Rapport de soutenance 1

Projet S3 - EPITA

OCR Sudoku Solver
par Optical Cramptés Recognition

November 2023

Sommaire

1 Reprise du cahier des charges

Pour cette première soutenance, nous avons décidé de nous répartir équitablement les tâches à effectuer. Au delà des attributions de tâches, c'est en travaillant en groupe et en s'entraidant que toutes les tâches ont pu être finalisées correctement, et que nous avons même, parfois, pu prendre de l'avance sur la deuxième soutenance. Il était avant tout nécessaire pour nous que toutes les fonctionnalités disponibles soient fonctionnelles, avant d'être optimisées. Néanmoins, nous avons parfois réussi à optimiser certaines tâches.

Ainsi, voilà comment les tâches ont été réparties :

	Lucas	**Nicolas**	**Arthur**	**Yan**
GUI		• Création d'une interface graphique (GTK) • Chargement, rotation et sauvegarde d'une image		
Image			• Transformation de Hough et diverses aides	• Début du prétraitement • Diverses fonctions triviales de manipulation de pixels
Détection de grille				• Transformée de Hough (détection et fusion des lignes de l'image) • Découpe de l'image
Résolution			• Solver pour le sudoku avec gestion de toutes les erreurs possibles	
IA	• Création d'un réseau de neurones • Implémentation du XOR et entrainement d'un modèle			

2 Lucas

Je m'occupe de la partie reconaissance des chiffres d'une grille de sudoku par l'intermédiaire d'un réseau de neurones. Pour cette première soutenance, j'ai créé un prototype de réseau neuronal permettant de calculer un XOR, une opération binaire qui fonctionne sur le même principe qu'un "différent" en maths.

2.1 Principe d'un réseau de neurones

2.1.1 La propagation avant

Un réseau de neuronnes n'est en réallité qu'un ensemble de fonctions mathématiques de la forme: $f(x) = w * x + b$, avec w le poids d'un noeud et b le biais assossié à la couche actuelle.

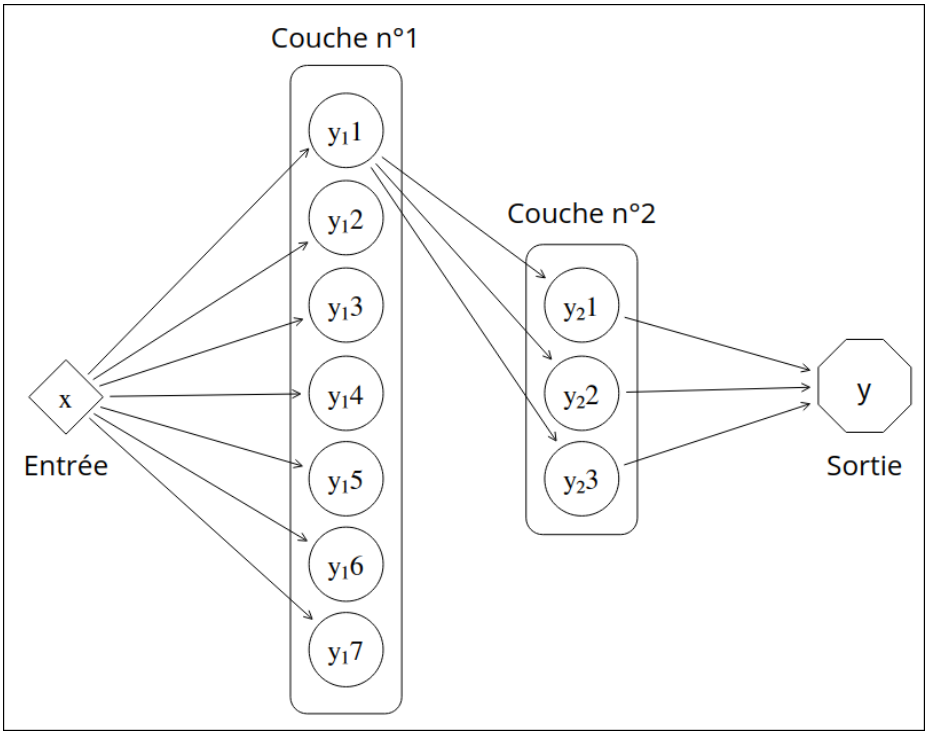

Figure 1: Representation d'un réseau de neurones

Comme nous pouvons le voir sur la figure 1, un réseau de neurones est constitué d'une couche "d'entrées" et d'une couche de "sorties", ces couches sont reliées par un ensemble de couches "cachées", ici numérotées 1 et 2. Pour pouvoir calculer la sortie y d'un réseau de neurones, on doit calculer la valeur de chacun de ses noeuds avec l'algorithme suivant : soit la valeur y_i^j associé au i-ème noeud de la j-ème couche, et n le nombre de noeud à la couche $j-1$, $y_j^i = \Sigma_{k=1}^{n} w_k^i * y_{j-1}^k + b_i$.

A chaque couche j de n noeuds est associé :

- Une liste de biais: b_1, b_2, ..., b_n, contenant un biais par noeud.
- Une matrice de poids: w, de dimension $[i, n]$, avec i le nombre de noeud à la couche $j-1$

Ainsi nous pouvons simplifier la formule de calcul des valeurs d'une couche en utiliseant des calculs matriciels. On obtient donc la formule suivante pour la matrice colone c des valeurs des noeuds de la j-ème couche : $c_j = w_j * c_{j-1} + b_j$.

La dernière chose à ajouter pour pouvoir calculer la/les sortie(s) d'un réseau de neurones c'est ce qu'on appelle la fonction d'activation. On peut trouver différentes fonctions d'activation sur internet et leur interet est de ramener la valeur de sortie d'un noeud dans un intervalle plus facile à manipuler, comme $[0, 1]$. La fonction que nous avons décidé d'utiliser pour le réseau de neurones de notre XOR s'appelle la fonction sigmoid : $sigmoid(x) = 1/(1 + exp(-x))$.

