

Rapport de Projet

Projet Dodoku

Valentin UHLRICH Dylan INNOU
Nicolas PREVOST Alexandre BAO

Décembre 2021

EPITA | Promotion 2025



Table des matières

1 Introduction	5
1.1 Présentation de l'équipe	5
2 Répartition des tâches	7
3 Conceptions	8
3.1 Prétraitement	8
3.1.1 Grayscale	8
3.1.2 Rotation	8
3.1.3 Binarisation	9
3.1.4 Canny Edge Detection	9
3.2 Détection de la grille et des cases	11
3.2.1 Détection de la grille	12
3.2.2 Détection de l'angle	14
3.2.3 Detection d'un rectangle	15
3.3 Découpe des images	16
3.3.1 cutter	16
3.3.2 redimension des images	16
3.4 Réseau de neurones	18
3.4.1 Fonctionnement	18
3.4.2 Implémentation	18
3.4.3 Reconnaissance de la porte XOR	19
3.4.4 Reconnaissance des chiffres	19
3.5 Sauvegarde des données	20
3.5.1 Sauvegarde	21
3.5.2 Chargement	21
3.5.3 Base de données	21
3.6 Recréation de l'image	22
3.6.1 Recréation depuis le .sudoku seul	22
3.7 L'interface graphique	23
3.7.1 Charger une image de sudoku	23
3.7.2 L'interface pour les filtres	24
3.7.3 L'interface pour la rotation de l'image	25
3.7.4 L'interface pour le découpage de l'image	27
3.7.5 L'interface pour le réseau de neurone	28
3.7.6 L'interface pour la résolution du sudoku	30
3.7.7 Autre	31
4 Avancement	32
5 Conclusion	33

1 Introduction

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

Durant ce premier projet en langage C nous avons appris son fonctionnement. Il nous a fallu nous familiariser avec les pointeurs que nous avons beaucoup utilisés tout le long du projet, et qui ont été une nouveauté pour nous. De plus, ce projet nous a permis de découvrir plein de domaines à la fois. Nous avons du, pour la plupart, découvrir et comprendre le fonctionnement des algorithmes à utiliser. Ensuite, il nous a fallu 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 meilleure façon à 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 neurones 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 sa son interface, mais aussi d'aider mes collègues à la réalisation de ce projet ambitieux.

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 la rotation de celle-ci. J'épaulerai également mes compagnons pour leurs tâches 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 le 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 que chacun puisse choisir les parties sur lesquels il voulait travailler. Chaque responsable de Tâches à dû superviser sa partie en demandant aux autres de l'aider s'il avait besoin étant donnée que nous sommes un groupe. Les suppléants étaient là pour aider le responsable en cas de besoin.

3 Conceptions

3.1 Prétraitemet

Les filtres sont ici pour rendre l'image utilisable par les procédés qui vont suivre. Le but n'est pas nécessairement de rendre l'image plus belle pour l'oeil humain, mais plutôt de rendre plus facile l'extraction des informations.

Afin de permettre un meilleur traitement des informations il est nécessaire de comprendre de quoi on a besoin pour résoudre un sudoku à l'aide d'une image :

Arriver à discerner des lignes droites pour la grille de notre sudoku, donc savoir discerner des lignes de rien. Cela peut-être obtenue en mettant l'image en noir et blanc.

Savoir lire des différents nombres de notre image, donc avoir les nombres droits.

Tout ces premiers objectifs seront réalisable dans la partie pré-traitement que l'on va traiter ci dessous.

3.1.1 Grayscale

La première étape est de rendre l'image en échelle de gris. En effet pour le traitement, il n'est pas utile de garder les trois composantes (R, G, B) pour traiter une image. En effet, en plus de rendre l'image plus rapide à traiter car il n'y a qu'une donnée à manipuler, garder les 3 paramètres sont inutiles, on cherche à garder les caractéristiques de l'image et pas l'image entière.

La fonction grayscale mets alors le rouge, le vert et le bleu dans le rouge. Le rouge de chaque pixel contient alors la moyenne des 3 valeurs.

