Rapport de Première Soutenance
Projet Dodoku

Valentin UHLRICH Dylan INNOU
Nicolas PREVOST Alexandre BAO

Octobre 2021

EPITA | Promotion 2025

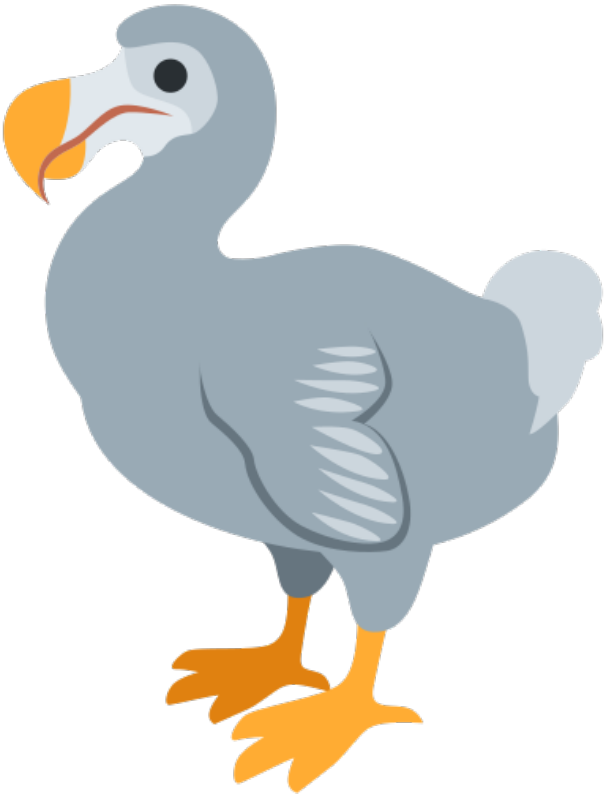


Table des matières

1 Introduction

Ce projet consiste a réaliser un logiciel de type OCR (Optical Character Recognition) qui résout une grille de sudoku. L'application prendra en entrée une image puis elle ressortira en sortie le sudoku résolu. Elle disposera d'une interface graphique afin de simplifier son utilisation.

Durant ce premier projet en langage C nous allons apprendre son fonctionnement. Il nous faut nous familiariser avec les pointeurs que nous allons beaucoup utiliser tout le long du projet, et qui sont une nouveauté pour nous. De plus, ce projet nous permet de découvrir plein de domaines à la fois. Nous devront, pour la plupart, découvrir et comprendre le fonctionnement des algorithmes à utiliser. Ensuite, il nous faut implémenter ces algorithmes en langage C et c'est souvent source problème car nous avons très peu d'outils à notre disposition et il faut trouver la meilleur façon de faire.

1.1 Présentation de l'équipe

Nous sommes tous les quatre des élèves de la B2. Nous étions dans la même classe l'année et même dans le même projet de S2 pour la plupart.

Valentin UHLRICH (Chef de Projet)

Bonjour moi c'est Valentin, je suis le chef de projet, j'ai donc la double casquette de m'occuper de mes parties mais aussi de faire en sorte que le projet soit le mieux organisé possible. Dans le projet, je m'occupe du réseau de Neurone et de la résolution du Sudoku. J'aide aussi mes coéquipiers dans les tâches où je suis suppléant.

Dylan INNOU

Bonjour, mon nom à moi c'est Dylan Innou, ça fait maintenant une année que je suis à l'EPITA et j'ai acquis plusieurs connaissances dans l'informatique. L'intelligence artificielle à toujours été pour moi un sujet qui m'intrigue. Je serais content de m'occuper des pré-traitements de l'image et se son interface, mais aussi d'aider mes collègues à la réalisation de ce projet ambicieux.

Nicolas PREVOST

Bonjour, je suis Nicolas Prevost. Dans l'informatique, je m'intéresse beaucoup au traitement et manipulation d'images, c'est pourquoi je serais dans ce projet en charge du pré-traitement de l'image et de de la rotation de celle-ci. J'épaulerai également mes compagnons pour leurs taches respectives. Je compte aussi me servir de ce projet comme prétexte pour approfondir ma connaissance du C et le fonctionnement des réseaux de neurones.