En appliquant cette formule de la première à la dernière couche, nous obtenons ainsi une sortie comprise dans l'intervalle $[0, 1]$. Ce calcul s'appelle la propagation avant (en anglais : forward propagation).

2.1.2 La propagation arrière

Maintenant que nous savons comment calculer la sortie d'un réseau de neuronne, il nous reste à déterminer les valeurs des poids et des biais qui permettent d'obtenir le meilleur résultat possible. Pour l'initialisation, il suffit de donner des valeurs aléatoires entre 0 et 1.

Pour pouvoir ajuster de la bonne manière les composantes de notre réseau de neurones, il nous faut des données sur les lesquelles entrainer notre modèle (le modèle étant l'ensemble des poids et des biais). Ces données sont sous la forme d'une liste de couples de deux valeurs, la valeur d'entrée et la valeur de sortie attendue. Pour chaque valeur d'entrée, nous alors alors calculer à l'aide de la propagation avant une valeur de sortie et la comparer à la valeur attendue.

Pour quantifier les erreurs effectuées par le modèle, nous allons utiliser une fonction coût (en anglais : cost function). La fonction coût que nous utiliserons pour notre modèle de XOR est la fonction Log Loss : $L = \frac{1}{m} * \Sigma_{i=1}^{m}(y_i * log(a_i) + (1 - y_i) * log(1 - a_i))$. Nons calculons ensuite la dérivée de cette fonction coût en fonction des valeurs w et b qui ont permit de donner sa valeur au noeud. Une dérivée négative implique que la valeur w_j^i/b_j doit augmenter pour faire diminuer la valeur de la fonction coût et donc diminuer l'écart entre la valeur obtenue et la valeur attendue, de même pour une valeur positive de la dérivée qui implique que la valeur de w_j^i/b_j doit diminuer.

En appliquant ce principe sur la couche de sortie de notre réseau de neurones, nous obtenons ainsi de nouvelles valeur pour notre matrice de poids et de biais de notre dernière couche. Par récursivité sur les couches précédentes, nous obtenons ainsi les nouvelles valeurs de w et b pour chacunes des couches de notre modèle.

On peut noter que plusieurs optimisations peuvent être effectuées sur un tel modèle :

- Le choix de la fonction coût
- Le choix de la fonction d'activation
- Le trie des données d'entrainement

2.2 L'implémentation en C

2.2.1 Les matrices

Pour implémenter un tel modèle en C, il faut dans un premier temps coder les outils nécessaires pour les formules mathématiques que nous devons utiliser pour faire les deux propagations de notre réseau de neuronnes, à savoir les matrices et leurs opérations : addition, multiplication, fonction, ... Le fichier *matrix.c* contient la structure d'une matrice en mémoire et toutes les opérations nécessaires.

2.2.2 Le réseau de neurones

Vient ensuite le réseau de neurones en lui même, avec sa structure en mémoire, son modèle et ses méthodes associés comme la propagation avant et arrière qui permettent respectivement de calculer la sortie du réseau en fonction de certaines entrées et d'ajuter le modèle pour se rapprocher d'une valeur souhaitée.
Le fichier qui contient notre réseau de neurones et le fichier *neural_network.c*.

2.2.3 Le XOR

Enfin pour pouvoir utiliser et simuler des entrainements du modèle de notre réseau de neurones, il nous faut un dernier fichier *xor.c* qui permet la création d'un réseau de neurones, son entrainement sur un nombre donné d'itération et d'afficher la précision du modèle à la fin de l'entrainement.

2.2.4 Les améliorations

Le désavantage majeur de cette technique est de devoir entrainer le modèle à chaque utilisation. Nous avons donc réfléchi à un système de sauvegarde du modèle dans un fichier ce qui permet d'entrainer le modèle sur plusieurs sessions et peut s'avérer très utile lors de l'entrainement du modèle final pour le sudoku. Le code actuel permet ainsi la création d'un réseau de neurones avec ou sans modèle prédéfini et la sauvegarde du modèle actuel dans un fichier à la fin de l'entrainement.

2.3 Ressentis

L'implémentation de l'IA capable de détecter le XOR a été une partie très intéressante à développer. Malgré la grande quantité de mathématiques et la difficulté relative pour trouver de la documentation et des informations, j'ai vraiment apprécié me pencher sur cette partie du projet.

3 Nicolas

Pour la première soutenance, la tâche de créer l'interface graphique m'a été assignée. Elle se devait d'être capable d'afficher un écran dans lequel on peut charger une image et effectuer une rotation.

3.1 Le logiciel Glade

Pour créer cette interface graphique, j'ai utilisé le logiciel Glade, comme vous pouvez le voir dans l'image ci-dessous. Glade est un outil puissant qui facilite la création de tous les éléments qui composent l'interface utilisateur, tels que les boutons, les images, les zones de texte, et bien d'autres éléments essentiels. Il permet de disposer visuellement ces composants et de personnaliser leur apparence.

