SUDOCODE

OCR - Sudo-code : Rapport de soutenance.

SOUTENANCE 1.

OUFFA Rawane - LOPES Christina - HAMDOUN Zaky - DHEHIBI Dima

EPITA - 66 Rue Guy Môquet, 94800 Villejuif

S3 - B1

Villejuif, France, Dimanche 30 Octobre 2022.

”Les détails font la perfection, et la perfection n’est pas un détail.”
Léonard de Vinci.

Table des matières

1 Avant-Propos.

1.1 Présentation.

Dans le cadre du projet du troisième semestre à l'EPITA, nous allons mettre en oeuvre les connaissances acquises lors des TDs et TPs, ainsi que nos capacités à rechercher de nouvelles informations et méthodes. Ce projet se voit être temporairement divisé en deux dates clés : elles sont celles de la première et deuxième soutenance.

Ce document est un rapport de soutenance. Il a pour but de mettre en exergue l'avancement du projet entre la formation du groupe ainsi que la première soutenance. Il met également en avant les prochaines étapes à suivre et la composition du groupe.

1.2 Sujet.

L'objectif principal de ce projet est de former le groupe à l'utilisation de divers outils informatiques. Principalement, le groupe devra mettre en place un réseau de neuronnes, une interface, et tout un système de traitement d'images.

Par ailleurs, ce projet entraîne également les compétences organisationnelles des membres du groupe, afin de perfectionner l'utilisation de git, et, plus généralement, permettre une amélioration des aptitudes à travailler en groupe.

1.3 Composition du groupe.

Le groupe est composé de quatre personnes. Le nombre d'individus présents dans le groupe est un atout : ce dernier garantit la diversité des opinions vis-à-vis des fonctionnalités et caractéristiques du groupe, mais également la possibilité d'effectuer une division des tâches efficace en fonction des spécialités de chacun. Ci-dessous la liste de ceux composant le groupe Sudo-Code :

Chef de groupe : Mr. HAMDOUN Zaky.

- Passionné par l'informatique et la géopolitique, j'ai rejoint l'EPITA avec plus tard l'envie de devenir ingénieur en informatique spécialisé en cybersécurité, ou en big data et intélligence artificielle. Cependant, mon penchant pour la géopolitique et les métiers de la défense font également que je suis intéressé par l'Open Source Intelligence (OSINT) et les renseignements, qui se résument très souvent à du traitement d'image, comme c'est le cas dans ce projet. J'aime également les mathématiques et la philosophie.

LOPES Christina.

- De nature curieuse, je suis autant intéressée par les sciences dures que les sciences humaines, avec une attirance particulière pour les concepts complexes au premier abord. C'est d'ailleurs ce qui alimente mon intérêt pour l'informatique, qui est l'union de plusieurs domaines scientiques bien distincts comme l'éléctronique, l'algorithmique, l'architecture

des données sans oublier les concepts mathématiques et les notions mises en avant dans la formation à l'EPITA. De plus l'envie de comprendre et d'en apprendre davantage sur ces notions, est ma principale source de motivation.

DHEHIBI Dima.

- Attentive et passionnée par les nouvelles technologies, j'ai le goût pour la découverte de nouvelles formes de communications et de résolutions de problématiques variées. C'est ce qui fait notamment mon intérêt pour la vocation d'ingénieure informatique, au sein de laquelle j'ai la certitude de pouvoir m'épanouir dans le futur. Les dernières innovations dans le domaine de la physique quantique m'inspirent autant dans leur formulation que leur développement. Il me tarde de voir la résolution de sudokus sous forme quantique.

OUFFA Rawane.

- Attirée depuis très jeune par la Science sous toutes ses formes, j'ai d'abord commencé par me passionner pour l'astrophysique, autant le côté historique que technique, ne cherchant par là qu'à satisfaire ma soif de savoir. Cela m'a amené à remettre en question le monde dans lequel j'évoluais et interagissais. Par la suite, j'ai entrepris de considérer plus sérieusement l'informatique pour son potentiel "créer quelque chose à partir de rien". Appréciant énormément la lecture de livres de fiction, j'ai vu la capacité de l'imagination à se déployer dans l'informatique. Par ailleurs, les découvertes et innovations informatiques font que l'on n'a jamais finit d'apprendre !

1.4 Contact.

— Mr. HAMDOUN Zaky :
  — Login : zaky.hamdoun
  — Adresse e-mail : zaky.hamdoun@epita.fr
  — Numéro de téléphone : +33 6 69 69 69 59

— Mme OUFFA Rawane :
  — Login : rawane.ouffa
  — Adresse e-mail : rawane.ouffa@epita.fr
  — Numéro de téléphone : +33 6 58 74 44 52

— Mme DHEHIBI Dima :
  — Login : dima.dhehibi
  — Adresse e-mail : dima.dhehibi@epita.fr
  — Numéro de téléphone : +33 6 63 76 10 64

— Mme LOPES Christina :
  — Login : christina
  — Adresse e-mail : christina.lopes@epita.fr
  — Numéro de téléphone : +33 6 63 52 29 98

La communication entre les membres du groupe se fait pricipalement via un **serveur Discord dédié**. Un contact peut également être établi avec eux par l'intermédiaire de ce réseau social. À noter qu'une communication présentielle est tout de même privilégiée au

sein de groupe, cependant Discord est un très bon outil permettant de s'adapter simplement aux disponibilités de chacun.