Alexandre BAO

Je m'appelle Bao Alexandre, je suis un étudiant d'Epita, j'ai appris le langage Ocaml, Python et C# durant ma première année d'Epita, j'ai également été chef de groupe du projet de jeu vidéo durant la S2. Durant le projet de S3, je souhaiterai acquérir quelque bases sur le machine learning et apprendre à coder en C.

2 Répartition des tâches

Tâches :	Responsable	Suppléant
Pré-traitements et Rotation	Nicolas	Dylan
Détection de la grille et cases	Dylan	Nicolas
Réseau de neurones	Valentin	Alexandre
Sauvegarde des données	Dylan	Valentin
Résolution du Sudoku	Valentin	Nicolas
Sauvegarde de la grille	Nicolas	Alexandre
Interface graphique	Alexandre	Dylan

TABLE 1 – Tableau des répartitions des tâches

Nous avons essayé de répartir les tâches de façon équitable mais aussi de sorte à que chacun puisse choisir les parties sur lesquels il voulait travailler. Chaque responsable de Tâches doit superviser cette partie en demandant aux autres de l'aider s'il a besoin d'aide étant donnée que nous sommes un groupe. Les suppléants sont là pour aider le responsable en cas de besoin.

3 Conceptions

3.1 Pré-traitements et Rotation

(Nicolas et Dylan)

Le pré-traitement de l'image permet de la rendre plus simple d'utilisations pour les autres opérations qui seront effectuées sur la grille de sudoku. Ce traitement est nécessaire avant de faire la détections des grilles pour découper les images individuellement, mais également pour rendre les chiffres plus lisibles et plus reconnaissables pour le futur réseau de neurones.

3.1.1 Grayscale

Avant tout traitement, nous mettons l'image en échelle de gris. Pour cela, chaque composante de chaque pixel (R,G,B) sera égale a la moyenne des 3 valeurs. En effet les couleurs ne nous sont pas utiles, et cela permet de manipuler un pixel comme un seul chiffre plutôt que trois.

3.1.2 Réduction de bruit

Les images possèdent en général un "bruit", ce que l'on appelle en photographie le "grain". Ce sont entre autres des pixels qui se dégagent de ses pixels voisins, et sont en quelque sortes des anomalies de l'image. Et dans notre quête de l'image la plus parfaite, notre but est de réduire le plus possible ce bruit. Nous utilisons la méthode du flou gaussien :
Cette méthode applique a chaque pixel de notre image un filtre dit passe-bas. Le nouveau pixel est alors la moyenne des 24 pixels voisins et lui même, avec des poids appliqués selon la distance à celui-ci. L'image perd alors en netteté, mais ce n'est pas un problème car elle sera mise plus tard en noir et blanc.

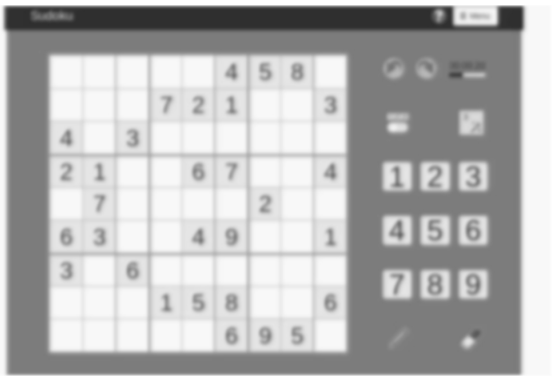
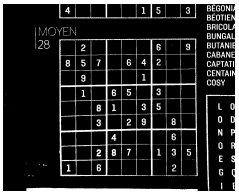


FIGURE 1 – image 3 floutée 8 fois

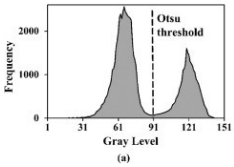
Ici l'image est très fortement floutée (volontairement), et on peut remarquer qu'elle ne possede plus de défaut, a part une perte de netteté.

3.1.3 Binarisation de l'image

La binarisation d'une image consiste à transformer une image, en échelle de gris, en une image noir et blanc. Cependant, la simple moyenne des valeurs des pixels ne pourra pas faire l'affaire car si jamais l'image donnée est trop claire ou trop foncée. La solution vient avec la méthode d'Otsu. Cette méthode consiste à faire la somme des valeurs individuels des pixels. Chacune des sommes individuelles va être placées dans un histograme. On s'aperçoit vite (cf. la figure ci-dessous) que des groupement de gris se retrouvent et il nous suffit juste de les séparer grâce à une formule très simple.



