OCR Sudoku Solver

Projet S3

Rapport Soutenance 1

Vianney Marticou, Romain Vallette-Grisel,
Baptiste Cormorant, Ugo Majer

Professeur référent : Patricia Lay

Novembre 2022

Table des matières

1 Introduction

Au cours du troisième semestre de scolarité au sein de l'ÉPITA, chaque étudiant doit réaliser un projet avec 3 autres étudiants. Nous quatre, Vianney Marticou, Romain Vallette-Grisel, Baptiste Cormorant et Ugo Majer, nous sommes réunis en ce début de semestre dans l'optique de former un groupe de travail.

Contrairement au projet du deuxième semestre, où le choix du projet était relativement libre, cette fois-ci le sujet nous impose de réaliser un programme de type OCR (optical character recognition en anglais et reconnaissance optique de caractères en français) en langage C en suivant un cahier des charges bien défini.

L'objectif final est que ce programme, muni d'une interface graphique, prenne en entrée une image d'une grille de Sudoku et qu'il se charge de faire tous les traitements numériques de l'image nécessaire pour ensuite tenter de reconnaître les différents chiffres présents dans la grille de Sudoku par l'intermédiaire d'un réseau de neurones. Ainsi, la grille sera restituée proprement et sera résolue par un algorithme de résolution par backtracking.

L'objectif de ce document est de présenter les différentes tâches réalisées, jusqu'à l'heure de la première soutenance. Nous évoquerons donc les principales étapes du traitement numérique de l'image, la conception d'un réseau de neurones simulant un XOR ainsi que le programme de résolution de la grille.

Notre rapport de première soutenance aura la structure suivante :

1. Répartition des tâches entre les différents membres du groupe ;
2. Conception des fichiers Makefile (ainsi que la compilation et l'architecture) ;
3. Conception de structures dans le but d'avoir un code propre et maintenable ;
4. Pré-traitement et traitement numérique de l'image ;
5. Conception du réseau de neurones ;
6. Algorithme de résolution du Sudoku par backtracking.

1.1 Vianney Marticou

Vianney Marticou, étudiant à ÉPITA et chef de ce projet. Issu d'une Première S (Sciences de l'ingénieur) ainsi que d'une Terminale Mathématiques/Numérique et Sciences de l'informatique, je pense pouvoir dire que je me passionne pour l'informatique. Il faut dire que pour satisfaire ma curiosité, je me passionne d'énormément de sujets, de la psychologie à l'impression 3D, du cinéma à la montagne. Je suis heureux et fier d'avoir été nommé chef de ce projet, il ne faut pas oublier que notre travail n'est pas que logiciel, il est aussi humain.

Je suis un grand admirateur de Linux. Je passe une bonne partie de mon temps libre à développer de nouvelles options sur mon OS. Un des mes plus grands rêves serait de m'investir dans le développement open-source d'une distribution, de préférence Arch. Du fait de mes compétences techniques en informatique, je suis aussi le dépanneur de mon entourage ce qui m'a oblige à avoir des compétences variées en électronique. J'espère apprendre une grande quantité de choses dans ce projet et plus particulièrement sur le fonctionnement de réseaux de neurones.

1.2 Romain Vallette-Grisel

Romain Vallette-Grisel, actuellement en InfoSpé à l'ÉPITA sur le campus de Lyon et issu d'un baccalauréat général spécialités Mathématiques/Physique-Chimie, j'aime l'univers des sciences et de l'informatique. Depuis très jeune, j'ai toujours été curieux de comprendre le fonctionnement d'un ordinateur et des différents systèmes d'information. À mes heures perdues au collège et au lycée, j'essayai de réaliser des plugins Minecraft. C'est ainsi que j'ai appris les concepts de base de l'algorithmique (conditions, boucles, etc..) et aussi la programmation orientée objet présente en Java.

C'est un honneur d'être étudiant à l'ÉPITA et de pouvoir participer à des projets de groupe permettant d'allier les connaissances théoriques et pratiques de programmation, ainsi que les 'soft skills' dans un projet qui est très formateur. En effet, jusqu'à présent j'avais déjà réalisé de la programmation haut niveau, le langage C et le bas niveau en général sont une nouveauté pour moi, tout comme le traitement d'images et les réseaux de neurones. Ainsi, je ne doute pas que ce projet est et sera plus formateur que celui réalisé au semestre précédent. De plus, les compétences mathématiques seront mises à rude épreuve, et cela est très challengeant.

1.3 Baptiste Cormorant

Baptiste Cormorant, passionné d'histoire, de sciences, mais surtout d'informatique, j'ai créé de nombreux projets qui m'ont permis d'apprendre de nombreuses technologies. La création de site Internet et d'applications m'ont permis d'apprendre les technologies Web. J'ai aussi créé de petits jeux en Python à titre récréatif, mais qui ont servi à m'en faire apprendre beaucoup. J'ai depuis peu commencé la création d'application mobile, donc apprentissage de technologies comme le React et le Java. J'ai ainsi appris à m'organiser et à respecter les délais imposés. J'ai aussi appris des compétences réseaux qui sont très utiles pour de nombreux projets.

Ce projet me permettra d'en apprendre beaucoup sur des thèmes que je n'avais pas encore abordé comme le traitement d'image ou les réseaux de neurones. Il va aussi me permettre de travailler en équipe et de gérer ses aléas. Le fait de savoir travailler en équipe est une compétence importante. Cet apprentissage continu permet de passer des caps et de pouvoir aller toujours plus loin.

1.4 Ugo Majer

Ugo Majer, étudiant à l'ÉPITA Lyon en deuxième année de cycle préparatoire. Je suis passionné d'informatique depuis ma quatrième (2015), où j'ai découvert d'abord l'informatique à travers de la programmation sur Scratch, grâce à des cours optionnels instaurés à mon collège situé à Montluçon en Allier. À partir de la seconde, j'ai commencé à apprendre le Python puis j'ai logiquement choisi pour mon bac les spécialités Mathématiques et NSI. Depuis ma troisième, j'ai commencé à apprendre en autodidacte sur OpenClassrooms, où j'ai pu commencer à comprendre le réseau et découvrir d'autres langages de programmation comme le C.

