être contenu dans une droite sur le plan paramétrique trigonométrique avec $x \times sin(\theta) - y \times cos(\theta) + \rho = 0$ avec ρ l'ordonnée à l'origine sur le plan paramétrique et θ l'angle par rapport à l'axe horizontal.

Méthode avec un accumulateur (m, c) :
Mise en place : L'accumulateur (m, c) stocke le nombre de points qui satisfont l'équation de chaque ligne. Les valeurs de m et de c sont choisies de manière à couvrir toutes les lignes potentielles dans l'espace bidimensionnel.
Avantages : Cette méthode est relativement simple à comprendre et à mettre en œuvre. Elle peut être efficace pour détecter des droites dans des espaces où les lignes sont bien représentées par des équations linéaires. De plus, les paramètres m et c peuvent être interprétés directement comme les caractéristiques de la droite détectée.
Problèmes : Cette méthode peut être limitée dans sa capacité à détecter des lignes qui ne peuvent pas être facilement représentées par une équation linéaire. De plus, elle peut rencontrer des problèmes de précision lorsque les lignes ne sont pas parfaitement droites ou lorsque les données sont sujettes à du bruit ou des imprécisions. Qui plus est, lorsque l'image est grande, des erreurs de "Segmentation fault" peuvent survenir de manière inexplicable et incongrue, ce qui m'a poussé à me tourner vers la seconde méthode. De plus cette méthode consomme beaucoup plus de mémoire étant donné qu'elle génère énormément de fois plus de droite suite aux différents m et c possibles.

Méthode avec un accumulateur (θ, ρ) :
Mise en place : L'accumulateur (θ, ρ) stocke le nombre de points qui satisfont l'équation de chaque ligne. Les valeurs de θ et de ρ sont choisies pour couvrir toutes les lignes potentielles dans l'espace bi-dimensionnel.
Avantages : Cette méthode est robuste pour détecter des lignes dans des espaces où les droites peuvent avoir différentes orientations. Elle peut mieux gérer les cas où les lignes ne sont pas parfaitement droites ou lorsque les données contiennent du bruit.
Problèmes : La méthode (θ, ρ) peut être plus complexe à mettre en œuvre que la méthode (m, c), car elle implique des opérations trigonométriques. De plus, l'interprétation des paramètres θ et ρ peut être moins directe que celle de m et c dans certaines applications.

4.3.2 Découpage de l'image

Première approche du problème :
Pour résoudre ce problème, on voulait tout d'abord avoir la liste des points qui sont en intersections sur les lignes afin d'obtenir des résultats de découpage quasi parfait en utilisant les coordonnées de ces points, mais pour cela il fallait qu'on implémente les fonctions suivantes :

- Une fonction qui supprime les lignes en dehors du sudoku afin de garder seulement celle qui représente les vrais points d'intersections de la grille
- Une fonction qui supprime le surplus de lignes dans la grille du sudoku afin d'avoir le bon nombre de points de points d'intersections
- Une fonction qui supprime le surplus de points détectés, en effet parfois, on peut détecter trop de lignes et lors du découpage cela pose problème car la découpe s'effectue sur les coordonnées de points d'intersections, donc s'il venait à avoir trop de points, le découpage pourrait par exemple se faire sur la moitié d'une case et fausserait tout le reste.
- Une fonction qui ajoute des points lorsqu'il manque des points, en effet parfois il manquera des points car certaines lignes n'ont pas été détecté et lors du découpage cela pose problème car la découpe s'effectue sur les coordonnées de points d'intersections, donc s'il venait à manquer des points, le découpage pourrait par exemple se faire sur plusieurs cases ce qui est problématique.