(a) Ce que donne la fonction Otsu avec l'image d'une grille



(b) Exemple d'un histogram

Ce séparage nous permettra de trouver un seuil, il sera donc unique à chaque image et le rendu en sera bien meilleur.

3.1.4 Rotations

Une image peut avoir un angle, et ainsi la grille n'est pas droite par rapport au cadre de la photo. Or le reseau de neurone sera lui entraîne a reconnaître des images de nombres droits. De plus pour la découpe de la grille en sous image, il est plus simple de découper des images si les carrés sont droits. Pour cette soutenance, la rotation se fait manuellement, c'est à dire qu'il faut indiquer la fonction `rotate()` l'angle a appliquer afin qu'elle génère une image plus droite. Dans le futur, une fonction `autoRotate()` détectera elle-même l'angle a appliquer et fera la rotation elle-mémé.

3.2 Détection de la grille et cases

(Dylan et Nicolas)

Une fois les traitements effectué sûr l'image, il faut redimensionner l'image pour qu'il y ait que la grille du sudoku et retirer le contenu parasite qui peut se trouver sur les côtés du soduku. Puis découper cette grille en cases avec tous les nombres.

3.2.1 Détection de la grille

La détection de ligne se fera à l'aide de la Transformée de Hough.
La Transformée de Hough est une méthode appliquée sur les images afin de détecter des lignes, des formes, ect...
Les lignes droites en mathématique sont facilement reconnue à l'aide de

cette fonction : $y = ax + b$.
Du coup, lorsque l'on donne une image à notre programme informatique, il doit trouver ce a et ce b.
Vue que c'est un programme informatique, il n'a qu'à assumer que c'est tous les a et les b possibles. Et on le fait pour chacun des points de l'image. On obtiendra à la fin plusieurs fonctions affines qui se croisent et l'intersection se ses fonctions affine corresponds à l'endroit où le plus de points ont pour lignes communes les éléments a et b. Par facilité, on utilisera la représentation par : $y = \frac{\rho - x * \cos(\theta)}{\sin(\theta)}$

Lorsque l'on applique la Transformée de Hough sur sur image, on obtient ce type d'image. Selon son théorème, on peut conclure que les points les

FIGURE 3 – Aperçu de la Transformée de Hough

plus noirs sont les points où le rho et le theta sont les arguments qui ont le plus de points qui les traversent. Ainsi, on a juste à choisir seulement les arguments avec le plus de points qui les traversent pour détecter les lignes de notre Sudoku.
Lorsque les lignes principales sont détectées, il nous reste plus qu'à découper l'image selon les lignes afin de les séparer pour les donner au réseau de neurones.

3.2.2 Détection des cases

La détection des cases sera faites dans un second temps. De même que la détection de la grille, nous utilisons la Transformée de Hough pour découper le Sudoku en cases et ainsi faire passer chaque nombre dans le réseau de neurones.

3.3 Réseau de neurones

(Valentin et Alexandre)

Construire un réseau de neurone peut paraître infaisable au premier abord. Mais après s'être documenté sur le sujet, nous avons compris le principe et son fonctionnement.

3.3.1 Fonctionnement

Le principe du réseau de neurones est de lui donner en entrer des valeurs puis, en faisant passer ces valeurs dans des neurones nous, obtenons des sorties qui permettent de déterminer quelque chose. On peut par exemple dans le cas de la porte XOR avoir deux entrées et une sortie. Afin de calculer la valeur de sortie, les neurones qui se situent dans la couche cachée ont des poids, et ces poids permettent de pondérer la valeur.

En premier nous allons propager les valeur vers l'avant (forward propagation) qui vont calculer un par un les valeurs des neurones avec les poids jusqu'à arriver aux neurones de sorties. Le calcul des valeurs des neurones un à un se fait avec la formule $\sum(W_i * V_i)$ où V est la valeur du neurone précédent avec le poids de la branche W. Ensuite il faut passer le résultat dans une fonction d'activation (sigmoïde dans notre cas) qui permet de déterminer si le neurone est activé ou non.
Une fois cette propagation effectuée, si le réseau n'a jamais été entraîné, nous obtiendrons des valeurs aléatoires en sortie. C'est là qu'intervient la propagation arrière (back propagation).