J'espère apprendre beaucoup avec ce projet, surtout que les thèmes abordés sont passionnants et très vastes. J'avais déjà fait du traitement d'images en terminale pour pouvoir faire un scanner de QR code. De plus, l'IA, comme le traitement d'images, sont des sujets au coeur de notre société actuellement donc c'est important de comprendre comment ça marche.

2 Répartition des tâches

Tâches	Vianney	Romain	Baptiste	Ugo
Chargement de l'image	X	X		
Redimensionnement de l'image	X	X		
Suppression des couleurs	X	X		
Pré-traitement de l'image	X	X		
Rotation de l'image			X	X
Détection de la grille			X	X
Découpage des cases			X	X
Rotation de l'image			X	X
Sauvegarde Poids Réseau Neurones	X	X		
Chargement Poids Réseau Neurones	X	X		
Algorithme Résolution Sudoku	X	X	X	X
Interface			X	X

FIGURE 1 – Répartition des tâches prévue au début du projet

Tâches	Vianney	Romain	Baptiste	Ugo
Conception des fichiers Makefile	X			
Conception de structures	X	X		
Chargement de l'image		X		
Redimensionnement de l'image		X		
Mise en nuance de gris	X	X		
Mise en noir et blanc		X		
Pré-traitement de l'image		X		
Détection de la grille			X	X
Découpage des cases			X	X
Rotation manuelle de l'image		X		
Réseau de neurones XOR	X			
Sauvegarde Poids Réseau Neurones	X			
Chargement Poids Réseau Neurones	X			
Jeu d'images pour l'apprentissage	X			
Algorithme Résolution Sudoku	X	X	X	X
Gestion des fichiers Sudoku Solver		X		

FIGURE 2 – Tâches réalisées pour la première soutenance

3 Makefile (compilation & architecture)

Nous avons choisi de faire un Makefile légèrement over-engineering dans l'optique de mieux comprendre cet outil, de mieux structurer notre code et de pouvoir le re-utiliser dans d'autres projets.

À la racine du projet, nous avons plusieurs dossiers :

1. **build** : Il contient tous les fichiers nécessaires à la compilation du projet. Le fait d'avoir un dossier qui contient le tout permet de ne pas avoir des .o qui traînent dans le dossier src. Ce dossier n'est jamais envoyé sur le répertoire distant (grâce au .gitignore).
2. **include** : Il contient l'intégralité des headers du projet. La séparation des headers et du code source apporte de la clarté dans l'architecture du projet.
3. **src** : Ce dossier contient le code source de l'OCR. Chaque fonctionnalité possède son propre dossier ce qui améliore aussi la clarté de l'architecture.
4. **sub-project** : Ce dossier est un peu particulier, il contient les fonctionnalités demandées par le projet mais pouvant être présentes de manière externe tel que le programme solver et le réseau de neurones XOR.

Un des principaux problèmes que nous avons rencontré pendant la création de ce Makefile est l'aspect récursif du répertoire. Nous ne pouvions pas utiliser la fonction wildcard pour avoir la liste des fichiers ayant l'extension .c. Nous avons dû utiliser une fonction du bash find. Cette dernière nous renvoie la liste des fichiers avec leur adresse relative, sous la forme src/MaFeature/monfichier.c. Une fois le fichier récupéré, il faut le compiler sous la forme d'un .o et le mettre dans le dossier build/. Pour cela, nous récupérons le nom du fichier, et remplaçons la chaîne src/ par build/ et l'extension par .o grâce à la fonction patsubst (une fonction native de Make).

Bien entendu, nous avons une variable CFLAGS qui contient les arguments que nous donnons à notre compilation. Cela nous permet de rajouter des librairies ou des arguments de debugging (tel que -fsanitize=address). Nous avons aussi créé une variable LFLAGS qui contient les arguments du linker, qui rassemble les .o du dossier build/. L'exécutable créé par le linker se met à la racine et il suffit d'exécuter la commande ./main.out (avec éventuellement des arguments) pour le lancer.

Un des intérêts de notre architecture est la règle clean de notre Makefile. Il suffit de supprimer le dossier build/ et le projet est propre. Dans les options du Makefile, nous avons implémenté le compile qui recompile l'intégralité du projet, le build qui ne recompile pas mais exécute le linker et la règle run qui compile, build et exécute le programme. Par défaut, le Makefile compile et build mais n'exécute pas le programme.

4 Structures

S'organiser c'est bien, avoir un code organisé l'est tout autant ! Les structures sont justement là pour ça. Cependant, qu'est-ce qu'une structure ? Une structure est un regroupement de plusieurs variables, de types différents ou non. Grosso modo, une structure est finalement une boîte qui regroupe plein de données différentes.

Nous avons pris le choix de nous documenter sur les structures, que nous n'avons pas vu jusqu'à présent en TP, car nous souhaitions réaliser des objets comme nous l'avions vu en C#. Cependant, le langage C n'est pas un langage orientée objet. Pourtant, nous avons tout de même découvert les structures qui permettent certaines choses que nous retrouvons dans les langages orientées objet.

Ainsi, nous avons fait le choix de concevoir les structures suivantes :

1. Matrix (représentant des matrices) ;
2. Image (représentant comme son nom l'indique une image) ;
3. Filter (représentant un filtre que l'on peut appliquer à une image) ;
4. NeuralNetwork (représentant un réseau de neurones que l'on peut entraîner).

4.1 Matrix

Dans le cadre de notre implémentation du réseau de neurones et des filtres, nous avons dû créer une structure Matrix. Elle contient 3 variables : la largeur (width),la hauteur (height) et un pointeur donc un tableau qui ne contient que des pointeurs, vers d'autres tableaux. Chaque tableau contient des éléments sous forme de nombre à virgule. Dans un premier temps, nous étions partis dans l'idée d'avoir des tableaux de nombres entiers, mais au fil des recherches nous avons découvert que les filtres et le réseau de neurones était représentés par des nombres à virgule.