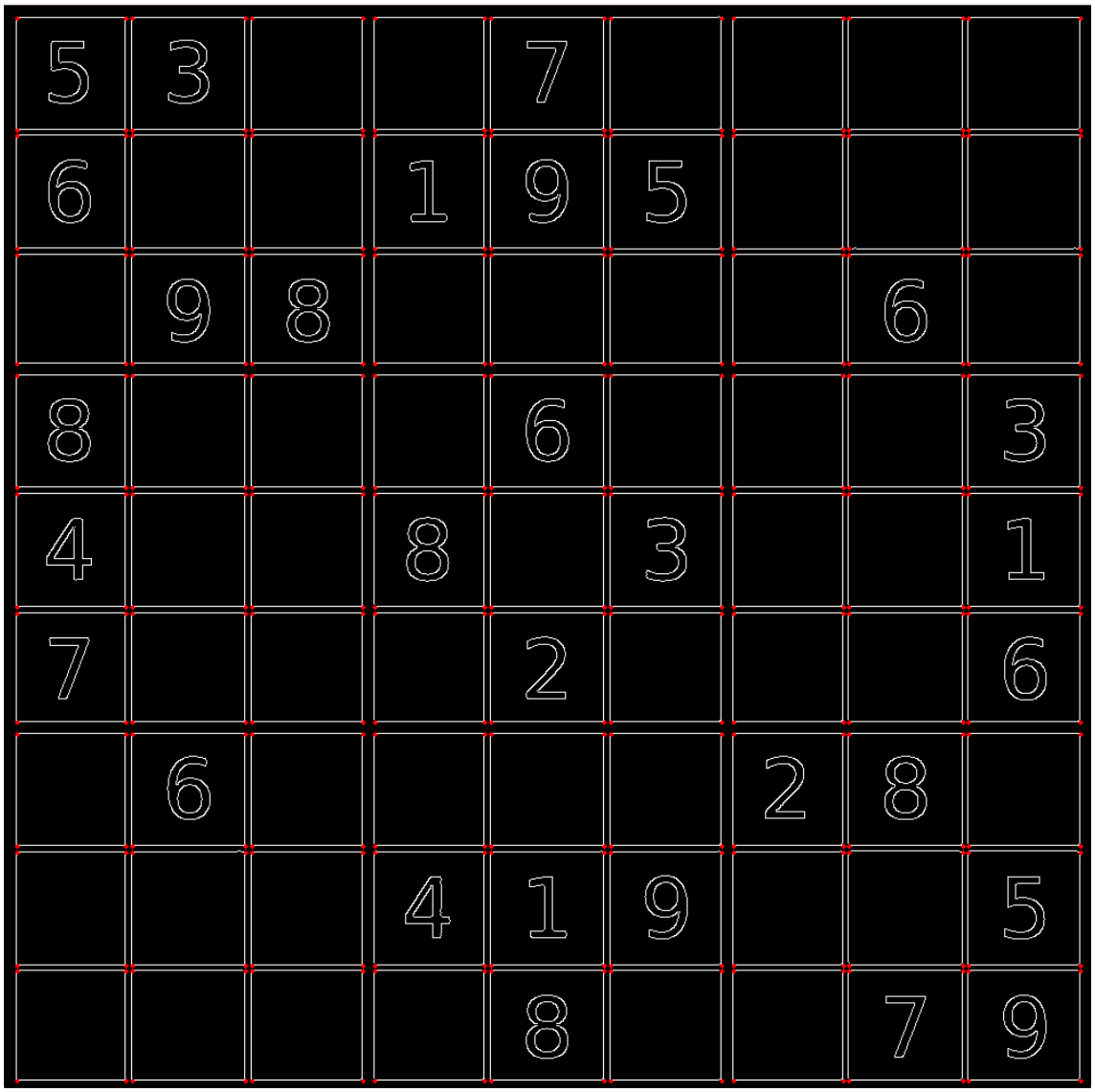

FIGURE 6 – Premiers test de détection de points

Comme on peut le voir ci-dessus la détection des points pour l'image 1 fonctionne parfaitement, ce qui nous avais laissé penser que cela allait fonctionner pour le reste des images

Mais lors du test sur d'autres images, j'ai aperçu qu'il pouvait manquer des points sur l'image même avec une valeur de seuil très petite, or j'ai essayé d'implémenter différentes manières pour résoudre ce problème mais la résolution n'était jamais fiable, soit il ne rajoutait pas les points aux endroits correctes, soit il manquait des valeurs dans la matrice et lors du parcours cela provoquait des débordements d'indexation dans la matrice ce qui faisait éventuellement crache le programme.