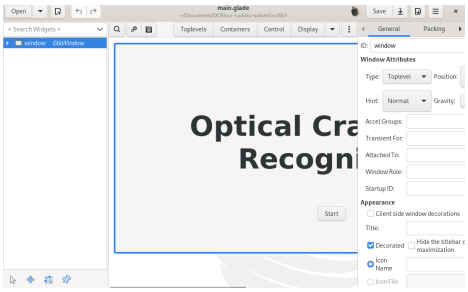

Figure 2: Image du logiciel Glade

L'un des avantages majeurs de Glade réside dans sa capacité à générer un fichier de configuration (comme illustré dans l'image ci-dessous) qui peut être directement utilisé par la bibliothèque GTK après son importation préalable. Cette intégration simplifie grandement le processus de conception de l'interface utilisateur et accélère le développement logiciel.

```
<object class="GtkScale" id="rotate_scale">
  <property name="visible">True</property>
  <property name="can-focus">True</property>
  <property name="hexpand">False</property>
  <property name="vexpand">False</property>
  <property name="fill-level">360</property>
  <property name="round-digits">0</property>
  <property name="digits">0</property>
  <property name="has-origin">False</property>
</object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="pack-type">end</property>
  <property name="position">3</property>
</packing>
```

Figure 3: Image du fichier généré par Glade

L'utilisation de Glade n'est pas seulement un gain de temps, mais elle contribue également à améliorer la qualité de votre interface utilisateur en offrant des outils de personnalisation avancés. Vous pouvez définir les propriétés visuelles de chaque composant, ajuster les couleurs, les polices, les alignements, et bien d'autres paramètres. Cela permet d'offrir une meilleure expérience utilisateur.

En résumé, l'utilisation de Glade pour la création de votre interface graphique est un choix judicieux qui améliore l'efficacité du développement logiciel tout en garantissant une interface utilisateur de haute qualité.

3.2 Les différentes pages de l'interface

L'interface graphique est composée de plusieurs sections. Tout d'abord quand on la lance, l'écran d'accueil apparait sur lequel figure le nom de notre groupe ainsi qu'un bouton permettant d'accéder a la véritable interface (voir image).

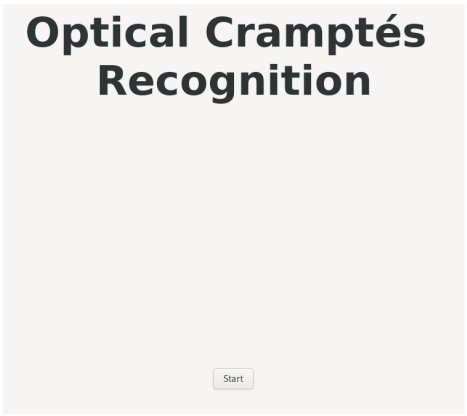

Figure 4: Image de l'écran d'accueil de notre interface

3.2.1 La page principale

Après avoir cliqué sur le précédent bouton, l'écran s'actualise et nous arrivons sur la page principal de l'interface. Sur celle-ci nous pouvons voir plusieurs boutons que je vais vous expliciter:

- Un bouton permettant de choisir un fichier disponible sur l'ordinateur de l'utilisateur avec la mention "Load Your Image". Si l'utilisateur choisit d'importer une image, elle s'affichera alors en haut de l'application.

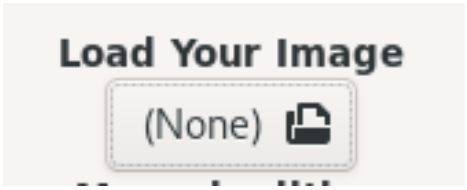

Figure 5: Image du bouton permettant d'importer l'image

- Un bouton "Rotation page" qui, quand l'utilisateur clique dessus, affiche la page de rotation (voir image).

Figure 6: Image du bouton permettant d'aller dans le menu de rotation

- Et enfin, un bouton "Quit" qui, comme son nom l'indique, permet de fermer l'interface (voir image).

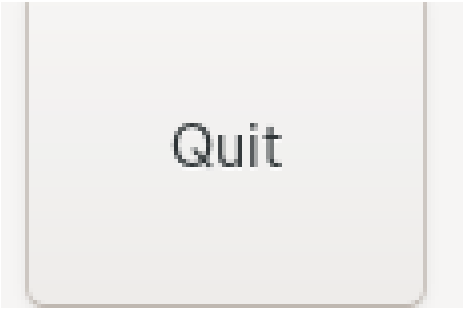

Figure 7: Image du bouton permettant de quitter l'application

3.2.2 La page de rotation

Lorsque vous accédez à cette page en cliquant sur le bouton précédemment mentionné, vous retrouvez l'image importée de la page précédente. Cependant, même si vous accédez directement à la page de rotation sans avoir importé d'image, le bouton permettant de choisir une image est toujours disponible pour que l'utilisateur puisse sélectionner une image à partir de son ordinateur. Une barre est également présente à l'écran, permettant à l'utilisateur de choisir l'angle souhaité pour la rotation de l'image. Une fois l'angle choisi, il ne reste plus qu'à cliquer sur le bouton "OK" pour effectuer la rotation de l'image (voir image ci-dessous).

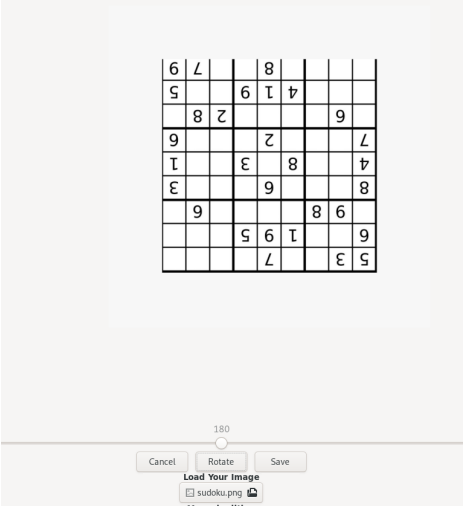

Figure 8: Image du menu de rotation avec une image tourné de 180°

Une fois la rotation fini, si l'utilisateur le souhaite, il peut enregistrer l'image dans un nouveau fichier en cliquant sur le bouton "Save"

3.3 Les fonctions nécessaires au fonctionnement de l'interface

Pour faire fonctionner cette interface graphique, il est essentiel de disposer d'une variété de fonctions qui assurent la gestion de l'interface. Voici un aperçu de certaines de ces fonctions clés :

- la fonction `changepanel()` : Cette fonction est responsable de l'actualisation de la page actuellement visible par l'utilisateur. Elle permet de passer d'une vue à une autre en fonction des actions de l'utilisateur. Par exemple, elle peut être utilisée pour passer de l'écran d'accueil à la page principale ou à d'autres parties de l'application.
- la fonction `showpage()` : Cette fonction est particulièrement utile pour alterner entre l'écran d'accueil et le menu principal de l'application. Elle peut être déclenchée en réponse à des événements spécifiques, tels que le clic d'un bouton de navigation.
- la fonction `changeimage()` : Cette fonction est responsable de l'affichage de l'image à l'utilisateur. Elle peut être utilisée pour charger une image depuis un fichier sélectionné par l'utilisateur et l'afficher à l'emplacement prévu dans l'interface.