Pour pouvoir utiliser cette structure, nous avons créé un certain nombre de fonctions élémentaires. On peut citer par exemple l'addition et la multiplication de deux matrices mais aussi l'addition et la multiplication par un scalaire. La fonction applyFunctionMatrix nous permet d'appliquer à chaque élément de la matrice une fonction bien définie, elle nous sera extrêmement utile lors de l'entraînement du réseau de neurones.

4.2 Image

La structure Image, comme son nom l'indique regroupe tous les éléments caractéristiques d'une image. Ainsi, elle possède un entier représentant la largeur, un autre représentant la hauteur. Enfin, elle contient aussi un tableau à 2 dimensions de Pixels. Pour rappel, un pixel est associée à une couleur, usuellement décomposée en trois composantes primaires par synthèse additive : le rouge, le vert et le bleu.

Dans le but de pouvoir utiliser simplement cette structure et y créer des éléments de celle-ci, nous avons développer plusieurs fonctions donc deux principales :

— **Image *importImage(char *filename) ;**
Elle permet, à partir d'un chemin, absolu ou relatif, vers une image numérique de l'importer dans notre programme et de créer un élément de type Image contenant les informations de cette image numérique. Pour lire l'image, nous avons utiliser SDL (Simple DirectMedia Layer).

— **void saveImage(Image *image, char *filename) ;**
Elle permet, à partir d'un pointeur vers une Image et d'un chemin de destination, d'enregistrer l'image vers le chemin de destination spécifié. Nous utilisons aussi SDL pour cette fonction.

Ces deux fonctions sont les seules à utiliser SDL dans notre projet, en effet, ne pas utiliser SDL était inconcevable pour ce projet. Cependant, nous souhaitons simplifier au maximum les choses, et c'est l'une des principales raisons qui nous a poussé à créer cette structure.

4.3 Filter

La structure Filter, comme son nom l'indique regroupe tous les éléments caractéristiques d'un filtre. Ainsi, elle possède un entier représentant le rayon du filtre, un autre représentant la largeur du filtre. Enfin, elle contient aussi un pointeur vers un élément de la structure Matrix. En effet, un filtre est avant tout une matrice !

Dans le but de pouvoir utiliser simplement cette structure et y créer des éléments de celle-ci, nous avons développer plusieurs fonctions donc deux principales :

— **Filter *createFilter(int radius, float (*func) (int, int)) ;**
Elle permet, à partir d'un rayon et d'une fonction, de créer le filtre associé.

— **Image *applyFilter(Filter *filter, Image *image) ;**
Elle permet, à partir d'un pointeur vers un filtre et d'un pointeur vers une image, de créer une nouvelle image étant le résultat du filtre sur l'image.

4.4 NeuralNetwork

Cette structure n'est pas particulièrement nécessaire. Nous l'avons afin de clarifier le code de notre fonction train. En effet comme dis depuis le début, nous avons pour but de faire un programme qui marche mais aussi un code lisible et maintenable.
Pour le réseau de neurones cela se passe par une structure qui contient :

- Un matrice 'output' qui représente les bits d'output du réseau de neurones
- Une matrice 'hidden' qui représente tous les neurones des couches internes.
- Une matrice de biais 'hiddenBiais' qui stocke les biais de chaque couche des neurones d'hidden.
- Une matrice de biais 'outputBiais' qui stocke les biais de chaque couche des neurones d'output.

L'un des intérêts d'avoir créer une structure qui contient notre réseau de neurones est le fait que l'on peut l'adapter de manière extrêmement simple. Par exemple, nous pouvons passer d'un XOR à un OCR qui reconnaît des symboles tels que des chiffres.
L'idée de stocker nos neurones dans une matrice ne vient pas de nous. Nous avons appris cette technique lors de la lecture de papier de recherche. Ce stockage facilite grandement le travail de forward propagation. Nous détaillerons plus amplement le sujet dans le chapitre suivant.

Pour la bonne utilisation de cette structure, nous avons du implémenter certaines fonctions :

- **NeuralNetwork* initNetwork(int numInput, int numHidden, int numOuput) ;**
  Cette fonction permet de générer un réseau de neurones, il contient le bon nombre de neurones ainsi que le bon nombre de layers. Le tout étant bien entendu stocké dans des matrices.
- **void printNeural(NeuralNetwork* net) ;**
  Cette fonction peut sembler bien innocent mais il n'en est rien. Il est important lors du débugage de son code d'avoir des print clair et précis.
- **void saveNeural(NeuralNetwork* net) ;**
  Cette fonction n'est pas encore implémenter dans notre code mais elle est prévu dans notre projet. Elle nous permettra de sauvegarder nos neurones, une fois entraîné.

5 Pré-traitement et traitement de l'image

La phase de pré-traitement et de traitement de l'image est une phase cruciale de ce projet. Car nous pouvons avoir un réseau de neurones fonctionnel, avec de bons résultats, cependant si l'image en entrée du réseau n'est pas de bonnes qualités, contient du bruit ou des pixels parasites, alors le résultat en sortie du réseau de neurones ne sera pas à la hauteur.

La phase de pré-traitement consiste à :

— Redimensionner l'image pour améliorer le temps d'exécution des algorithmes ;

— Convertir, dans un premier, chaque pixel en une nuance de gris ;

— Appliquer un flou gaussien pour réduire le bruit présent sur l'image d'origine ;

— Convertir, dans un second temps, chaque pixel soit en noir, soit en blanc.

La phase de pré-traitement est délicate car chaque image à ses propres caractéristiques. Par exemple, certaines sont très ombrées, tandis que d'autres peuvent être très clairs. Ou encore, certaines peuvent contenir beaucoup de bruits, tandis que d'autres peuvent en contenir très peu. Ainsi, il est difficile de trouver le pré-traitement le plus optimal possible sur le plus grand nombre d'images numériques.

Pour cela, nous avons essayer : un algorithme de réajustement des gammas, un algorithme de réajustement des constrastes, un algorithme de dilatation, un algorithme d'érosion. Cependant, l'utilisation de ces algorithmes et leurs combinaisons ne nous ont pas permis d'obtenir de meilleurs résultats finaux, au contraire la qualité de l'image numérique avait tendance à se détériorer.