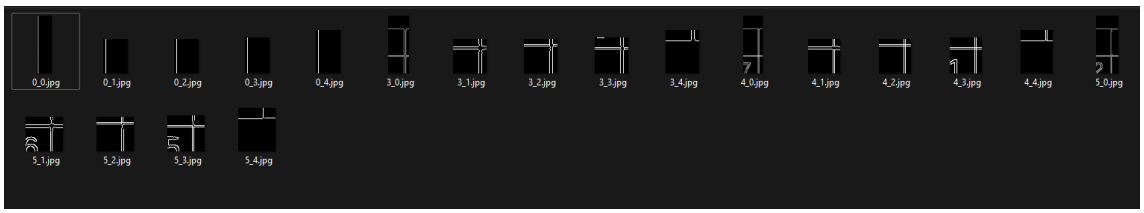

FIGURE 7 – Exemple de découpage raté avec le principe des points

Ainsi, avec cette fonction, le nouveau problème de comment rajouter ou supprimer les points aux bons endroits n'a pas pu être résolue de manière constante sur toute les images. On a alors considérer une nouvelle approche, qui semble à la fois plus simple mais également plus bête, le principe est de chercher les coordonnées du point tout en haut à gauche du sudoku et de celui tout en bas à droite, je m'étais d'abord résolu à ne pas utiliser cette implémentation car aux premiers abords, on pense directement que cette fonction aura des problèmes à découper correctement les cases, étant donné que nous avons utilisé une base de données pour l'ia qui ne traite pas le cas où les images peuvent avoir des bords ou des résidus sur les bords de l'images mais vu la situation problématique dans laquelle nous nous trouvions, je me suis résolu à essayer cette technique et il c'est avéré que c'était plus facile à implémenter, de plus, pour pallier à ce problème de bords, j'ai simplement réduit les bords des cases du sudoku sur l'image finale avec un seuil prédéterminé et proportionnelle à la taille de la sous_image. Avec ce principe on obtient des images correctes (moins bien qu'avec le découpage par points) mais satisfaisant tout de même.

4.3.3 Redimensionner les images

Ainsi vient un nouveau problème avec ces images : comment faire pour les redimensionner correctement ? En effet, si l'on essaie de convertir les images d'une taille par exemple de 100 par 100 vers une images de 28 par 28 (taille requise pour l'ia), il vient que ces problèmes auront des pertes de données sur les certains pixels ce qui rend non reconnaissable certains chiffres,

4.4 Le réseau neuronal

Pour cette fin de projet, il a fallu améliorer le réseau neuronal pour qu'il corresponde à nos besoins. On passe d'une simple fonction XOR, à une machine capable de reconnaître des caractères. Pour cela, plusieurs modifications ont pu être apportées et énormément de pistes ont été étudiées.

4.4.1 Le choix de la structure du réseau

Pour avoir un réseau neuronal capable de reconnaître au mieux des caractères, il faut être sûr de sa structure, faut-il plusieurs couches cachées ? Combien de neurones par couches ?...
Après avoir lu énormément d'articles de recherche au sujet de la reconnaissance de caractères, le modèle qui ressortait le plus souvent était un modèle comportant 2 couches cachées, contenant chacune 16 neurones, et après application, il manquait énormément de précision à nos tests, avec environ 10% de précision en général. J'ai pendant longtemps pensé que quelque chose m'échappait, mais après avoir revu et ré-assimilé le principe de propagation arrière pour l'apprentissage du réseau de neurones, je me suis rendu compte que le manque de précision pouvait être dû au faible nombre de neurones présentes sur mes couches cachées. En effet, avec cette configuration, le réseau était plus susceptible de converger vers des mauvaises valeurs sans pouvoir les changer par la suite.
J'ai ensuite essayé de créer des configurations loufoques jusqu'à avoir des résultats satisfaisants, en passant par des réseaux possédant 5 à 6 couches cachées en vain. Et si au final, une structure similaire à la structure utilisée pour l'apprentissage du XOR était suffisante ? Et si un réseau neuronal possédant une seule et unique couche cachée était la solution au problème. Cela s'est avéré être le cas puisqu'avec une couche cachée comptant 100 neurones, on s'est instantanément approchés d'une précision excellente, passant de 10% de précision à 95%.