En plus de ces fonctions liées à l'affichage, des fonctions plus spécialisées sont nécessaires. Par exemple :

- la fonction de rotation d'image : Une fonction spécifique doit être mise en place pour effectuer la rotation de l'image en fonction de l'angle choisi par l'utilisateur. Cette fonction peut appliquer des transformations à l'image pour obtenir l'effet souhaité.
- la fonction de modification de taille d'image : Une fonction dédiée peut être nécessaire pour redimensionner l'image en fonction des préférences de l'utilisateur. Cela peut être utile pour ajuster la taille de l'image avant de l'enregistrer.

Toutefois, la fonction la plus cruciale est la fonction `main()`. Elle agit comme le point d'entrée de l'application et relie le fichier généré à partir du logiciel Glade avec les fonctions que vous avez créées. Voici quelques-unes de ses tâches importantes :

- Importation du fichier généré par Glade.
- Création de la fenêtre principale de l'application.
- Création des différents composants de l'interface, y compris les boutons, les zones d'affichage, etc.
- Assurer que les interactions entre ces composants sont correctement gérées.

3.4 Ressentis

Pour ma part, les tâches que j'ai dû effectuer étaient un petit peu compliquées car je devais tout d'abord comprendre la bibliothèque GTK pour pouvoir faire cette interface. J'ai ensuite découvert Glade ce qui m'a permit d'avancer plus vite à la réalisation de l'interface. Mais cette partie était tout de même intéressante car cela reste toujours utile de savoir faire une interface graphique.

4 Arthur

Pour la première partie de la soutenance j'avais pour mission de faire le solver de sudoku et épauler Yan sur ses différentes parties (mais principalement sur la transformation de Hough (plutôt dans la partie théorique que pratique)).

4.1 Solver sudoku

J'ai tout d'abord commencé à travailler sur l'OCR en faisant le solver, partie plutôt simple mais essentielle pour tout le projet car c'est la partie la plus importante . Mon travail sur le solver s'est découpé en 3 axes majeurs:

- Vérifier les erreurs du sudoku entré
- Le solver
- Et enfin la fonction qui permet de tout réunir et de renvoyer un résultat comme demandé

4.1.1 Vérification des erreurs

Dans cette première sous partie j'ai tout simplement réalisé une fonction très simple qui vérifiait premièrement qu'il n'y avait pas plus de 9 colonnes et lignes ainsi qu'aucun chiffre ne se répétait .

4.1.2 Le solver

Le solver était très simple à réaliser, je l'ai réalisé sous forme de "backtracking" c'est un algorithme qui consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage. La méthode des essais et erreurs constitue un exemple simple de backtracking. Il pourrait ressembler de loin à un brut force mais en un peu plus optimisée.

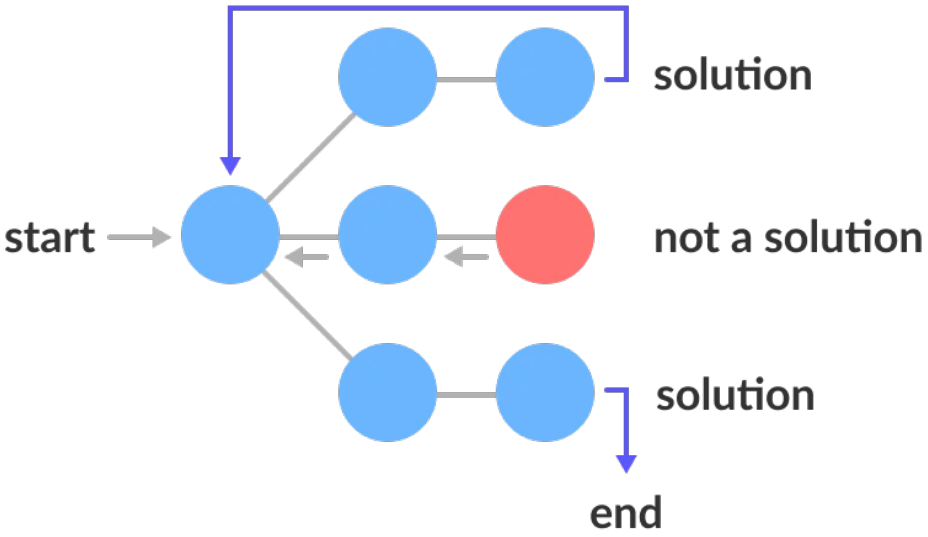

Figure 9: Exemple backtracking (Source Programiz.com)

4.1.3 La fonction principale

Contrairement au solver que nous utilisions initialement, la fonction principale réunissant toutes les étapes s'est révélée être bien plus complexe et exigeante. En effet, étant donné que nous avions l'habitude de travailler avec la grille de sudoku dans un format texte (comme illustré dans l'image ci-dessous, nous nous sommes retrouvés confrontés à la nécessité de réaliser une conversion complète vers une matrice d'entiers, une tâche essentielle avant d'appliquer l'algorithme de résolution. Une fois que le sudoku était résolu, il fallait ensuite le reconvertir dans le format initial, en format texte, tout en veillant à respecter scrupuleusement toutes les spécifications nécessaires pour obtenir le format final approprié (en évitant soigneusement d'avoir des espaces en excès et en positionnant les espaces à la fin, si requis). Cette étape post-résolution a pris quelques heures, un laps de temps considérable qui a eu pour effet de décourager toute tentative de s'y replonger ultérieurement. En ce qui concerne l'optimisation, il convient de noter que notre solver est si remarquablement performant qu'il est capable de résoudre avec succès n'importe quel sudoku en moins d'un dixième de seconde.