La phase de traitement consiste à :

— Détecter la grille de Sudoku ;

— Effectuer une rotation de l'image, en cas de besoin ;

— Découper chaque case du Sudoku.

Pour la première soutenance, la rotation de l'image se fera de façon manuelle, c'est-à-dire que l'angle de rotation sera spécifié par l'utilisateur. Cependant, l'objectif pour la soutenance finale est que le programme soit en mesure de calculer l'angle de rotation optimal, ainsi l'utilisateur n'aura plus besoin de fournir l'angle de rotation manuellement au programme.

5.1 Redimensionnement de l'image

Notre programme doit être en capacité de prendre n'importe quelle grille de Sudoku en entrée, et ce, peu importe la dimension de l'image. Cependant, la taille de l'image ne doit pas avoir de conséquence drastique sur le temps d'exécution de notre programme. Ainsi, pour éviter que les grandes images, qui auraient donc beaucoup de pixels prennent trop de temps à être traiter par nos algorithmes, nous avons fait le choix de les redimensionner. Ainsi, nous réduisons le nombre de pixels présent sur ces images et gagnons du temps d'exécution pour nos algorithmes nécessitant un parcours complet de chaque pixel de l'image.

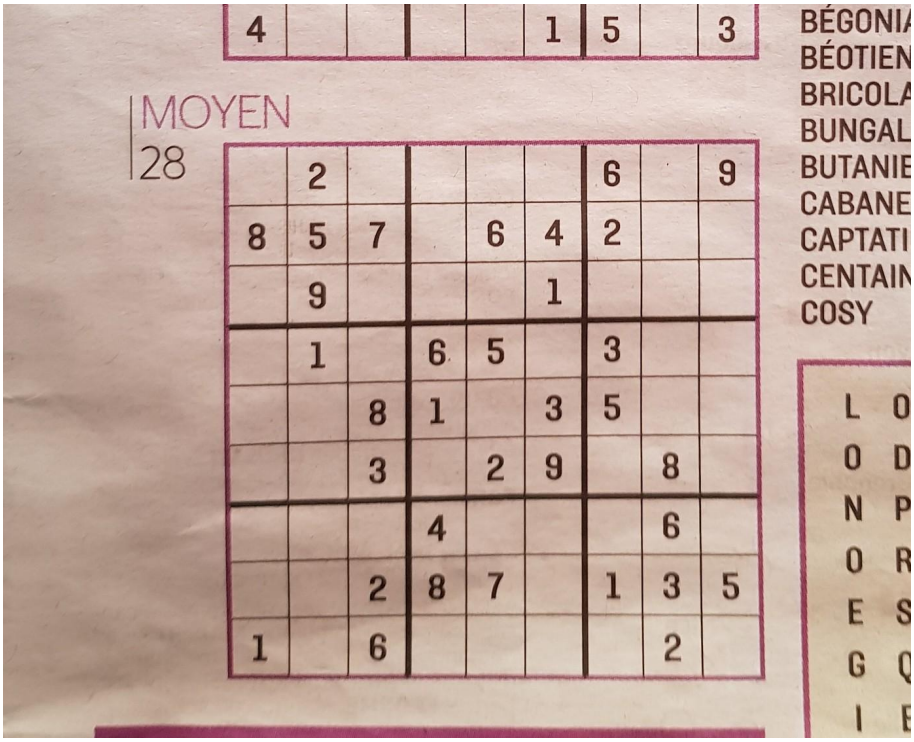

FIGURE 3 – Grille de Sudoku après un redimensionnement

5.2 Conversion en niveau de gris

Les couleurs n'ayant pas d'importance sur la grille de Sudoku, nous avons pris le choix de la convertir en nuances de gris. Pour se faire, nous calculons pour chaque pixel de l'image la nuance de gris correspondant au mieux :

$$gris = 0.3 * rouge + 0.59 * vert + 0.11 * bleu$$

Puis ensuite, nous remplaçons chaque composante du pixel par la valeur du gris calculée. Par exemple, si la valeur de gris calculée est 142. Alors, nous attribuerons la valeur 142 aux composantes rouge, vert et bleu du pixel. Ainsi, les trois composantes du pixel ont la même valeur, et cela représente visuellement un gris (plus ou moins foncé en fonction de la valeur du gris calculée).