2 Nature.

2.1 Nature du projet.

OCR - Sudo-code est une **interface de reconnaissance optique de caractères et de résolution de sudoku** programmée en langage C. Le projet contient également une partie de prétraitement d'image. À partir d'une photo de sudoku, le rendu final doit pouvoir extraire la grille, la résoudre, ainsi que la reconstruire de manière automatique, ou plus ou moins manuelle. La reconnaissance des chiffres doit se faire grâce à un réseau de neuronnes capable de s'entraîner sur un ensemble fini d'exemples. Tout de même, le projet contient des contraintes spécifiques à respecter. Dans un premier temps, il y a toute une série de contraintes d'écritures, qui permettent la création d'un code compréhensible et plaisant visuellement. Il y a ensuite des contraintes d'organisation entre autres constituées des règles sur le dépôt git ainsi que les dates de rendus. Enfin, le projet doit se limiter à l'utilisation de librairies extérieures C spécifiques : SDL et GTK.

3 Exécution du programme principal (*main*).

3.1 Makefile.

Pour chaque partie du projet, un fichier *Makefile* est présent afin de générer les exécutables des différents programmes de manière individuelle. Cela permet notamment un meilleur débogage, ainsi que la possibilité de travailler à plusieurs de manière organisée.

Par ailleurs, les fichiers Makefile des programmes générant des images possèdent tous l'instruction *clean_images* permettant de ne supprimer que les images. L'instruction *clean* quant à elle, supprime tous les fichiers générés (images, résultats, et exécutables). L'instruction *all* est également présente.

```
$ make program
$ make clean_images
$ make clean
$ make all
$ make clean
```

FIGURE 1 : fonctionnement de la commande make.

3.2 Main.

Un fichier *main.c* est présent dans la racine du projet afin de pouvoir générer toutes les étapes principales de celui-ci. Plusieurs options sont possibles :

```
$ make all
$ ./main "img" gray                    # Mise en nuance de gris.
$ ./main "img" gauss                   # Flou gaussien.
$ ./main "img" gamma                   # Augmentation du gamma.
$ ./main "img" bin                     # Binarisation.
$ ./main "img" rotation angle          # Rotation avec angle.
$ ./main "img" auto-rotation           # Rotation automatique.
$ ./main "img" all-no-rotate           # Tout sauf rotation.
$ ./main "img" all-man-rotate angle    # Tout, rotation manuelle.
$ ./main "img" all-auto-rotate         # Tout, rotation automatique.
$ ./main "img" grid_extract _          # Extraction de la grille.*
$ make clean
```

FIGURE 2 : fonctionnement de la commande ./main.

* : *L'extraction de la grille ne peut se faire que si l'image est au stade final du prétraitement.*

4 Tâches.

4.1 Chargement et sauvegarde des images.

Avant même de commencer à modifier les images, il est nécessaire de les charger ainsi que de pouvoir sauvegarder les modifications de ces dernières. Étant donné que le chargement et la sauvegarde d'image constituent une sous-étape qui sera présente dans une majorité des autres tâches, il était obligatoire de les séparer afin de pouvoir permettre une utilisation des fonctions dédiées partout dans le programme. C'est notamment la raison pour laquelle nous avons décidé de créer un dossier *utils* dans lequel on trouve le fichier *auxiliary.c* contenant une vingtaine de fonctions auxiliaires, utilisables dans tous les fichiers du projet par l'intermédiaire de la directive d'inclusion :

```
#include "../utils/auxiliary.h"
```

FIGURE 3 : directive d'inclusion d'auxiliary.h.

Deux fonctions permettant pour la première de charger une image sous forme de surface SDL, et pour la deuxième de sauvegarder une surface en .jpeg grâce à une fonction présente nativement dans SDL ont donc été définies.

Pour le bien du rapport, une image sera utilisée afin de mettre en avant les différentes étapes du traitement de l'image ainsi que de la détection des grilles. Sachant qu'il est nécessaire de montrer également la rotation, l'image choisie pour l'exemple sera l'image numéro cinq.

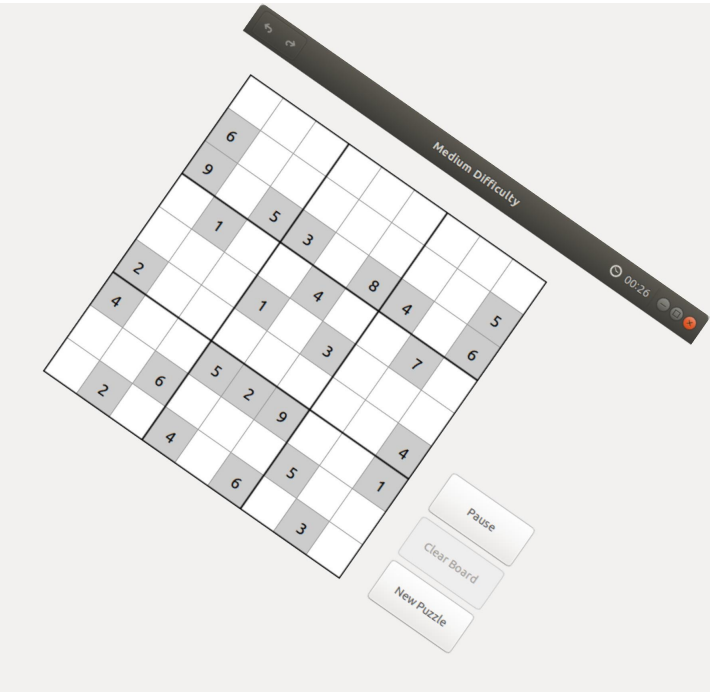

FIGURE 4 : image numéro 5 - après chargement (original).