La propagation vers l'arrière va permettre de recalculer les poids pour qu'ils correspondent mieux aux attentes. Nous allons donc en premier lieu calculer le taux d'erreur de chaque neurones de sortie avec $C = valeur\ attendu - valeur\ obtenu$. Ce taux d'erreur va nous permettre de revenir en arrière et de modifier les poids suivant la valeur du taux d'erreur. C'est donc grâce à ça que notre réseau de neurones va devenir intelligent aux fur et à mesure des allers-retours.

Étant donné que l'apprentissage du réseau de neurones peut être long, il est préférable d'enregistrer les poids des neurones une fois que nous avons des valeurs satisfaisantes. Le plus souvent les poids sont enregistrés dans un fichier puis ils sont chargés au démarrage du réseau.

3.3.2 Implémentation

Passons maintenant a la partie implémentation. Après une brève explication du fonctionnement d'un réseau de neurones, il nous faut maintenant trouver une solution pour l'implémenter en C. Le plus simple que nous avons trouvez c'est d'utiliser des tableaux de structs avec 3 structs : Network, Layer et Neurones.

Une fois ces structs et tableaux initialisés nous allons pouvoir générer des valeurs aléatoires entre -1 et 1 pour les mettre dans les poids. Ensuite nous pouvons entainer notre réseau de neurones en propagent les valeurs et en revenant en arrières puis une fois que l'on trouve des valeurs satisfaisante on peut s'arrêter. Les poids reste enregistrer dans les structs qui contiennes un tableaux avec leurs poids.

Pour tester notre réseau il suffit de faire une propagation vers l'avant et de regarder les valeurs dans les neurones de sortie. Il est donc très simple d'utiliser notre réseau une fois implémenté avec nos fonctions en C.

Maintenant afin d'éviter de relancer l'entraînement à chaque redémarrage du programme nous avons fait un moyen de sauvegarder et de charger le réseau à partir d'un fichier. Il est donc possible après l'entraînement

de sauvegarder les poids dans un fichier en précisant l'endroit de sauvegarde. Si on veut charger le réseau il est possible de donner le chemin du fichier et il va lire le fichier et initialiser le réseau avec les valeurs qui sont dans le fichier. Le réseau sera donc directement fonctionnel et aucun entraînement ne sera nécessaire.

3.3.3 Reconnaissance de la porte XOR

Pour cette première soutenance, nous avons réalisé un réseau de neurones capables d'apprendre la porte XOR. Cette porte est très simple à faire apprendre à notre réseau de neurones mais elle permet de tester si celui-ci est fonctionnel. Nous allons donner deux entrées avec des 0 ou 1 et il va nous ressortir une valeur qui se rapproche de 1 ou de 0. Dans notre cas il nous faut environ 500 itérations d'entraînement pour que le réseau de neurones puisse bien comprendre et ressortir des valeurs satisfaisantes.

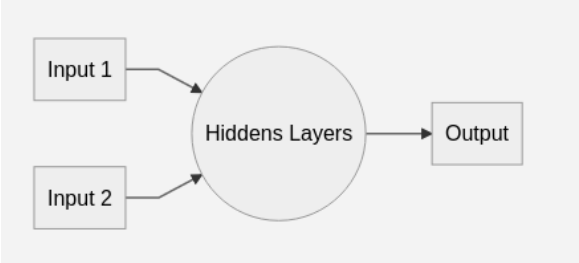


FIGURE 4 – Schéma du réseau pour le XOR

3.3.4 Reconnaissance des nombres

Pour le moment cette fonctionnalité n'est pas encore disponible. Mais pour la dernière soutenance il nous faudra implémenter cette partie qui consistera à faire entrer tous les pixels de l'image dans le réseau de neurones et qu'il en déduise le nombre auquel il correspond.

3.4 Sauvegarde des données

(Dylan et Valentin)

Cette partie consiste à sauvegarder et à charger les données qui sont produites par le réseau de neurones. Cela permet d'éviter de réentrainer les neurones à chaque redémarrage du programme.