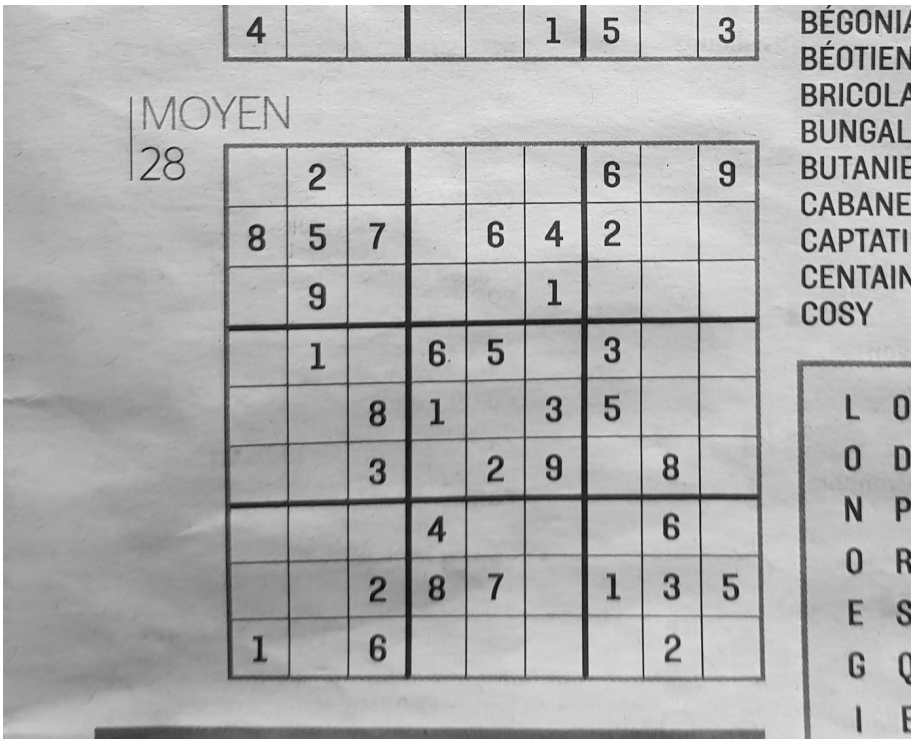

FIGURE 4 – Grille de Sudoku après un passage en niveau de gris

5.3 Filtre de flou (flou de Gauss)

La prochaine étape du processus de pré-traitement est d'appliquer un flou gaussien sur l'ensemble de l'image numérique. Nous utilisons ce filtre pour atténuer le bruit contenu dans l'image. La processus de ce filtre est simple. Pour chaque pixel de l'image, on calcule une nouvelle valeur qui est égale à la somme du produit des pixels voisins et d'une matrice gaussienne. Cette matrice est générée à l'aide d'une fonction gaussienne, qui est de la forme suivante :

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

FIGURE 5 – Grille de Sudoku après l'application d'un flou de Gauss

5.4 Binarisation de l'image (méthode d'Otsu)

La binarisation est l'opération qui permet de convertir une image en une image ayant uniquement deux couleurs distinctes (généralement le noir et le blanc). Pour pouvoir binariser l'image que l'utilisateur va fournir à notre programme, nous avons implémenter la méthode d'Otsu. Cette méthode nécessite d'abord de générer un histogramme représentant le nombre de pixels en fonction de la nuance de gris.

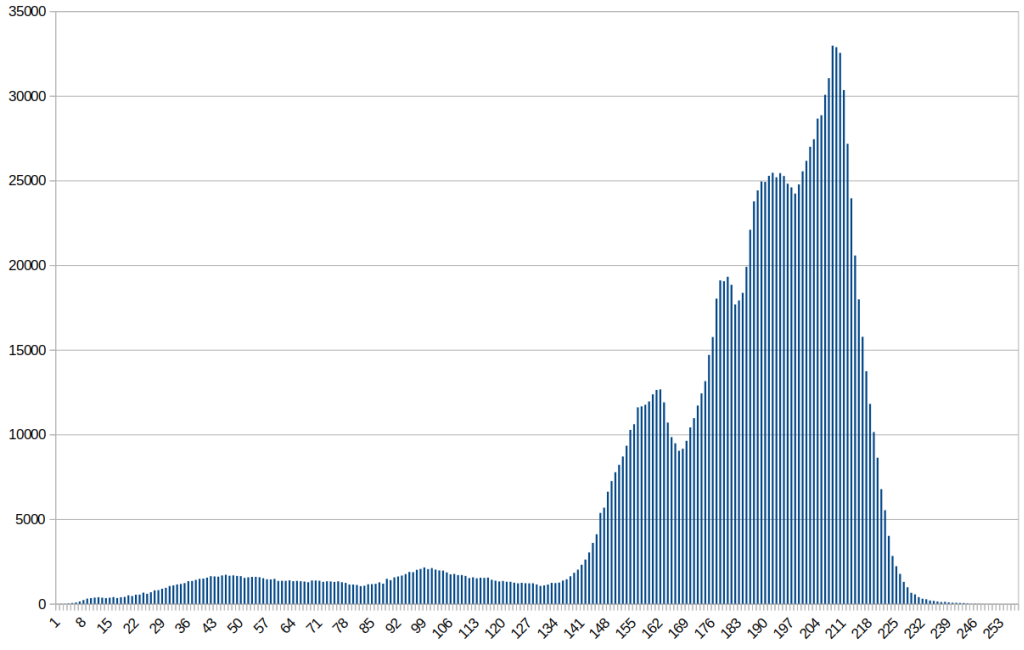

FIGURE 6 – Histogramme de la grille présente à la figure 4

La méthode d'Otsu suppose alors que l'image à binariser ne contienne que deux classes de pixels, (c'est-à-dire le premier plan et l'arrière-plan) puis calcule le seuil optimal qui sépare ces deux classes afin que leur variance intra-classe soit minimale.

Otsu montre que minimiser la variance intra-classe revient à maximiser la variance inter-classe. Pour les plus curieux et les plus matheux, voici la formule de la variance inter-classe :

$$\sigma^2 = W_b W_f (\mu_b - \mu_f)^2$$

W_b (respectivement W_f) correspond au nombre de pixels en arrière-plan (respectivement au premier plan) divisé par le nombre total de pixels. μ_b (respectivement μ_f) correspond à l'intensité moyenne de l'arrière-plan (respectivement du premier plan).

Une fois le seuil optimal trouvé, il suffit de mettre à 0 (représentant le noir) les pixels ayant une valeur inférieure au seuil optimal, et les autres à 255 (représentant le blanc). Ainsi, notre image est devenue binaire, elle ne contient que des pixels blancs et des pixels noirs. Ci-dessous, un aperçu d'une image binarisée :