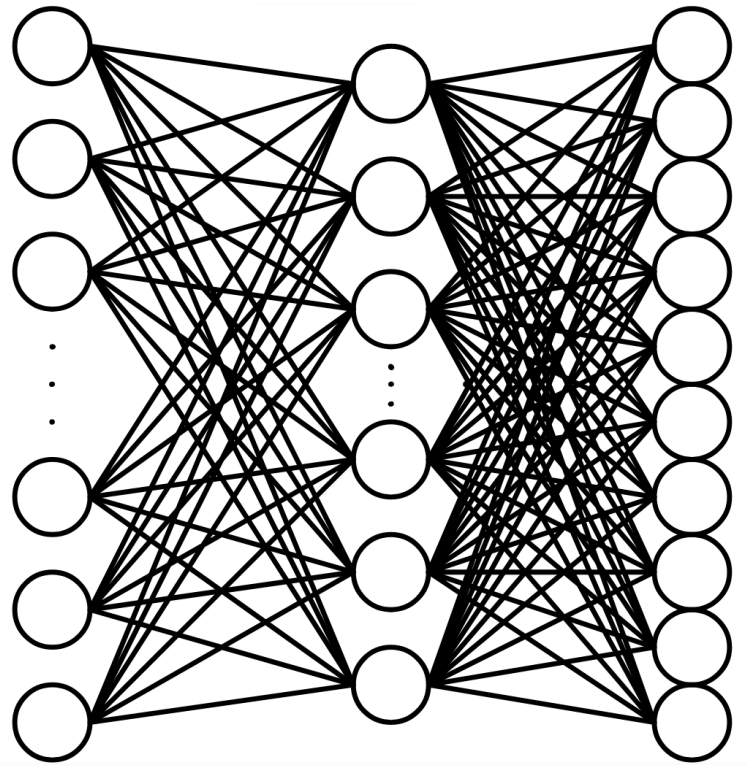

FIGURE 8 – Notre belle structure du réseau. 784 Entrées, 100 cachées, 10 sorties

4.4.2 Les fonctions d'activation

Pour l'apprentissage de la fonction XOR au réseau de neurones, nous avions utilisé la fonction sigmoïde : $f(x) = \dfrac{1}{1 + e^{-\lambda x}}$.

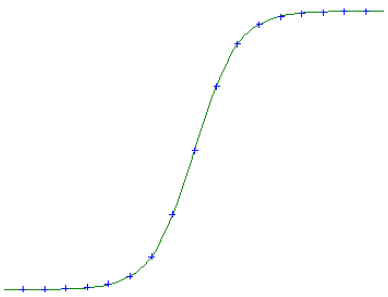

FIGURE 9 – La fonction sigmoïde

Ainsi que sa dérivée $f'(x) = f(x) \times (1 - f(x))$ car c'était la plus adaptée pour cette situation.
Pourtant, il existe bien d'autres fonctions d'activation, comme la fonction de redressage (ReLU), notée $f(x) = max(0, x)$, ou encore leaky-ReLU, notée $f(x) = max(0.01x, x)$. Il était aussi possible de "customiser" la fonction ReLU en lui donnant une variable capable de remplacer la valeur $0.01x$ de LeakyReLU.

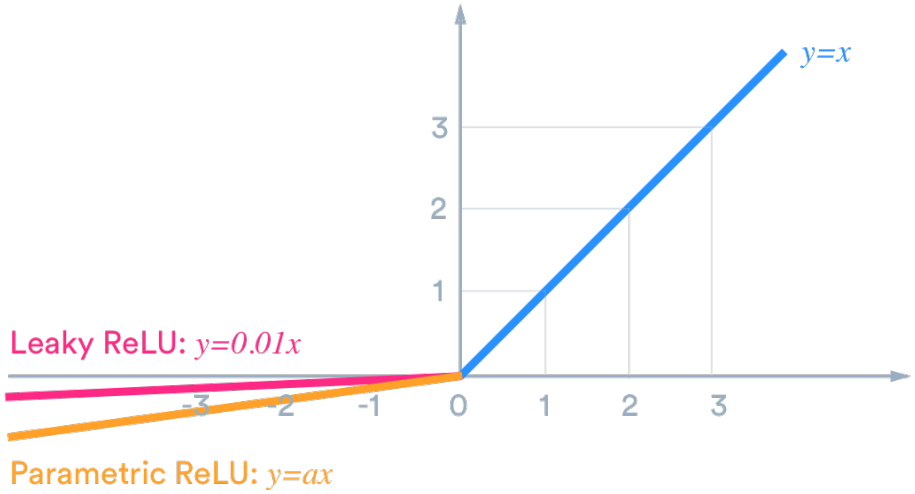

FIGURE 10 – La fonction ReLU, sous toutes ses formes

Après plusieurs essais, ces fonctions ne donnaient pas de résultats satisfaisants, elles sont utiles lorsque le réseau comporte beaucoup de valeurs négatives, or, dans tous les essais effectuées, à chaque fois, le réseau ne comportait que très peu de valeurs négatives.
Une autre fonction rivalisait très bien avec la fonction sigmoïde, la fonction Softmax, notée $f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$ qui était très pratique pour notre manière de fonctionner sous forme de représentation matricielle. Elle était parfaite et se prêtait très bien à son fonctionnement, son seul problème était qu'elle laissait les neurones apprendre trop vite et on se rapprochait donc trop rapidement de résultats faux. Après toutes ces tentatives, nous sommes donc restés sur notre choix initial, les fonctions sigmoïde et sa dérivée, qui avaient déjà portées leurs fruits.