4.2 Mise en nuance de gris (grayscale).

La deuxième étape correspond à la première étape du prétraitement. Cette dernière est la mise en nuance de gris de l'image. En effet, les couleurs ne sont pas nécessaires pour la reconnaissance des caractères, il est donc beaucoup plus utile de transformer l'image en noir et blanc afin de pouvoir traiter les pixels en fonction de leur couleur. Cependant, pour atteindre un noir et blanc complet, il faut d'abord passer par des nuances de gris. Pour convertir une image infographique couleur en niveau de gris il faut remplacer, pour chaque pixel les trois valeurs représentant les niveaux de rouge, de vert et de bleu, par une seule valeur représentant la luminosité. Pour cela, l'utilisation d'une formule mathématique calculant la moyenne des composantes rouges, vertes et bleues de chaque pixel et nécessaire. Celle-ci fut définie de la manière suivante :

$$gray = 0.4 * red + 0.35 * green + 0.25 * blue$$

FIGURE 5 : formule de calcul du niveau de gris.

Cette transformation permet alors, lorsqu'appliquée sur tous les pixels de l'image, d'obtenir la version visuellement nivelée en gris suivante :

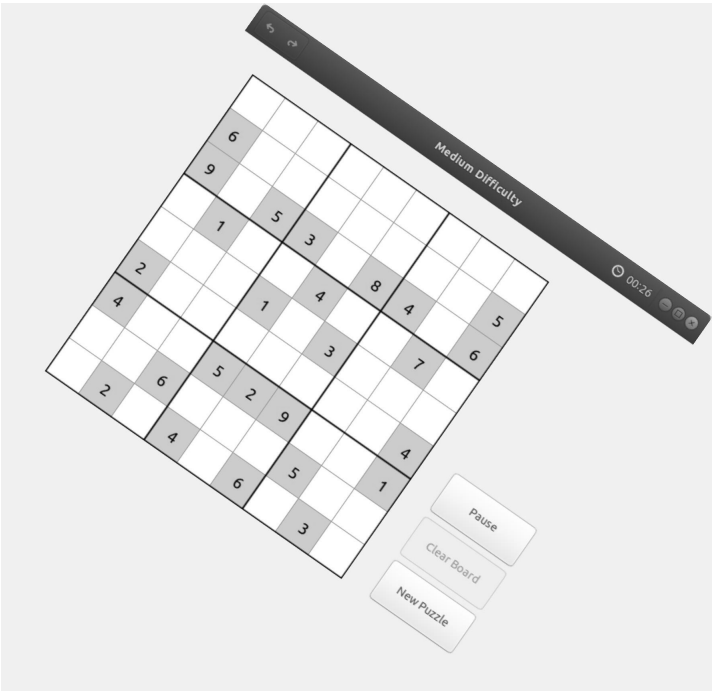

FIGURE 6 : image numéro 5 en niveau de gris.

L'image est ensuite sauvegardée sous le nom de "gray_image.jpeg". C'est l'image qui sera utilisée pour la prochaine étape. Une nouvelle sauvegarde se fait d'ailleurs pour chaque étape du prétraitement afin d'examiner les résultats en cas d'erreur. Par ailleurs, à chaque étape, c'est le résultat de l'étape précédente qui est utilisé, il est donc possible de préciser la nouvelle adresse à modifier de manière arbitraire. Afin de simplifier les différentes transitions, l'image est tout le temps sauvegardée dans le dossier racine. Enfin, le message suivant apparaît à la fin de la mise en nuance de gris :

Grayscale: **done**!

FIGURE 7 : message affiché après la mise en nuance de gris.

4.3 Réduction des bruits (flou gaussien).

Afin de simplifier la reconnaissance des pixels pour la mise en noir et blanc de l'image, une réduction des bruits des pixels est nécessaire. On appelle bruit numérique toute fluctuation parasite ou dégradation que subit l'image dès l'instant de son acquisition jusqu'à son enregistrement. En l'occurrence, la qualité des images contenant les grilles de sudoku n'est pas connue en avance, il est donc nécessaire de faire le mieux possible pour réduire les bruits, dans le cas où il y en aurait. Pour cela, le flou gaussien constitue une solution fonctionnelle et simple à implémenter à l'aide des fonctions déjà présentes dans les fichiers du projet. Il utilise notamment une moyenne qui est calculée par rapport aux pixels alentours de celui qui est analysé. Dès lors, le pixel concerné peut voir ses valeurs RGB modifiées de la manière suivante :

$$(R, G, B) = (\frac{\sum_{n=0}^{n} r}{n}, \frac{\sum_{n=0}^{n} g}{n}, \frac{\sum_{n=0}^{n} b}{n})$$

FIGURE 8 : formule de calcul du flou gaussien.