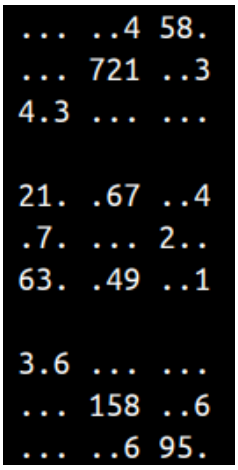

Figure 10: Format d'entrée et de sortie

4.2 Transformée de Hough - Théorie

Dans cette partie, je vais vous expliquer la partie théorique de la Transformée de Hough, et Yan vous expliquera la partie pratique et la manière dont nous l'avons implémentée.

Pour tout ce qui est transformation d'image, parmi les filtres de Sobel, celui de Canny ou même les nuances de gris, pour moi le plus intéressant et le plus complexe était bien la transformée de Hough. La transformée de Hough permet, avec des points, de retrouver des formes géométriques comme des lignes, des cercles ou des ellipses (voir Figure 11). Il y a 2 méthodes différentes pour trouver les lignes sur une image :

- Une méthode non optimisée que nous n'avons pas utilisée, mais qui se base sur l'équation cartésienne d'une droite telle que $y = ax + b$. Cependant, le problème avec cette méthode est que b peut être compris entre 0 et $+$, ce qui est problématique si nous voulons gérer ce problème de façon rationnelle
- Pour résoudre ce problème, nous pouvons donc passer sous une équation polaire telle que

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

Ici, c'est plus simple car Θ ne varie que entre 0 et Π, et ρ, entre $-2R$ et $2R$, R étant la diagonale de notre image calculée avec Pythagore. Pour calculer donc les lignes, on crée un accumulateur (qui sera sous forme de matrice) et on "trace" les fonctions sinusoïdales dans cet accumulateur en faisant varier Θ et en associant à chaque résultat son ρ (voir Figure 12). Ainsi, lorsque que nous avons fini de tracer les courbes pour les pixels blancs (car les pixels sont soit en noirs soit en blancs), nous avons donc des valeurs différentes dans chaque case de notre accumulateur. Dès lors, nous prenons le maximum et nous avons donc notre Θ et notre ρ pour notre équation polaire. Pour notre sudoku, on aura juste à modifier notre algorithme, non pas en cherchant un maximum, mais plusieurs maximums pour tracer nos lignes.

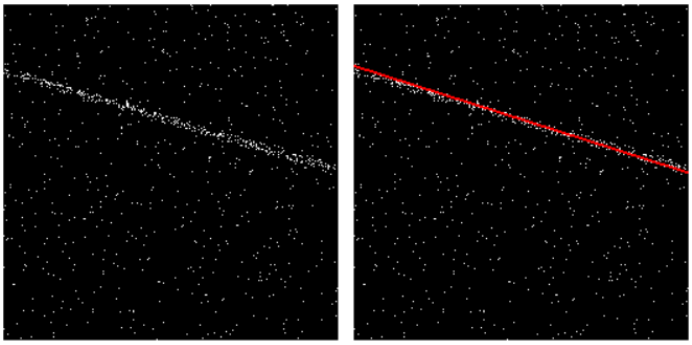

Figure 11: Transformation de Hough

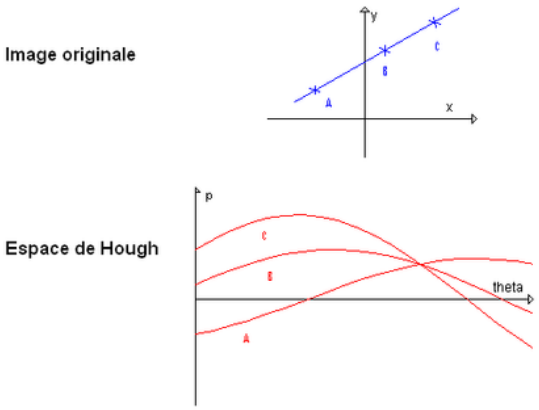

Figure 12: En haut nos points dans notre repère de base et et en bas la retranscription de chaque point avec l'équation polaire

4.3 Ressentis

Pour cette première partie, qui est passée très vite je ne l'ai pas trop ressenti de manière négative ou même éprouvé de difficulté car ce que j'ai réalisé était plutôt simple. Mais pour la prochaine soutenance je dois me préparer car je sais que j'aurais beaucoup plus à faire.

5 Yan

Pour la première soutenance, deux tâches m'ont été assignées. Dans un premier temps, commencer le prétraitement de l'image (enlever les couleurs, détecter les contours, ...). Dans un second temps, j'ai dû coder l'algorithme de détection des lignes et de découpage de la grille (en collaboration avec Arthur qui m'a aussi aidé sur cette partie).

5.1 Opérations sur les pixels

Pour pouvoir mener à bien l'ensemble des tâches à réaliser, j'ai commencé par coder quelques opérations sur les pixels. Parmi elles, j'ai fait :

- Une fonction `getPixel()`, qui renvoie la valeur du pixel à une position `[x, y]` donnée en paramètre.
- Une fonction `setPixel()`, qui permet de remplacer un pixel à la position `[x, y]` par un autre pixel passé en paramètre de la fonction après les coordonnées.
- Et enfin une fonction `getIntensity()` renvoyant la valeur d'intensité du pixel aux coordonnées `[x, y]` de l'image. Cette fonction sert par la suite pour savoir si un pixel est blanc ou noir par exemple.