4.4.3 Problèmes

Cela fait maintenant quelques temps que l'on progresse dans l'avancement du réseau de neurones, mais n'y a-t-il pas un terme qui reste sans explication ? De quoi parle-t-on vraiment lorsqu'on parle du taux de précision ?
C'est là que le réseau de neurones intervient et que tout ce qui a été vu précédemment va s'avérer être utile. Le taux de précision est un nombre représentant le ratio $\frac{Images\ trouvees}{Images\ donnees}$ sur un ensemble d'images.
Mais donc, est-il possible de ne pas trouver le bon caractère présent sur une image ? Faisons un essai.

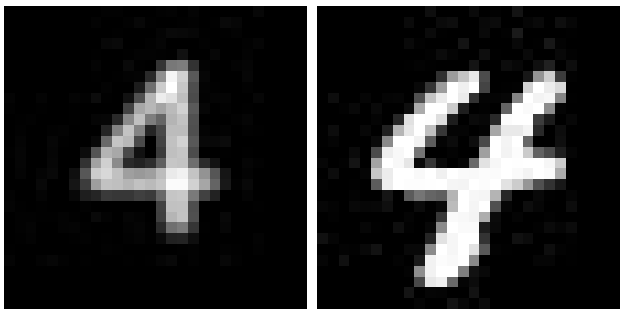

FIGURE 11 – Images de '4'

Ici, vous n'aurez sans aucun doute reconnu deux images du chiffre '4'. L'un deux est dactylographié, l'autre est écrit à la main. Mais maintenant, essayons de reconnaître les caractères suivants.

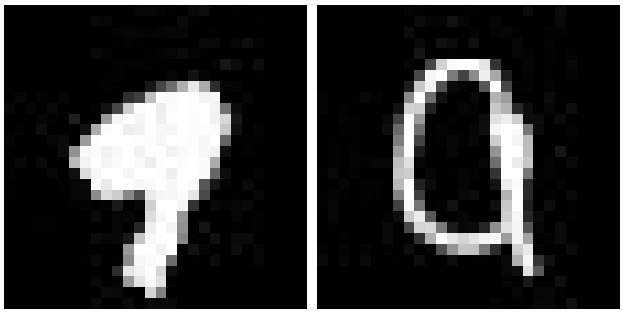

FIGURE 12 – Images de '9'

Ces deux images sont belles et biens deux images représentant le chiffre '9', et cela ne forme ni "10", ni "1a", mais bien "99". Alors si même l'humain a du mal à reconnaître ces caractères, qu'en est-il du réseau ? Lui aussi est incapable de reconnaître correctement des chiffres pareils. Malheureusement, il n'y a pas que ce point qui nous ammène à des problèmes de précision, les résultats peuvent varier selon la taille et la position du chiffre sur l'image ou de sa qualité. De plus, les résultats peuvent varier d'un réseau à l'autre puisque l'on part de données aléatoires, et que, plus la durée d'entraînement sera longue, plus les résultats auront de chances d'être justes.

4.4.4 Sauvegarde et Retranscription du réseau

Afin de ne pas avoir à faire un entraînement de 30 minutes avant chaque utilisation, il est préférable d'enregistrer les valeurs présentes dans le réseau dans un fichier, ce fichier contiendra donc tous les poids et les biais du réseau, ainsi que le nombre de neurones de chaque couche. Ce fichier sera ainsi très utile pur reconnaître les caractères de la grille de sudoku, car une fois le réseau entraîné, il n'est plus nécessaire de

réapprendre au réseau comment reconnaître des caractères, il lui suffira simplement de charger un fichier.

4.4.5 Récapitulatif

Pour faire fonctionner notre réseau de neurones et faire en sorte qu'il soit capable de reconnaître des caractères, la meilleure option était de créer un réseau neuronal à 3 couches.