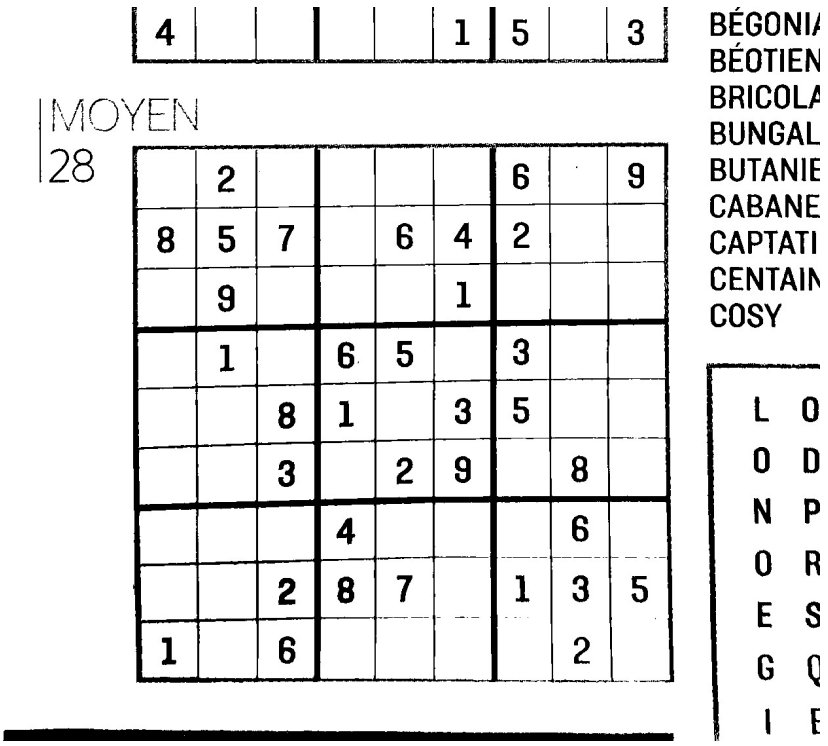

FIGURE 7 – Grille de Sudoku après binarisation

Bien que le rendu semble très qualitatif, ce n'est pas toujours le cas en fonction de la photo d'origine. En effet, si l'image d'origine contient des zones ombrées, le résultat avec la méthode d'Otsu n'est pas celui souhaité. Pour cela, nous devons essayer de trouver un moyen d'enlever ces zones d'ombres pendant la phase de prétraitement. Hélas, ne n'avons pas trouvé de moyens efficaces d'améliorer nos résultats. Nous avons essayé un ajustement des gammas, des contrastes, une dilatation, une érosion mais rien n'a été concluant.

5.5 Détection de la grille (transformée de Hough)

5.5.1 Transformée de Hough

La transformée de Hough consiste à représenter chaque point de contour détecté dans un espace de paramètres à deux dimensions, par exemple une droite est caractérisée par deux paramètres, donc dans cette espace c'est un point.

La transformée de Hough d'un point de l'image analysée est la courbe de l'espace des paramètres correspondant à l'ensemble des droites passant par ce point. Si des points sont colinéaires, alors toutes les courbes de l'espace de paramètres se coupent au point représentant la droite en question. Du fait des imperfections de l'image, les points détectés ne sont pas parfaitement alignés et donc les courbes ne sont pas parfaitement concourantes. La méthode consiste donc à discrétiser l'espace de paramètres, à le découper en petits rectangles, et à dénombrer pour chaque rectangle le nombre de courbe y passant. On construit ainsi une matrice dite d'accumulation, les maxima locaux de cette matrice correspondant à des droites probables.

5.5.2 L'algorithme

On va commencer par parcourir l'image. Puis pour chaque point de contour que l'on détecte, on détermine la courbe correspondante. Ensuite, on construit la matrice d'accumulation à partir des courbes déterminées. On termine en détectant les pics dans la matrice créée.

On utilise les paramètres ρ et θ où ρ est la distance de la droite du repère et θ est l'angle que fait la perpendiculaire à la droite avec l'axe x.

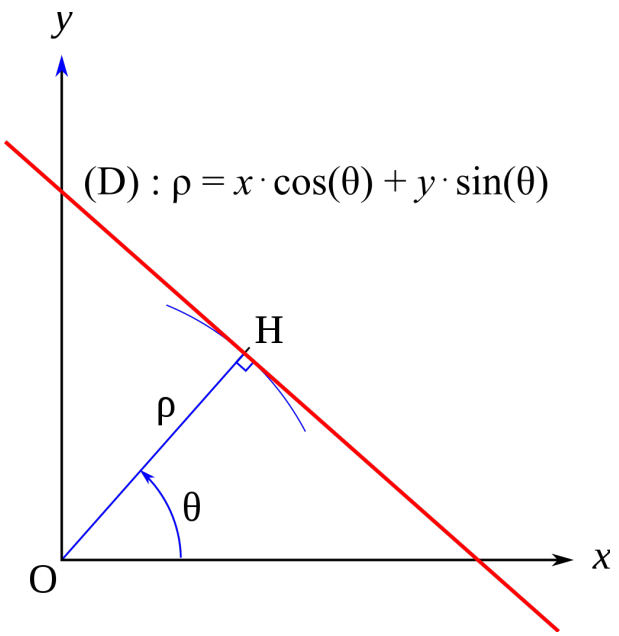

FIGURE 8 – Figure d'explication pour l'algorithme de Hough

5.6 Rotation de l'image

Avant de passer à la dernière étape du traitement, à savoir le découpage de la grille, nous devons effectuer une rotation de l'image. En effet, si l'utilisateur a fourni au programme une image qui est de travers, nous devons effectuer une rotation pour permettre ensuite à l'algorithme de découpage de correctement découper la grille.

Pour la première soutenance, l'angle de rotation devra être fourni par l'utilisateur dans le cas où une rotation est nécessaire. Cependant, l'objectif pour la soutenance finale est que le programme soit en mesure de savoir par lui-même si une rotation est nécessaire, et si c'est le cas, de calculer l'angle pour ensuite appliquer la rotation.

Pour effectuer une rotation d'un angle θ, nous appliquons une matrice de rotation à chaque pixel $\begin{pmatrix} x \\ y \end{pmatrix}$ de notre image numérique :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} * \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} = \begin{pmatrix} xcos(\theta) + ysin(\theta) \\ -xsin(\theta) + ycos(\theta) \end{pmatrix}$$