5.2 Prétraitement de l'image

Le prétraitement de l'image est une étape phare du projet. Il faut ajuster le contraste de l'image, retirer les couleurs, réussir à passer l'image en noir et blanc sans perdre les informations de la position de la grille, ... Toutes ces étapes sont cruciales et nécessaires pour arriver à détecter la grille sans souci. Le prétraitement que nous avons n'est pas encore parfait. En effet, nous n'avons aucune modification de contraste pour l'instant (mais pas d'inquiétude, ces tâches sont à réaliser pour la dernière soutenance). Néanmoins, les objectifs pour la soutenance sont largement atteints puisque je suis même allé plus loin que prévu dans le traitement. J'ai donc bien préparé le terrain pour la prochaine soutenance en m'avançant sur cette tâche.

5.2.1 Nuances de gris

La première étape du prétraitement a été le passage en nuances de gris. Le passage en nuances de gris lors du prétraitement d'une image est essentiel car il permet de simplifier l'information visuelle en réduisant l'image à une seule composante de luminosité. Cela facilite l'analyse des contours, des formes et des textures de l'image, tout en réduisant la complexité du traitement par rapport à une image couleur. De plus, en convertissant une image en nuances de gris, on élimine les variations de couleur qui pourraient ne pas être pertinentes pour la tâche d'analyse en cours, ce qui permet de concentrer l'attention sur les caractéristiques structurelles de l'image. En somme, le passage en nuances de gris est une étape cruciale pour de nombreuses applications d'analyse d'image, y compris la détection de contours qui nous intéressera par la suite.

5.2.2 Inversion des couleurs

L'inversion des couleurs sur une image en noir et blanc relève plutôt du détail que d'une obligation. Cependant, cela permet tout de même de préciser certains points en améliorant par exemple la visibilité de l'image à l'oeil nu. Sinon, travailler en noir et blanc inversé était un choix personnel.

5.2.3 Réduction du bruit

Ce que l'on appelle le "bruit" sur une image se réfère aux variations indésirables, aléatoires et souvent imperceptibles, des niveaux de luminosité ou de couleur des pixels, ce qui peut altérer la qualité visuelle de l'image.

Pour éviter ça, il existe diverse méthodes. Celle que nous avons choisi d'utiliser est l'algorithme du filtre Médian. Le filtre médian est une technique de traitement d'image qui vise à réduire le bruit et à lisser une image. Il fonctionne en remplaçant la valeur de chaque pixel par la valeur médiane des pixels voisins dans une fenêtre spécifiée. Ce processus permet d'éliminer les valeurs d'intensité extrêmes, ce qui est particulièrement efficace pour éliminer les pixels parasites sans affecter significativement les contours ou les détails de l'image.

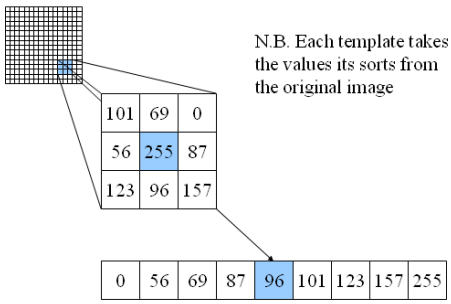

Figure 13: Illustration du filtre médian sur un noyau de taille 3

5.2.4 Le filtre de Sobel

Le filtre de Sobel est un opérateur de détection de contours couramment utilisé en traitement d'images pour mettre en évidence les changements d'intensité lumineuse, tels que les bords, dans une image.

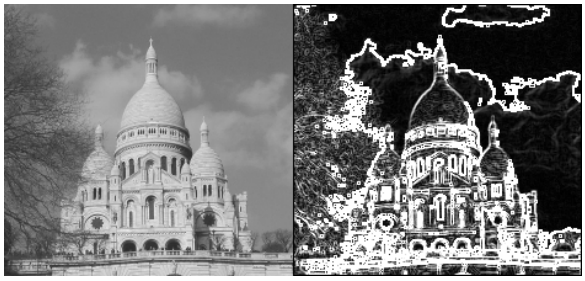

Figure 14: Exemple du filtre de Sobel

Dans mon cas, j'ai décidé d'implémenter le filtre de Sobel en utilisant deux noyaux (kernels) : un pour détecter les countours horizontaux, et l'autre pour les contours verticaux. Pour chaque pixel de l'image, le filtre calcule la somme des produits entre les valeurs du noyau et les valeurs de luminosité des pixels voisins. La combinaison des résultats de ces deux filtres de Sobel donne une

bonne estimation du gradient d'intensité, qui sera identifié par la suite pour identifier les bords et les contours de l'image.

5.2.5 Le filtre de Canny

Le filtre de Canny est un filtre qui donne de très bons résultats sur certaines images, mais de plus mauvais résultats sur d'autre. Ainsi, son implémentation ne sera sans doute pas la même dans le rendu final. Le filtre de Canny est une technique de détection de bords en traitement d'images qui vise à identifier les contours et les structures significatives d'une image tout en réduisant le bruit.