— Une couche d'entrée : Contenant 784 neurones, correspondant aux nombre de pixels d'une image
— Une couche cachée : Contenant 100 neurones
— Une couche de sortie : Contenant 10 neurones, qui représente la proposition du réseau devant une image

Même si nous ne reconnaissons pas les caractères à 100%, nous gardons tout de même un taux de précision élevé qui suffira pour le projet. Faire cette "intelligence artificielle" fût un vrai plaisir et un honneur pour moi car c'était de loin un challenge extrêmement difficile qui me paraissait impossible et qui m'aura fait souffrir, mais j'étais aussi à la recherche de cela...

4.5 Interface graphique

Pour cette deuxième soutenance, le panorama de mon travail s'est considérablement élargi. Bien que j'ai déjà exploré la diversité des fonctionnalités lors de la première soutenance, notamment avec l'intégration du solveur et la rotation d'image, cette deuxième phase a été marquée par une variété encore plus importante. La continuité de certaines fonctionnalités a contribué à cette diversité, mais l'effort s'est intensifié avec l'inclusion d'éléments tels que la tentative d'auto-rotation de l'image et la création d'une interface utilisateur. Le temps s'est avéré être notre pire ennemi au cours de cette deuxième partie du projet, soulignant la nécessité d'une gestion temporelle plus efficace. Une réflexion sur l'amélioration de ma gestion du temps devient ainsi impérative pour optimiser le rendement de ce projet en évolution constante.

4.5.1 Solveur

Mon algorithme de résolution se divise en plusieurs composantes distinctes. Dans un premier temps, un programme se charge de mettre en place les tableaux nécessaires à mes travaux ultérieurs. Ensuite, un second programme prend en charge les vérifications des règles du sudoku.

Le cœur du projet réside dans l'algorithme récursif, qui sollicite les programmes de vérification à chaque tentative de placement de valeurs. J'ai également mis en œuvre une fonction de résolution qui lit un fichier selon la structure requise, complète le sudoku, puis renvoie le résultat sous forme de fichier, reproduisant ainsi le processus en sens inverse. Pour cela, j'ai simplement lu les caractères du fichier et les ai intégrés aux tableaux déjà en place.

```
greg@PC-en-manque-de-fer:~/ocr/epita-prepa-asm-PROJ-OCR-2027-lyn-011/solver$ ./sudoku_solver grid_00
+-------+-------+-------+
| . . . | . . 4 | 5 8 . |
| . . . | 7 2 1 | . . 3 |
| 4 . 3 | . . . | . . . |
+-------+-------+-------+
| 2 1 . | . 6 7 | . . 4 |
| . 7 . | . . . | 2 . . |
| 6 3 . | . 4 9 | . . 1 |
+-------+-------+-------+
| 3 . 6 | . . . | . . . |
| . . . | 1 5 8 | . . 6 |
| . . . | . . 6 | 9 5 . |
+-------+-------+-------+
| 1 2 7 | 6 3 4 | 5 8 9 |
| 5 8 9 | 7 2 1 | 6 4 3 |
| 4 6 3 | 9 8 5 | 1 2 7 |
+-------+-------+-------+
| 2 1 8 | 5 6 7 | 3 9 4 |
| 9 7 4 | 8 1 3 | 2 6 5 |
| 6 3 5 | 2 4 9 | 8 7 1 |
+-------+-------+-------+
| 3 5 6 | 4 9 2 | 7 1 8 |
| 7 9 2 | 1 5 8 | 4 3 6 |
| 8 4 1 | 3 7 6 | 9 5 2 |
```

L'implémentation de la résolution du sudoku en elle-même ne m'a pas posé de difficultés majeures, bien que l'ajustement des valeurs de retour et la gestion des booléens m'aient parfois amené à réévaluer mon programme. Une part importante de mon temps a été dédiée à la consultation de la documentation en ligne pour la lecture et la création de fichiers, ainsi qu'à l'organisation de mon code pour faciliter la détection d'éventuelles erreurs. Bien que des erreurs liées aux types de données et des erreurs de syntaxe aient émergé, l'ensemble du processus s'est déroulé sans accroc, en grande partie grâce à la structure efficace du fichier d'entrée, facilitant la distinction des emplacements des chiffres et des sections.