Avec n correspondant au nombre de pixel choisi dans les paramètres de la fonction, le rayon du carré autour du pixel concerné. En l'occurrence, n = 1. Pour rappel, cette étape est effectuée après la mise en niveau de gris. Cette transformation permet alors, lorsqu'appliquée sur tous les pixels de l'image, d'obtenir la version gaussienne de cette dernière :

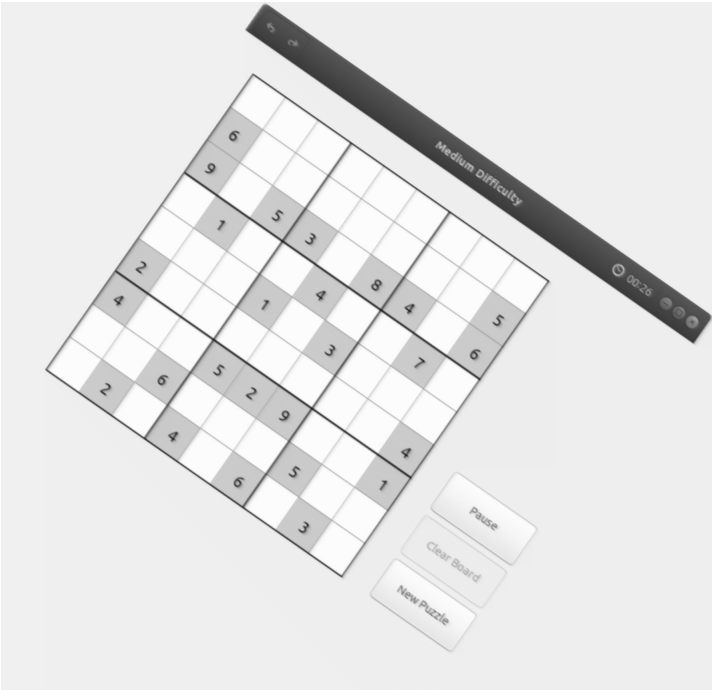

FIGURE 9 : image numéro 5 avec flou gaussien.

Enfin, le message suivant apparaît à la fin du flou gaussien :

Gaussian : **done**!

FIGURE 10 : message affiché après le flou gaussien.

4.4 Augmentation de l'éclairage contrasté (gamma).

Pour certaines images, notamment les images prises avec un téléphone, il est fréquent d'observer la présence de l'ombre du photographe. Celle-ci assombrit certains pixels, et il est donc nécessaire d'intégrer une fonction permettant la correction de la luminosité de ces pixels. Une augmentation du gamma de l'image est donc obligatoire. En photographie, le gamma, ou facteur de contraste, caractérise le contraste de l'émulsion. Plus sa valeur absolue est élevée, plus l'émulsion est contrastée et inversement. Une émulsion négative a un gamma positif puisqu'elle s'opacifie avec la lumière. À l'inverse une émulsion inversible (diapositive) a un gamma négatif. L'augmentation du gamma se fait par l'intermédiaire d'une fonction g(x) définie en l'occurrence par :

$$g(x) = 255 * (\frac{x}{255})^n$$

FIGURE 11 : formule de l'augmentation du gamma.

Avec :
— x : valeur de la couleur du pixel en 32 bits non signés.
— n : coefficient du gamma, en l'occurrence 0,623.
Cette étape est effectuée après le flou gaussien. La transformation permet alors, lorsqu'appliquée sur tous les pixels de l'image, d'obtenir la version contrastée de cette dernière :

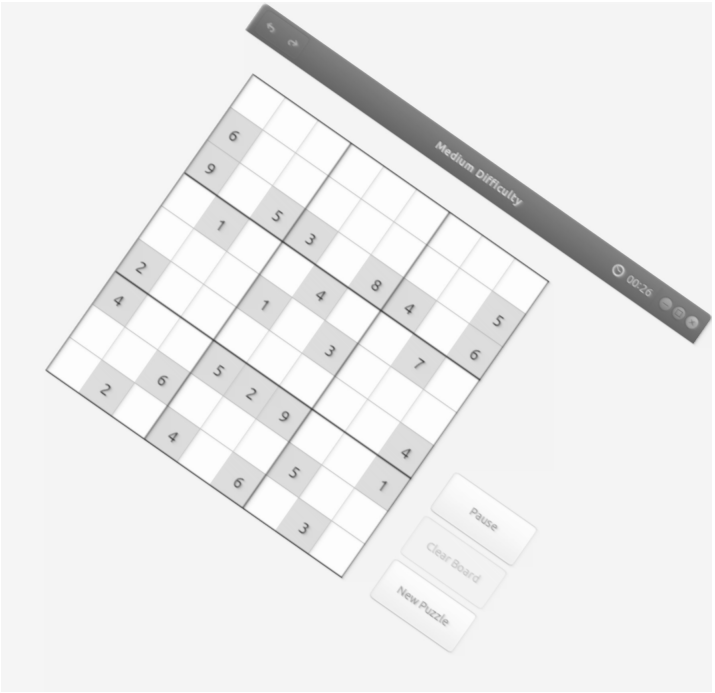

FIGURE 12 : image numéro 5 avec augmentation du gamma.

Enfin, le message suivant apparaît à la fin de l'augmentation du contraste :

Gamma: **done**!

FIGURE 13 : message affiché après l'augmentation du contraste.