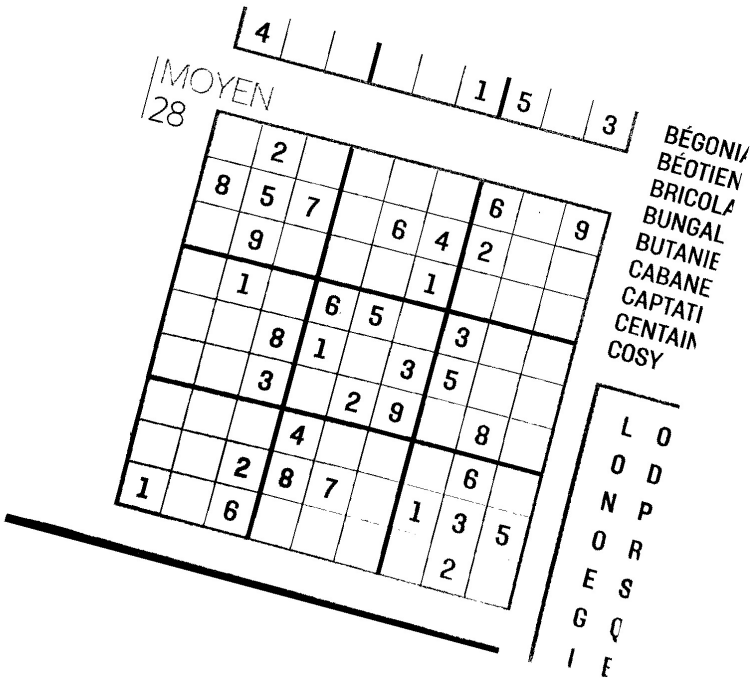

FIGURE 9 – Grille de Sudoku après une rotation de 15 degrés

L'image présente à la figure 4 n'avait besoin que d'une très légère rotation. Cependant, pour montrer que notre algorithme de rotation fonctionne bien, nous avons volontairement forcé le trait pour notre exemple présent à la figure 9.

6 Réseau de neurones

6.1 Un neurone

Un neurone reçoit plusieurs entrées et fournit une sortie, grâce à différentes caractéristiques :

— Des **poids** accordés à chacune des entrées, permettant de modifier l'importance de certaines par rapport aux autres. On les note :

$$w_1, w_2, ..., w_n = \left[w_1, w_2, ..., w_n\right]$$

.

— Une **fonction d'agrégation** (ou fonction de combinaison), qui permet de calculer une unique valeur à partir des entrées et des poids correspondants. On l'écrit mathématiquement de cette façon :

$$\sum_{i=1}^{n} x_i w_i$$

.

— Un **seuil** (ou biais) est un offset que l'on ajoute avant de calculer la fonction d'activation. On le note b.

— Une **fonction d'activation**, qui associe à chaque valeur agrégée une unique valeur de sortie dépendant du seuil. On la symbolise la plupart du temps par la fonction f.

Maintenant que nous avons vus les caractéristiques d'un neurone, nous sommes en capacité de l'exprimer d'un point de vue mathématique :

$$y = f(b + \sum_{i=1}^{n} x_i w_i)$$

6.1.1 Perceptron

Dans la partie précédente, nous avons vu le fonctionnement interne d'un neurone. Maintenant, nous allons nous intéresser aux réseaux de neurones, plus particulièrement celui que l'on appelle le perceptron. C'est le réseau de neurones le plus simple que l'on puisse faire. En effet, il ne contient que un seul neurone et une seule entrée.Son utilisation est assez restreinte car un neurone seul n'est pas extrêmement puissant mais il permet déjà faire une séparer deux éléments distincts. Effectivement si l'on regarde la définition du perceptron.

$$y = f(b + x_1 w_1) = f(ax + b)$$

Dans le cadre du perceptron, la fonction d'activation permet de savoir si le point se trouve au dessus ou bien en dessous de la ligne.

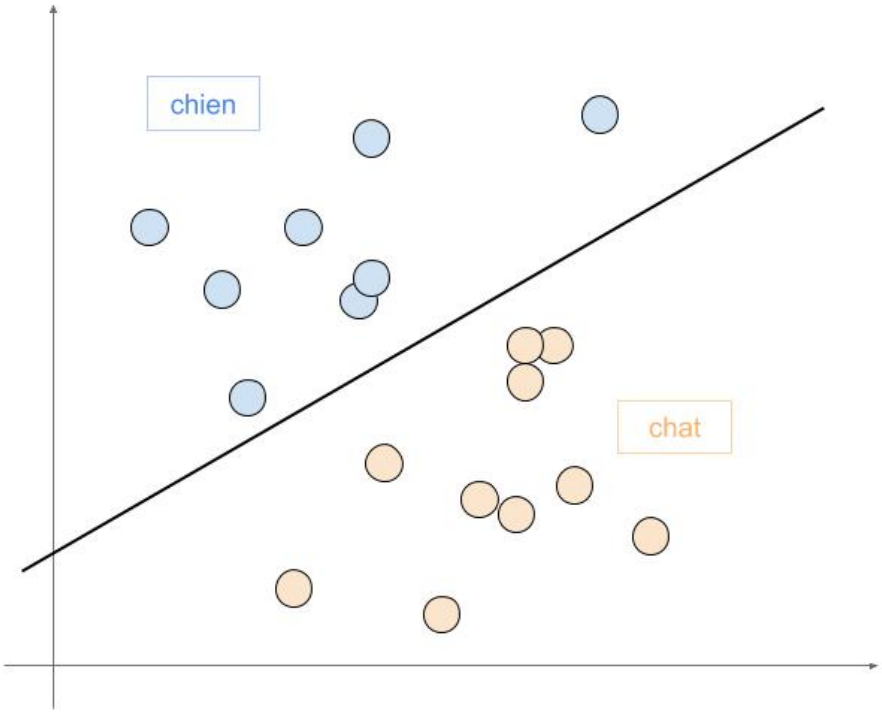

FIGURE 10 – Représentation graphique de la classification faîte par un perceptron

6.2 Forward Propagation

On utilise ce terme de forward propagation (ou propagation avant) lorsque les signaux ne peuvent se propager que dans un seul sens, d'une couche d'entrée vers une couche de sortie, sans possibilité de retour en arrière Ils s'opposent aux réseaux de neurones récurrents dans lesquels, les signaux peuvent se propager en avant et en arrière.

C'est une étape primordiale de l'entraînement du réseaux. En effet, les signaux calculer lors de la propagation vont nous être utile lors de la backward propagation.