4.5.2 Rotation

Contrairement à ce qui fût énoncé durant la première soutenance nous n'avons pas changé la manière de faire notre rotation et avons conservé la rotation SDL.
Pour rappel, La fonction SDL_RenderCopyEx est une fonction clé de la bibliothèque SDL (Simple DirectMedia Layer), largement utilisée pour le développement de jeux et d'applications multimédias. Cette fonction permet de copier une texture sur le rendu, en offrant la possibilité d'ap-

```
greg@PC-en-manque-de-fer:~/ocr/epita-prepa-asm-PROJ-OCR-2027-lyn-011/solver$ cat grid_00
... ..4 58.
... 721 ..3
4.3 ... ...

21. .67 ..4
.7. ... 2..
63. .49 ..1

3.6 ... ...
... 158 ..6
... ..6 95.
greg@PC-en-manque-de-fer:~/ocr/epita-prepa-asm-PROJ-OCR-2027-lyn-011/solver$ cat grid_00.result
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

356 492 718
792 158 436
841 376 952
greg@PC-en-manque-de-fer:~/ocr/epita-prepa-asm-PROJ-OCR-2027-lyn-011/solver$
```

pliquer des transformations de manière aisée.
La signature de la fonction se fait de la manière suivante, regardons en détail ces paramètres :

- Renderer : Le pointeur vers le rendu sur lequel la texture sera copiée.
- Texture : La texture source que vous souhaitez copier.
- Srcrect : Un pointeur vers la zone rectangulaire de la texture source que vous voulez copier, ou NULL pour copier toute la texture.
- Dstrect : Un pointeur vers la zone rectangulaire du rendu où vous souhaitez copier la texture.
- Angle : L'angle de rotation en degrés que vous souhaitez appliquer à la texture lors de la copie.
- Center : Un point autour duquel la rotation sera effectuée. Si NULL, la rotation se fait autour du centre de la texture.
- Flip : Une valeur qui détermine comment la texture doit être retournée : SDL_FLIP_NONE, SDL_FLIP_HORIZONTAL, SDL_FLIP_VERTICAL, ou une combinaison de ceux-ci).

Cette fonction est extrêmement utile pour animer des objets à l'écran, permettant une variété d'effets visuels tels que la rotation et le renversement.
Ceci dit celle-ci pose un problème pour l'enregistrement de fichier a cause du format du fichier qui sera utilisé. Nous devons forcer un passage par le Pixel color de SDL ce qui veut dire une perte de performances mémoires mais une augmentation de la qualité de la photo rendu au final.

4.5.3 User Interface

Dans la phase initiale de notre projet, le processus de conception a débuté par une délibération sur les fonctionnalités à intégrer dans notre application. Notre vision principale consistait à disposer, à gauche de la fenêtre, des différentes actions que l'utilisateur pouvait entreprendre, tandis qu'à droite, une visualisation en temps réel de l'image était envisagée. Pour matérialiser cette séparation, nous avons utilisé un Gtk_Paned. Le choix du fichier sur lequel appliquer notre OCR constituait une étape cruciale. Deux options s'offraient à nous : définir le fichier via un argument dans la commande de lancement de l'application ou offrir à l'utilisateur la possibilité de le sélectionner directement depuis la fenêtre Gtk. La seconde option a été préférée, car elle conférait la flexibilité d'utiliser notre solveur de sudoku sur différentes images sans nécessité de relancer l'application. À cet effet, nous avons intégré un gtk_File_Chooser_Button. Pour accroître la convivialité, nous avons envisagé la possibilité d'exécuter le solveur en une seule opération ou d'effectuer chaque étape séparément. Cette flexibilité a été implémentée à l'aide de différents Gtk_Button, chacun déclenchant une partie spécifique du code, à l'exception du dernier qui consolide l'ensemble des étapes avant d'afficher le résultat final. Le problème étant que nous n'avons pas pu comprendre comment tout effectuer automatiquement en raison de nombreux paramètre nécessaire au pré-traitement. L'auto-rotation est implémentée mais ça ne suffit pas pour tout effectuer immédiatement sans utiliser un programme brute force. Puisque l'étape de rotation manuelle était un élément incontournable du processus, nous avons créé un espace où l'utilisateur pouvait spécifier l'angle de rotation souhaité. En outre, la nécessité de paramètres pour le pré-traitement nous a amené à introduire une zone de texte dédiée à la saisie de ces derniers.
La première phase de l'implémentation de l'interface en C s'est concentrée sur la création des structures "application" et "userinterface". La première englobe la seconde, ainsi qu'un espace destiné au nom d'un fichier. La seconde, quant à elle, intègre les différents éléments de l'interface, tels que la fenêtre, la zone d'affichage de l'image, les boutons de sélection de fichier et de pré-traitement, la zone de texte pour l'angle de rotation, le bouton pour la rotation manuelle, le déclencheur de segmentation, le bouton de reconnaissance de caractères, le solveur de sudoku avec affichage de la grille résolue sous forme d'image, et enfin, le bouton "all" pour exécuter l'ensemble des étapes en une seule fois. La phase suivante consistait à élaborer les fonctions associées à chaque bouton,