4.5 Binarisation de l'image (mise en noir et blanc).

Le principe de binarisation consiste à passer d'une image grayscale (c'est à dire avec plusieurs niveaux de gris différents) à une image aux couleurs totalement binaire c'est à dire dans laquelle chaque pixel est soit noir soit blanc. Pour faire cela, le principe est identique à l'étape de grayscale qui crée une surface (à partir d'une image colorée) avec différents tons de gris. Dans un premier temps nous calculons la valeur du pixel traité en fonctions de ces coefficients RGB associés, puis nous vérifions si le résultat obtenu est supérieur à un certain seuil. Si oui, alors le pixel passe au noir, sinon, le pixel sera blanc.

La valeur de seuil n'est pas fixe. Elle dépend de l'image traitée. Le calcul consiste à prendre les valeurs des pixels voisins à notre pixel traités afin de calculer la valeur de t tel que la variance reste identique le plus souvent, puis nous renvoyons cette valeur comme étant le seuil. Deux méthodes nous permettent de calculer cela :

$$\sigma^2(t) = \omega_{bg}(t)\sigma^2_{bg}(t) + \omega_{fg}(t)\sigma^2_{fg}(t)$$

FIGURE 14 : formule probabilistique de la variance.

$$\sigma^2(t) = \frac{\sum(x_i - \overline{x})^2}{N-1}$$

FIGURE 15 : formule par somme de la variance.

Il est ensuite possible de calculer la variance pour différentes valeurs de t. La variance sera alors seuillée étant donnée que pour un certain t ¿ threshold, celle-ci ne changera plus. Le t deviendra alors le seuil à utiliser dans le cadre de la binarisation. Cependant, il est important de noter que la binarisation d'Otsu ne fonctionne pas avec toutes les images. Nous avons donc pris le choix d'ajouter une méthode de binarisation par moyenne, qui n'a malheuresement pas pu être implémentée pour la première soutenance, mais qui le sera pour la deuxième soutenance. L'image suivante est alors obtenue :

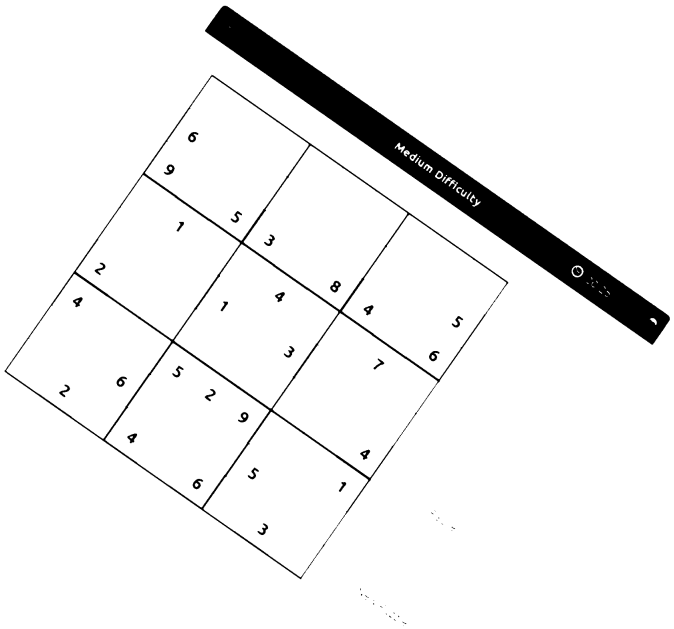

FIGURE 16 : image numéro 5 binarisée.

Enfin, le message suivant apparaît à la fin de l'augmentation du contraste :

```
Binarization : done!
```

FIGURE 17 : message affiché après binarisation.

4.6 Rotation.

4.6.1 Rotation manuelle.

Dans le cas où l'image aurait subie une rotation avant d'être insérée en paramètre, il est nécessaire de pouvoir la faire tourner d'un certain angle à l'aide d'une fonction de rotation centrale. La partie rotation manuelle s'occupe donc d'effectuer cette opération géométrique de manière arbitraire, avec un angle qui doit être entré lors de l'appel de la fonction. Il est ensuite possible de calculer les nouvelles dimensions de l'image, étant donné qu'elles risquent d'être modifiées lors du processus, on a donc :

$$l_d = ceil(l_s * |cos(\theta)| + h_s * |sin(\theta)|)$$
$$h_d = ceil(l_s * |sin(\theta)| + h_s * |cos(\theta)|)$$

FIGURE 18 : formules de calcul des nouvelles dimensions de l'image.

Avec :
— l : longueur de l'image (d pour destination et s pour source).
— h : largeur (hauteur) de l'image (d pour destination et s pour source).
— θ : angle de rotation en radian (une fonction de conversion est présente dans le fichier des fonctions auxiliaires).
— ceil : arrondi à l'entier supérieur.

On peut ensuite calculer pour chaque pixel p(x, y) son équivalent r(x, y) correspondant à ses coordonnées dans la matrice de rotation avec les formules suivantes :

$$r_x = ceil(cos(\theta) * (x - m_{xd}) + sin(\theta) * (y - m_{yd}) + m_x)$$
$$r_y = ceil(-sin(\theta) * (x - m_{xd}) + cos(\theta) * (y - m_{yd}) + m_y)$$

FIGURE 19 : formules de calcul des nouvelles coordonnées d'un pixel p(x, y).

Avec :
— r : nouvelles positions (x et y).
— m : milieu (x pour la longueur, y pour la hauteur, d si destination).
— θ : angle de rotation en radian (une fonction de conversion est présente dans le fichier des fonctions auxiliaires).
— ceil : arrondi à l'entier supérieur.