3.4.1 Sauvegarde

Pour sauvegarder les données de notre réseau nous allons utiliser des fichiers qui seront enregistrés sur le disque dur. Les données qui sont utiles pour enregistrer notre réseau sont principalement les poids de chaque branches entre les neurones. Nous allons utiliser cette forme dans le fichier :

```
nb_lignes nb_colonnes | nb_input nb_hiddens_layers nb_hiddens nb_output
poids_1_neurone_1_layer_1 poids_2_neurone_1_layer_1 ...
poids_1_neurone_2_layer_1 poids_2_neurone_2_layer_1 ...
...
poids_1_neurone_1_layer_2 poids_2_neurone_1_layer_2 ...
...
```

L'écriture n'est donc pas compliquée, il suffit juste de récupérer un par un les données du réseau et de les écrire dans le fichier. Nous avons dû préciser les dimensions du fichier pour que lors du chargement du fichier, l'initialisation du tableau soit plus simple.

3.4.2 Chargement

Le chargement des données à partir d'un fichier est plus compliqué que la sauvegarde. En effet, il faut parser un par un les caractères et couper à chaque espace. Une fois que nous avons toutes les valeurs sous forme de chaîne de caractères, il faut les convertir au format `double` qui est le type utiliser dans le réseau. Pour nous simplifier la tâche nous avons utilisé la fonction `atof(char*) -> double` qui permet de faire la conversion a notre place. En effet, la conversion en `double` aurait été fastidieuse avec les virgules. Pour revenir a notre chargement de réseau, une fois que nous avons toutes les valeurs au format `double`, il nous suffit juste de les mettre dans les tableaux du réseau à la bonne position. À la fin notre réseau de neurones est donc fonctionnel et nous pouvons faire une propagation vers l'avant pour vérifier qu'il fonctionne bien.

3.5 Résolution du Sudoku

(Valentin et Nicolas)

Dans cette partie nous nous occupons de la résolution du sudoku une fois qu'il a été reconnu avec l'intelligence artificielle. Après une documentation sur internet, nous avons remarqué qu'il existe plusieurs façons de résoudre celui-ci. Mais la méthode la plus simple à implémenter et la plus efficace est le Backtracking. Cette méthode consiste à essayer toutes les combinaisons possibles et revenir en arrière quand on est bloqué. L'essaie de toutes les combinaisons possibles est faciles à calculer par l'ordinateur et cela est très rapide.

Afin de permettre de tester notre programme de résolution nous avons en premier lieu, intégrer un parser qui va lire un fichier avec une grille non résolu. Cette grille sera sous cette forme :

```
... ..4 58.
... 721 ..3
4.3 ... ...

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

```
8
9  3.6 ... ...
10 ... 158 ..6
11 ... ..6 95.
```

Une fois cette grille chargée grâce à la fonction `init_grid()`, elle est sauvegardée dans un tableau à deux dimensions qui est plus facile à manipuler. Une fois que nous avons la grille en mémoire nous allons pouvoir nous occuper de la résoudre pour cela nous allons découper cela en plusieurs fonctions :

Les fonctions `is_..._solved()` Elles consistent à vérifier si le Sudoku est résolu sur la ligne, la colonne et dans chaque carré de 3x3.

Les fonctions `already_in_...()` Elles consistent à vérifier si le nombre spécifier en argument est déjà dans la ligne, la colonne ou dans le carré 3x3.

La fonction `solve()` Elle va essayer toutes les possibilités et vérifier si cette essaie est correcte avec les fonctions ci-dessus.

Pour résoudre le Sudoku il suffit juste d'appeler la fonction `solve()` qui va s'occuper de modifier la grille passée en argument avec les nouvelles valeurs afin de rendre la grille complète. Pour pouvoir voir le résultat, la grille résolu est sauvegarder au même endroit que celle de début avec l'extension `.out`.
Vous pouvez retrouver un exemple d'exécution ci-dessous :

```
$ cat tests/solver/grid_01
... ..4 58.
... 721 ..3
4.3 ... ...