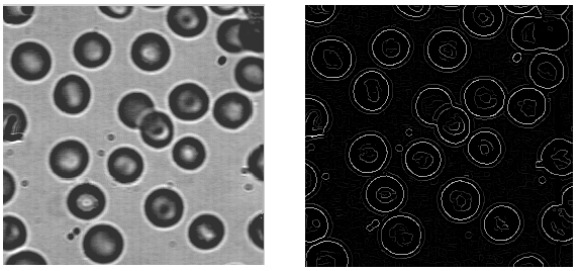

Figure 15: Exemple avant/après l'application du filtre de Canny

Dans notre cas, le filtre de Canny prend en paramètre deux seuils qui serviront à déterminer si le pixel est un pixel de fond, ou un pixel de contour. L'implémentation du filtre de Canny ce fait comme cela : tout d'abord, on applique le filtre de Sobel afin de détecter les contours de l'image. Ensuite, on parcourt l'image pixel par pixel. Si l'intensité du pixel est inférieure au seuil bas, on considère ça comme un pixel de fond et on le met en noir. Si le pixel est au dessus du seuil haut, on le compte comme un pixel important à garder (du contour par exemple), et on le met en blanc. Si le pixel est compris entre les deux seuils, une vérification est effectuée en comparant leur intensité avec celle de leurs voisins pour déterminer s'ils représentent un maximum local dans le gradient d'intensité. Si c'est le cas, ils sont également considérés comme des pixels de contour (mis en blanc). Enfin, une étape de suivi des contours est effectuée pour renforcer les contours en reliant les pixels de contour voisin. Si un pixel est déjà considéré comme un contour potentiel et est entouré de pixels de contour, il est marqué comme un contour.

```
void applyCannyFilter(SDL_Surface *image, Uint8 lowThreshold, Uint8 highThreshold);
```

Un problème du filtre de Canny est le dédoublement des contours. Ce phénomène survient souvent à cause d'un réglage trop imprécis des valeurs seuils utilisées. Lorsque les seuils de détection des contours ne sont pas correctement réglés, des contours faibles peuvent être détectés légèrement décalés par rapport aux contours forts, créant ainsi une réponse double.

Le résultat final est une image en noir et blanc, où seuls les contours significatifs sont mis en évidence, tout en supprimant le bruit indésirable.

5.2.6 Dilatation et érosion

La dilatation est une opération qui consiste, dans notre cas à agrandir les contours de l'image (pixels blancs) tout en réduisant les régions de fond (pixels noirs). Cette étape est utilisée afin de

réduire les petits trous dans les contours, et pour "recoller" les parties dédoublées par le filtre de Canny.

L'érosion est l'opération inverse de la dilatation, elle réduit les régions d'objets tout en agrandissant les régions de fond. Pour nous, elle est implémentée en parcourant toute l'image et en remplaçant chaque pixel par la valeur minimale des pixels de son voisinage. Cela permet également d'éviter d'avoir des bords trop épais dans certains cas.

5.3 Détection de la grille

Pour cette partie, j'ai été aidé par Arthur. Nous avons dû implémenter deux algorithmes : l'un pour détecter les lignes sur une image, et un deuxième pour limiter le nombre de lignes sur l'image et ainsi garder uniquement les "vraies" lignes. Pour ce faire, nous avons décidé d'utiliser la transformée de Hough.

5.3.1 Transformée de Hough - Implémentation

Comme dit précédemment par Arthur, la transformée de Hough est une technique de traitement d'image qui permet de détecter des formes spécifiques, telles que des lignes, des cercles ou des ellipses, en convertissant les coordonnées des points d'une image en paramètres dans un espace appelé "espace de Hough." Dans notre cas, nous nous sommes penchés sur la détection de lignes dans une image. Pour ce faire, la transformée de Hough convertit les coordonnées cartésiennes des points de l'image en coordonnées polaires (rho et theta) dans l'espace de Hough. Les points appartenant à une même ligne dans l'image sont représentés par des courbes sinusoidales qui se croisent au point correspondant aux paramètres rho et theta de la ligne.

```
typedef struct
{
    double rho;
    double theta;
} HoughLine;
```

Pour notre implémentation, il a d'abord été important de créer un type qui puisse stocker une ligne sous forme de coordonnées polaires (type défini juste au dessus). Ensuite, nous avons suivi ces étapes :

- Tout d'abord, nous avons créé un accumulateur pour stocker les votes associés aux paramètres rho et theta.
- Ensuite, on parcourt tous les pixels de l'image et, pour chaque pixel qui correspond à un bord blanc (intensité de 255), on calcule les valeurs rho et theta pour chaque angle (theta) prédéfini.
- Ensuite on incrémente les compteurs dans l'accumulateur pour les paires rho-theta qui correspondent aux coordonnées des bords détectés.
- Dans un second temps ensuite, on cherche le nombre maximum de votes dans l'accumulateur et on défini un seuil basé sur un pourcentage de ce nombre maximum pour filtrer les votes significatifs.
- On parcourt donc à nouveau l'accumulateur pour extraire les lignes détectées dont le nombre de votes dépasse le seuil.

- Enfin, on renvoie une liste de type `HoughLine *` avec toutes les lignes que l'on a gardé.

Cette implémentation nous permet de détecter les lignes sur des images.

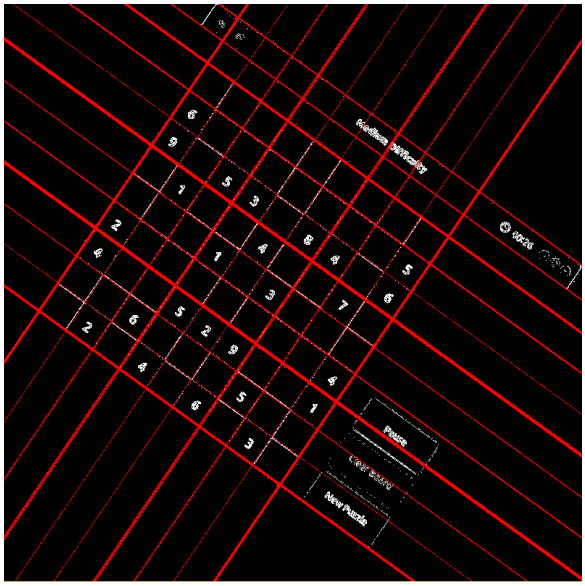

Figure 16: Illustration de la transformée de Hough

Le prétraitement effectué ici est assez simple : un passage en nuances de gris, puis une inversion des couleurs, et enfin l'application du filtre de Canny. Les lignes détectées sont ici tracées en rouge sur l'image.