Cette opération se répète ensuite pour chacun des pixels de l'image, mais n'est exécutée que si les valeurs ne dépassent pas la matrice de rotation. Cette vérification est faite à l'aide de la condition suivante :

```
    if (0 <= rotate_x
&& rotate_x < src_width
&& 0 <= rotate_y
&& rotate_y < src_height)
```

FIGURE 20 : condition de vérification de la matrice de rotation.

4.6.2 Rotation automatique.

Il y a deux étapes majeures pour la rotation automatique : dans un premier temps, il faut calculer l'angle de rotation. Puis, dans un second temps, il faut effectuer la rotation avec l'angle trouvé. L'étape la plus complexe est sans aucun doute la détection de l'angle de rotation. Pour cela, une transformée de Hough a été effectuée. Celle-ci se décrit par l'algorithme suivant :

```
accumulateur = tableau la taille de l'image (2D).
pour chaque x dans longueur:
pour chaque y dans largueur:
si p(x, y) constitue une bordure (couleur):
    pour theta allant de 0 -> 180:
        rho = ceil(x * cos(theta) + y * sin(theta))
        accumulateur[theta][rho]++
```

FIGURE 21 : création de l'accumulateur de Hough.

Il ne suffit alors que de calculer l'index de l'angle le plus présent dans l'accumulateur, et d'effectuer une rotation de l'image par un angle $\theta_r = 360+90-\theta = 450-\theta$ qui correspond à l'angle opposé par trigonométrie. En appliquant cette rotation sur notre image, le résultat suivant est alors obtenu :

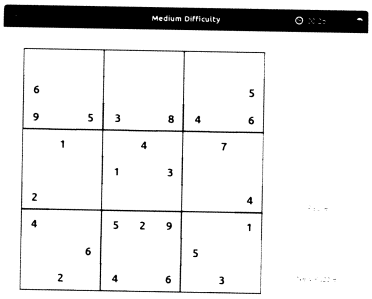

FIGURE 22 : image numéro 5 après rotation automatique.

Le message suivant apparaît au début de la rotation automatique :

```
Auto-Rotation: starting rotation with angle.
```

FIGURE 23 : message affiché au début de la rotation automatique.

4.7 Détection et extraction de la grille.

Tout le processus de prétraitement terminé, il faut ensuite extraire la grille de sudoku de l'image. C'est-à-dire créer une nouvelle image ne contenant que cette dernière, afin par la suite d'obtenir une grille pouvant être traitée de manière efficace. Il existe une multitude de manières d'effectuer cette extraction. En l'occurrence ici, il s'agit d'une extraction arbitraire : la détection commence par la recherche des abscisses des quatres plus grandes lignes verticales de l'image. Le minimum de ces abscisse correspond conséquemment à la ligne la plus à gauche, qui devrait correspondre à une des extrémités du sudoku. Il faut ensuite retrouver les valeurs du haut et du bas de la ligne, afin de pouvoir en calculer la hauteur, pour, ensuite, extraire le carré sachant que la grille de sudoku est un carré. On peut donc retrouver les étapes suivantes :

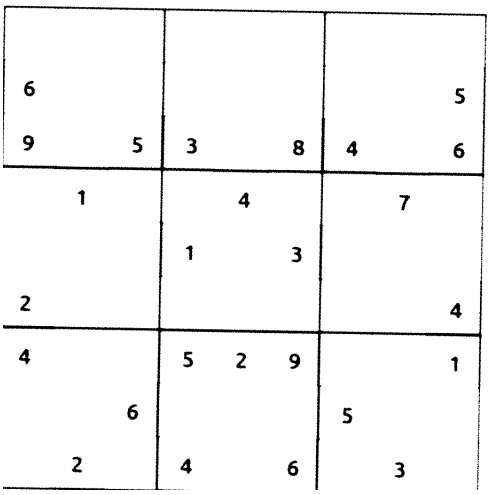

FIGURE 24 : grille extraite.

Il est ensuite possible d'extraire chacune des rangées de la grille par une simple division par 9.

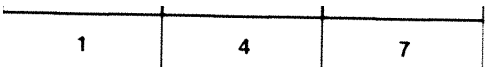

FIGURE 25 : rangée extraite.

Cependant, cette méthode possède quelques points faibles, et elle est notamment mise à défaut dans le cas ou il y a une plus grande ligne verticale à gauche qui ne constitue pas un côté de la grille de sudoku. C'est la raison pour laquelle un système de détection de grille par recherche de la plus grande composante connexe sera effectué afin de pouvoir prendre en compte tous les différents cas. Toutes les rangées sont sauvegardées sous le format "RANG.jpeg"

Les messages de l'extraction de la grille sont d'ailleurs les suivants :

```
Grid Detection: sudoku edges found at: 1211  915  619  1519.
Grid Detection: found probable edge at x =  619
Grid Row Extraction: height found.
Grid Row Extraction: interval set to: 102
```

FIGURE 26 : messages affichés lors de l'extraction de la grille.

4.8 Détection et extraction des chiffres.

Après avoir extrait les rangées, il est très facile d'obtenir les chiffres individuels. Il suffit uniquement de passer à travers chacune d'entre-elles et à découper avec un interval égal au neuvième de la longueur de l'image recadrée (étant donné qu'il n'y a que 9 chiffres maximum par rangée).

FIGURE 27 : chiffre extrait.