définies lors de leur activation. La fonction du sélecteur de fichier, par exemple, assigne le chemin du fichier choisi à l'attribut "filename" de l'application et l'affiche. Une fois le fichier sélectionné, seules les étapes suivantes possibles sont le pré-traitement et l'exécution complète du processus, désactivant ainsi les autres boutons. La fonction liée au pré-traitement ajuste l'attribut "filename" avec le chemin de l'image traitée, ouvrant ainsi l'accès à la segmentation et à la rotation manuelle. La fonction du bouton de rotation modifie directement l'image sans nécessiter de modification de l'attribut "filename". La fonction associée au bouton de segmentation permet, quant à elle, d'accéder à la reconnaissance de caractères, débloquant à son tour l'accès au solveur de sudoku. Cette structuration claire des fonctions permet une progression logique à travers les différentes étapes du processus, offrant à l'utilisateur une expérience intuitive et fluide.

5 Conclusions

5.1 Dardan

Ce projet a été la validation de mes choix de majeurs. À présent, je suis pleinement certain de ma volonté de me consacrer professionnellement au traitement d'images. Grâce à cette expérience, j'ai exploré de nombreux concepts et eu l'occasion de découvrir des individus dont les travaux m'ont profondément inspiré, à l'instar d'Acerola, un programmeur d'images dont la chaîne YouTube est une source d'inspiration majeure. De plus, la relation directe entre les images et l'art visuel, un domaine qui suscite un intérêt particulier chez moi, m'a doté de connaissances essentielles pour entreprendre divers projets personnels. La dynamique de groupe a été positive, rendant le projet non seulement instructif mais aussi agréable à réaliser.

5.2 Léo

Ces quelques mois de travail acharné m'ont permis de dépasser mes propres limites que je ne pensais jamais atteindre, après avoir fait preuve d'une grande patience et surtout d'un calme majestueux et après m'être retourné le cerveau des dizaines de fois. Je suis passé par toutes les émotions, des erreurs bêtes qui m'ont faites perdre un temps fou, être forcé de recommencer un code car la piste abordée était mauvaise et donnait de mauvais résultats. Une suite de casse-têtes qui paraissent simples à résoudre qui m'ont causés beaucoup de tort.

Je suis tout de même très fier de ce que j'ai accompli et en bref, je me pensais incapable de faire une "intelligence artificielle".

5.3 Grégoire

Participer à un projet de groupe m'a apporté une expérience enrichissante sur plusieurs aspects. Sur le plan technique, j'ai renforcé mes compétences en programmation, en conception d'interfaces et en résolution de problèmes concrets. Sur le plan collaboratif, j'ai appris à travailler efficacement avec des membres d'équipes aux compétences diverses, à gérer les délais et à coordonner les tâches. En outre, le projet m'a offert une perspective précieuse sur la dynamique de groupe, la communication et la nécessité d'adaptabilité dans un environnement de travail collaboratif. Globalement, cette expérience a contribué à mon développement personnel en consolidant mes compétences techniques et en renforçant mes aptitudes en travail d'équipe.

6 Postlude

Nous y sommes, le projet est terminé, ce fût une épreuve très compliquée par laquelle nous étions obligés de passer. Mais cela nous a appris énormément de choses et nous a fait découvrir une quantité astronomique de branches de l'informatique, ce projet a aussi réconforté Dardan dans son idée de s'orienter vers la majeure IMAGE et pour Léo, de s'orienter vers la majeure SCIA.
Ce projet nous a aussi permis de maîtriser encore mieux les outils que l'ont nous a fait découvrir à l'école.
Et pour l'avenir du projet ? Nous ne comptons pas nous en arrêter là, on imagine déjà pouvoir utiliser le réseau neuronal de Léo pour essayer de reconnaître encore plus de caractères différents