6.3 Backward Propagation

La backward propagation (ou rétropropagation d gradient) est une méthode pour entraîner un réseau de neurones, consistant à mettre à jour les poids de chaque neurone de la dernière couche vers la première. Elle vise à corriger les erreurs selon l'importance de la contribution de chaque élément à celles-ci. Dans le cas des réseaux de neurones, les poids synaptiques qui contribuent plus à une erreur seront modifiés de manière plus importante que les poids qui provoquent une erreur moindre.
Pour cela, nous devons la calculer l'erreur, voici la fonction mathématique.

$$e^{\text{sortie}} = f'(g^{\text{sortie}})(y - t)$$

En sachant, que l'on note y la sortie obtenue par le réseau et t la sortie voulu. On execute ce calcul uniquement sur la derniere couche du reseaux, nous avons une autre fonction pour les couches internes et d'entrée.

$$e^{(n-1)} = f'^{(n-1)}(g^{(n-1)}) \sum_i w_i^{(n)} e_i^{(n)}$$

On note g exposant (n) la fonction d'agrégation que l'on applique sur l'intégralité de la couche n.

6.3.1 Sur-entrainement

Lorsque l'on cherche a entraîner un réseaux de neurones, il peut arrive que le modèle reconnaisse très bien les images d'entraînement et moins bien celles de validation. Le modèle aura une faible capacité prédictive, il n'arrive pas à généraliser. On parle alors de sur-entraînement. Dans ce cas là on peut ajouter davantage d'images pour pallier ce problème ou il faut utiliser un modèle moins complexe. Un autre facteur sur lequel nous pouvons jouer est le nombre de fois que l'on entraîne notre modèle.

6.4 Dataset

Le dataset est une partie importante de l'entraînement quasiment plus importante que le modèle en lui même. En effet, un réseaux de neurones n'est rien sans le base de données qui l'entraîne. L'ideale etant d'avoir une base de données couvrant le maximum de cas possible. Elle doit aussi prendre en compte les cas qui n'ont qu'une probabilité faible. Il faut savoir que le dataset se découpe le plus souvent en deux parties.

6.4.1 Données d'entraînement

Cette base de donnés est utilisé par le modèle lors de l'entraînement. Pour pouvoir, utiliser ces données, il faut que l'on connaisse la valeur souhaité. On utilise la plupart du temps, un fichier au format CSV ou JSON pour stocker le nom de l'image ainsi que la solution souhaité. Cette même solution qui sera utilisé par notre algorithme de backward propagation.

6.4.2 Données de validation

Cette base de données est utilisé afin de savoir si notre modèle est fiable. Elle permet aussi de dire si nous avons sur-entraîné notre réseau. Le but étant d'avoir un échantillon le plus large possible afin de voir comment réagi notre Intelligence Articielle (IA). Une fois l'étape de la validation passé, nous pouvons sauvegarder les poids synaptiques du modèle et les recharger à volonté.

7 Algorithme de résolution par backtracking

Le cahier des charges du projet nous demande d'implémenter un algorithme de résolution de Sudoku en langage C. Pour tester cet algorithme, nous devions aussi concevoir un petit programme appelé solver, qui s'exécute en ligne de commande uniquement. Il prend en entrée un nom de fichier dans un format spécifique, résout le Sudoku correspondant, et génère un fichier contenant le résultat. Le format du fichier de sortie est le même que celui du fichier d'entrée. De plus, le nom du fichier de sortie est le nom du fichier d'entrée avec l'ajout de l'extension «.result».

D'abord, nous devons lire le contenu du fichier spécifié en paramètres de notre programme. Dans le cas où le nombre d'arguments spécifiées est égale à 1, le fichier spécifié existe et qu'il respecte le bon format alors il va être lu et les données vont être stockées dans un tableau à 2 dimensions représentant ainsi la grille de Sudoku.

Maintenant que le fichier est lu et chargé en mémoire par l'intermédiaire d'un tableau à 2 dimensions, nous devons nous assurer que la grille fournie respecte les règles standards du Sudoku. Nous vérifions donc que dans chaque ligne, chaque colonne et chaque région, les chiffres de 1 à 9 apparaissent au maximum une fois.

Maintenant que nous nous sommes assurés que la grille de Sudoku respecte les règles standards du jeu, nous pouvons tenter de la résoudre par l'intermédiaire d'un algorithme de backtracking. Le backtracking, consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage. Concrètement, le backtracking peut s'apparenter à un parcours en profondeur d'un arbre avec une contrainte sur les noeuds : dés que la condition n'est plus remplie sur le noeud courant, on stoppe la descente sur ce noeud. Dans notre cas, les contraintes sont simplement les règles du Sudoku. Nous tenons à noter que notre algorithme de résolution est récursif.

Enfin, dans le cas où notre algorithme à réussi à résoudre la grille, nous enregistrons le résultat dans un nouveau fichier portant le même nom que le fichier d'origine avec l'ajout de l'extension «.result».

Vous trouverez ci-dessous un tableau représentant les différents messages d'erreurs et codes de retour en fonction de l'état du programme.

Etat	Message d'erreur	Code de retour
Nombre d'arguments incorrect	The number of arguments is not valid	1
Fichier non existant	The file specified in argument doesn't exist	2
Format fichier incorrect	The file doesn't respect the correct format	3
Sudoku représenté non valide	The Sudoku grid is not valid	4
Aucune solution trouvée	The Sudoku grid doesn't have solution	5
Sudoku résolu		0

FIGURE 11 – Messages d'erreur et codes de retour du solver

8 Conclusion

Pour conclure, on peut dire que même si nous avons eu de nombreux défis, notamment à cause de notre manque de connaissances, nous avons continué à apprendre de nous-même afin de dépasser les problématiques auxquelles nous étions exposés ce qui est particulièrement valorisant.

Dans le cadre de ce projet, nous avons du apprendre a nous renseigner par nous mêmes. Le sujet étant pointu, nous avons du lire de nombreux articles de recherches en anglais. Ce fut un expérience particulièrement enrichissante.

Sur le plan technique, nous avons commencé à voir émerger certaines fonctionnalités liée au traitement de l'image ce qui nous permis de découvrir de nombreux aspects, la transformée de Hough ainsi que les matrices de rotation. Cela rend le projet particulièrement stimulant et instructif. D'autant plus que le travail qui nous attend concernera les fonctionnalités les plus intéressantes à implémenter ce qui renforce l'aspect stimulant du projet.

Table des figures