Tous les chiffres sont stockés dans le dossier *numbers/* par ordre d'apparition, afin ensuite de pouvoir les manipuler par *Raw Major Order*. Le dossier ressemble alors à ceci :

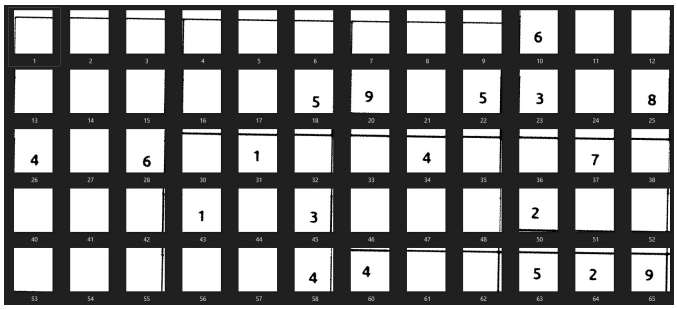

FIGURE 28 : dossier contenant les chiffres extraits.

4.9 Réseau de neuronnes.

L'objectif pour cette première soutenance était de réaliser un réseau de neurones simple, capable d'apprendre la fonction XOR. C'est à dire :

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 29 : Table XOR.

Comme nous pouvons le voir, pour avoir un réseau de neurones capable d'apprendre la fonction XOR, il nous faut deux neurones en Input constitutant notre *Input Layer* ainsi qu'un neurone de sortie (Output) consituant notre *Output Layer*.

En ce qui concerne notre *Hidden Layer*, le nombre de neurones est plus arbitraire. Nous avons trouvé cela judicieux d'en choisir deux, étant donné que le problème présenté est assez simple, et surtout pour ne pas surchager notre réseau de long calcul (pas forcément

pertinent dans notre cas de figure) qui le ferait perdre en efficacité.
Notre réseau ressemble donc au schéma suivant :

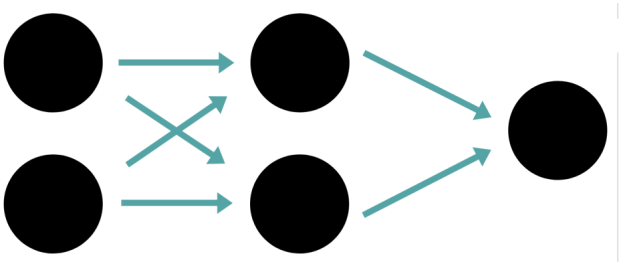

FIGURE 30 : Schéma du réseau de neurones utilisé.

Comment marche un réseau de neurones ?

Le neurone calcule la somme de ses entrées x pondérées par les poids w, avant d'appliquer la fonction d'activation ϕ . Le résultat de cette somme sera ensuite passée à une fonction d'activation qui définira le résultat en Output obtenu. Autrement dit :

$$a_j = \phi(b_j + \sum(w_{ij}x_i))$$

Maintenant que nous avons expliqué le principe, nous allons nous attarder sur l'implémentation de notre réseau.

La première étape est d'initialiser nos poids et nos biais de manière aléatoire (avec des valeurs comprises entre 0 et 1). Nos deux neurones constituant notre *Inputs Layers*, se verront attribuer un poids reliant cette couche à l'*Hidden Layer*. Ce dernier, se verra également attribuer des poids le reliant à l' *Outputs Layers*. Et pour finir, notre neurone en sortie se verra attribuer un biais.

La deuxième étape est celle de l'entraînement.

Pour cette étape, nous faisons le calcul décrit précédement, auquel nous appliquons la fonction d'activation Sigmoïde parfaite qui est de la forme suivante :

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

Cette fonction d'activation permet de voir l'évolution de notre réseau de neurones avec des résultats de plus en plus proches des valeurs attendues (contrairement à la fonction Marche Heaviside qui renvoie un résultat binaire, 0 ou 1).

Une fois avoir appliqué la fonction, nous pouvons effectuer la dernière étape, celle de **Backpropagation**.

Cette étape consiste à rééquilibrer les poids et les biais en fonction du taux d'erreur. Ce taux d'erreur se calcule en faisant la différence entre le résultat attendu et le résultat en Output. Cette différence sera multipliée par l'application de la dérivée sur chaque valeur des neurones constituant l'*Hidden Layer*.

Pour rappel, la dérivée de la fonction Sigmoïde s'écrit sous la forme :

$$f'(x) = x * (1 - x)$$

Pour cette étape, notre réseau effectue 4 cycles de 50 000 entraînements. Ces chiffres sont totalement arbraitres mais nous permettent d'avoir des résultats très satisfaisants. Cependant, ils seront amenés à être modifiés au cours de ce projet afin de réduire le temps de la

phase d'entraînement de notre réseau.

Les résultats obtenus pour 4 cycles de 10 000 entraînements sont déjà suffisamment intéressants pour notre projet.

4.10 Chargement de la grille de sudoku.

Préalablement, un programme se chargeant de reconstruire la grille à partir des images de chachun des chiffres de la grille (images vides comprises) devra être implémenté et appelé afin de créer un fichier texte contenant la représentation de grille à resoudre.

```
$ ls
grid_00 solver
$ cat grid_00
... ..4 58.
... 721 ..3
4.3 ... ...

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

3.6 ... ...
... 158 ..6
... ..6 95.
```

FIGURE 31 : Fichier texte contenant la grille à résoudre.