Seulement, même si cela fonctionne, on comprend assez vite que si les lignes rouges sont si épaisses, c'est parce que le programme trace toutes les lignes trouvées par la fonction. Il a donc fallu implémenter un algorithme qui a pour objectif de fusionner les lignes similaires afin de réduire le nombre de lignes et d'éliminer les doublons. Ainsi, dans notre implémentation, on compare les lignes en fonction de leurs paramètres rho et theta dans le plan de la transformée de Hough. Pour savoir si deux lignes sont identiques, on calcule la différence entre les valeurs de deux lignes. Si cette différence est trop petite, alors les lignes sont collées (ou presque) et ne montrent rien de différent. On fusionne donc ces lignes.

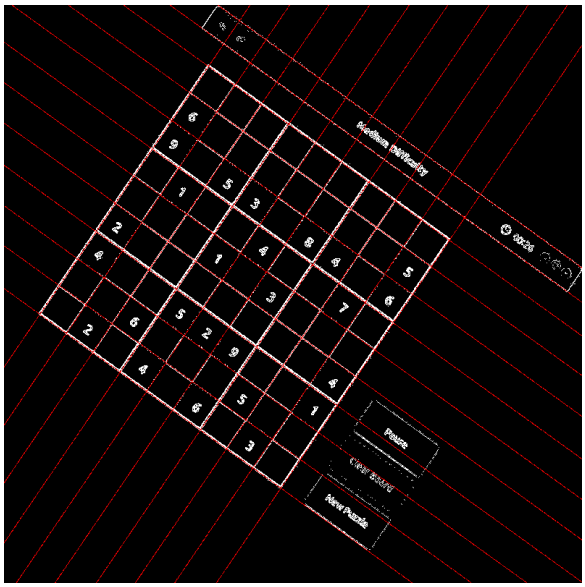

Figure 17: Illustration de la transformée de Hough après fusion des lignes

Ici, on voit bien que le programme n'a gardé qu'une ligne à chaque fois.

5.3.2 Transformée de Hough - Problème(s) rencontré(s)

Quand j'ai codé pour la première fois la transformation de Hough j'ai rencontré plusieurs problèmes le premier étant que toutes les lignes n'étaient pas prises en compte.

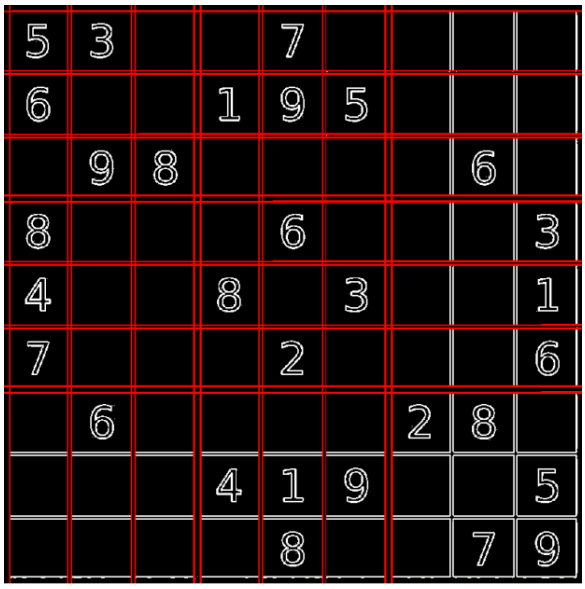

Figure 18: Résultat de la transformée de Hough avec des lignes manquantes

Je n'ai pas compris quel était le problème jusqu'à ce que je réalise que j'avais malencontreusement oublié de prendre en compte que mon ρ pouvait être négatif. J'ai donc du modifier la taille de mon accumulateur pour pouvoir accueillir tout les ρ y compris négatif. La deuxième partie de la solution à de problème était d'augmenter par 2 la taille de la diagonale (`diagonalLength`), car en effet comme expliqué par Arthur au dessus ρ peut être compris entre 2R et -2R, R étant la diagonale de notre image (en nombre de pixels). Après avoir réglé ces 2 problèmes mon code fonctionnait parfaitement.

5.4 Découpage de la grille

Pour le découpage de la grille, j'ai implémenté la fonction `saveSquares` qui découpe une image de sudoku en 81 cases égales, correspondant aux cellules de la grille. Chaque case est extraite de l'image d'origine, puis enregistrée individuellement au format PNG dans un dossier appelé "saved_images". La fonction vérifie également si le dossier "saved_images" existe, et le crée s'il est absent.

5.5 Ressentis

Pour ma part, les tâches de cette première soutenance on représenté un "rush" vers la fin. En effet, j'avais bien pris de l'avance sur le début du projet en commençant assez tôt le prétraitement, partie assez intéressante et instruisante (car on voit bien son avancée au fur et à mesure des opérations qu'on applique sur la matrice des pixels de l'image). C'était intéressant de voir à quel point les maths étaient liées au concept d'image en informatique. Par contre, pour la partie détection de la grille, j'avais clairement sous estimé la tâche a accomplir et c'est bien pour cela que je remercie Arthur de m'avoir aidé, sans quoi le code ne marcherait encore qu'à moitié.

6 Conclusion

Ainsi, cette soutenance s'achève sur un succès car nous avons atteints tous nos objectifs. Tout le monde a pu s'adapter au langage C qui était nouveau pour chacun de nous, à la librairie SDL pour certains, à la manipulation d'interfaces graphiques avec GTK pour d'autre, et enfin au fonctionnement d'une intelligence artificielle, notion informatique assez complexe mais pas moins intéressante. Nous avons rencontrés plusieurs problèmes au fil du temps, mais le travail en groupe s'est parfaitement déroulé et à permit de finir dans les temps. Ce projet commence bien, mais n'est pas fini et nous sommes tous impatients de nous plonger sur la suite du projet, ainsi que les optimisations des tâches que nous avons accomplis pour cette soutenance.