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

3.6 ... ...
... 158 ..6
... ..6 95.
$ ./build/solver tests/solver/grid_01
Sudoku solved and saved as "tests/solver/grid_01.out"
$ cat tests/solver/grid_01.out
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

356 492 718
792 158 436
841 376 952
```

3.6 Sauvegarde de la grille

(Nicolas et Alexandre)

La grille résolue peut être sauvegardée dans un fichier grid, comme montré dans l'exemple précédent. Mais nous avons aussi mis en place un système, qui depuis un ficher grid dans le même format, nous générons une image en .jpeg, pour rendre la visualisation plus simple. Ainsi, la grille résolue dans l'exemple précédent générera l'image suivante :

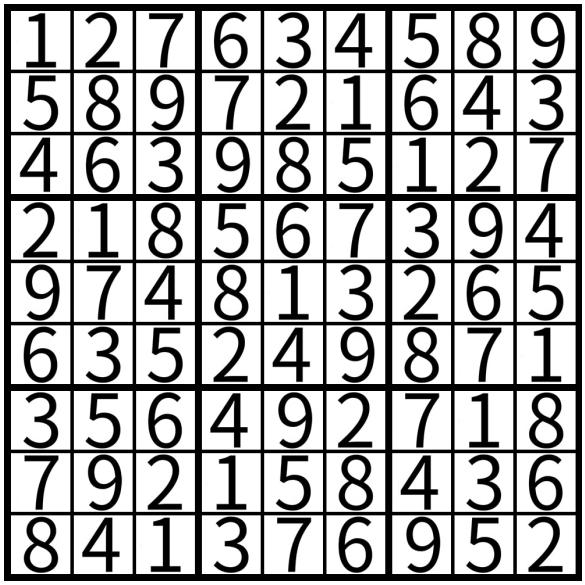


FIGURE 5 – Image du Sudoku résolu généré

Dans le futur, au lieu de générer une image à partir de rien, le but est de placer les numéros directement sur l'image d'origine.

3.7 Interface graphique

(Alexandre et Dylan)

Pour réaliser l'interface graphique de l'utilisateur, nous avons décidé d'utiliser l'outil GTK3. L'interface graphique va nous permettre d'utiliser notre programme de façon plus agréable. En effet, nous nous adressons à des utilisateurs qui ne savent pas forcément utiliser un terminal et l'interface graphique est plus intuitive. Cette interface va permettant de : charger une image de sudoku, la lire, puis la sauvegarder une fois résolu avec des boutons et des champs de textes. Pour le moment, cette partie n'a pas encore été commencée car elle n'est pas essentielle pour le fonctionnement de notre logiciel, seuls des tests et des maquettes ont été fait qui seront utiles pour le logiciel final. Vous pouvez retrouver ci-dessous une maquette de notre future interface graphique.

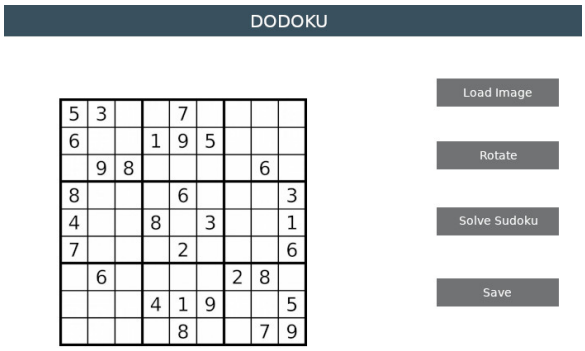


FIGURE 6 – Maquette de l'Interface Graphique

4 Avancement

Tâches	Première Soutenance	Soutenance Finale
Pré-traitements et Rotation	75%	100%
Détection de la grille et cases	35%	100%
Réseau de neurones	65%	100%
Sauvegarde des données	65%	100%
Résolution du Sudoku	100%	100%
Interface graphique	5%	100%

TABLE 2 – Tableau du niveau d'avancement (en %)

5 Conclusion

En conclusion, notre projet a bien avancé, nous somme aux environs de la moitié du développement de notre logiciel. Nous avons commencé toutes les tâches, et bien que certaines soient encore en phase de développement, nous avons la résolution du sudoku qui est complètement fonctionnel et les pré traitement qui sont bien avancés et un prototype de réseau de neurone plus que prometteur.