Le principe est simple, il s'agit d'une lecture de fichier. La méthode sera appliquée identiquement à chaque fichier étant donné que ceux-ci sont censés avoir une mise en forme similaire. La fonction d'import va lire un caractère à la fois dans le fichier et stocker la valeur numérique de celui-ci dans un tableau 2D quand cela est possible, c'est-à-dire lorsqu'elle lit un chiffre. Notons que les '.' correspondent à des cases vides (étiquetés 0 dans le tableau) et les sauts à la ligne témoignent d'un changement de ligne dans la grille du sudoku. Les lignes vides et les espaces entre les cases ne sont, quant à eux, présents que par pur souci de lecture et de visibilité pour le programmeur. Ainsi à la fin, la fonction assigne chaque ligne du fichier à un tableau de dimension 1 dans le tableau final du sudoku.

4.11 Résolution de la grille de sudoku.

Pour la résolution du sudoku, nous avons décidé d'opter pour la méthode du ***backtracking***. En plus d'être récursive, elle se veut complète et garantit un résultat unique pour chaque grille.

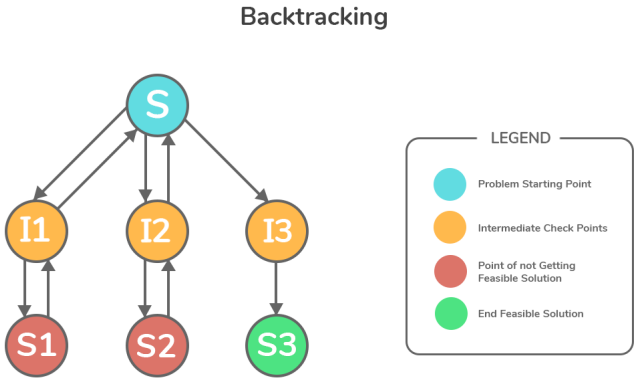

FIGURE 32 : Principe du backtracking.

Le programme solver prend en entrée un fichier de la forme de la FIGURE 31, importe ce fichier dans un tableau 2D (fonction d'import) puis se charge de résoudre ce tableau. S'il existe une solution alors la grille initiale sera modifiée sinon, grâce au backtracking le programme retournera à la case de coordonées (0,0) en ayant remis toutes les cases modifiées à 0 et le tableau demeura donc inchangé par rapport à celui du début.

```
$ ./solver grid_file
```

FIGURE 33 : Appel au programme solver.

Lorsque la résolution est possible et positive, le programme se charge d'exporter le tableau final sous forme de fichier texte. Le fichier sera alors sous la forme : ***grid_file.result*** où ***grid_file*** est le fichier passé en entrée. Il aura la même mise en forme que le fichier d'import. À l'inverse, si la grille est inrésolvable, aucun fichier .result ne sera crée et le programme renverra une erreur.

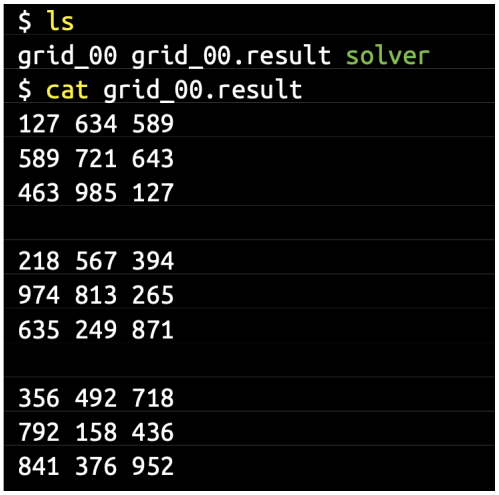

FIGURE 34 : Fichier contenant la grille résolue (cf grille fig.31).

La résolution fonctionne donc parfaitement, que la grille soit résolvable ou non le programme renverra toujours une réponse. Un affichage sous forme d'image pour la grille résolue est prévu pour la seconde soutenance. Il prendra en compte la grille initiale et affichera en couleurs les chiffres placés pendant la résolution. Cette image sera créee et sauvagardée dans le répértoire courant afin de, plus tard, la visualiser sur l'interface graphique.

5 Conclusion.

Pour le moment, nous avons donc un main ainsi qu'un Makefile permettant d'effectuer les opérations de chargement et de sauvegarde des images tests, leur mise en nuance de gris, ainsi que leur prétraitement, en d'autres termes, le redressement automatique et manuel de l'image, l'élimination des bruits parasites par la méthode gaussienne ainsi que le renforcement des contrastes. De plus, nous avons également mis en application les processus de détection de la grille et des cases. Enfin, l'algorithme de résolution de la grille et la sauvegarde de celle-ci, résolue, sont des étapes que l'on peut considérer accomplies.

Les réalisations futures considérées pour le bon déroulement du projet seront la binarisation et l'extraction de la grille plus optimales. Finalement, les tâches auxquels nous nous attèlerons sont un prétraitement ainsi qu'un réseau de neurones complets et fonctionnels, comportant l'apprentissage et la reconnaissance des chiffres de la grille. Il est donc nécessaire de finaliser la reconstruction de la grille, l'affichage de celle-ci et puis la sauvegarde du résultat obtenu sous forme d'image. Ces opérations devront être accessibles depuis une interface graphique permettant d'utiliser tous ces éléments.