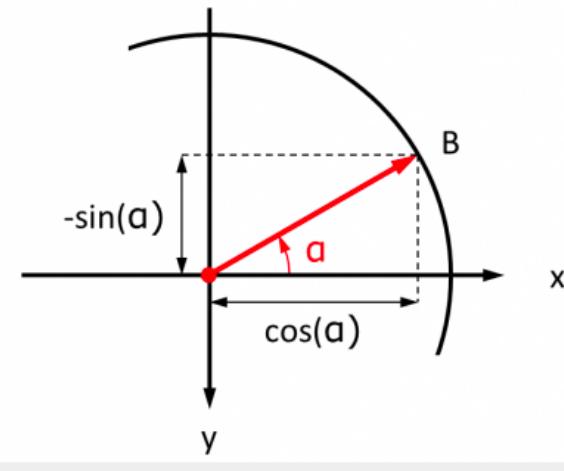
$$\text{pixel.r} = (\text{pixel.r} + \text{pixel.g} + \text{pixel.b})/3$$

On ne garde que la moyenne algébrique !

3.1.2 Rotation

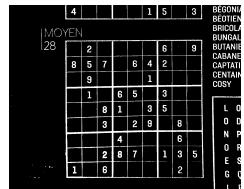
La rotation permet de garder à l'image un angle constant. En effet le réseau est entraîné à reconnaître des chiffres dans le sens vertical. Mettre des images qui ont un angle risque de fausser les résultats. Ici dans ce filtre, nous utilisons un angle en radian pour faire tourner l'image de l'angle donné. Pour chaque pixel, nous calculons sa nouvelle position selon sa distance au centre et son angle. Évidemment cette fonction fait appel à de la trigonométrie pour calculer le nouveaux pixel.

Nous verrons plus tard comment rendre la rotation automatique. Pour autant, cette rotation manuelle est utilisée par la rotation automatique. f

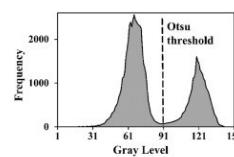


3.1.3 Binarisation

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 individuelles des pixels. Chacune des sommes individuelles va être placées dans un histogramme. On s'aperçoit vite (cf. la figure ci-dessous) que des groupements 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 histogramme

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 Canny Edge Detection

Le filtre de Canny est un filtre extrêmement connu au sein des programme informatique de traitement d'image.

Son but ? Obtenir une image avec que les contours de cette dernière. Cette fonction est extrêmement utile, car non seulement elle permet de bien mettre en évidence les lignes droites d'une image, elle permet également d'affiner ces dernières ce qui permet de ne pas détecter plusieurs lignes là où il n'y en a qu'une seule.

Nous avons décidé de ne pas inclure le filtre de Canny à l'intérieur de notre projet nous apercevant que son intégration pourrait être extrême-

ment difficile et chronophage. Ceppendant, nous avions voulu que même reprendre plusieurs parties de son algorithme qui pourrait nous être utile pour la suite de notre projet.

3.1.4.1 Filtre Gaussien

Le filtre Gaussien à pour but de lisser l'image en la rendant au passage plus floue. L'image va perdre en qualité mais ce n'est pas grave car elle sera binarisé plus tard, ce qui va rendre la résolution a l'image. Le filtre gaussien peut prendre plusieurs tailles de filtres. Pour le Canny, le filtre est un filtre passe-bas de taille 9 par 9.

1	4	7	4	1
4	20	33	20	4
7	33	55	33	7
4	20	33	20	4
1	4	7	4	1

$\frac{1}{331} \times$

FIGURE 2 – Schéma du filtre passe bas 9*9

Le filtre est fait de telle façon que plus un pixel est loin du noyau du filtre, moins il est pris en compte dans la valeur du nouveau pixel noyau. Pour chaque pixel, le filtre est applique sur ses voisins pour obtenir l'image floutée. Pour obtenir une image plus floutées, il suffit d'appliquer le filtre à de multiple reprises.

3.1.4.2 Edge Map

Comme dit précédemment, le filtre de Canny a l'avantage d'avoir une fonction qui permet d'obtenir les contours d'une image de manière extrêmement précise.

Cependant une fonction de ce type pour détecter un sudoku serait trop complexes pour une tâche aussi simple. C'est pourquoi on a décidé d'utiliser une simple méthode edgemap.

Cette dernière consiste, (cf. image n°3) de laisser blanc un pixel blanc, de laisser noir un pixel noir si et seulement si ce dernier est au moins à l'extrémité d'un pixel blanc et mettre .

En utilisant cette méthode avec une image avec laquel notre filtre de binarisation a du mal, on obtient alors cette image (cf. image n°4)

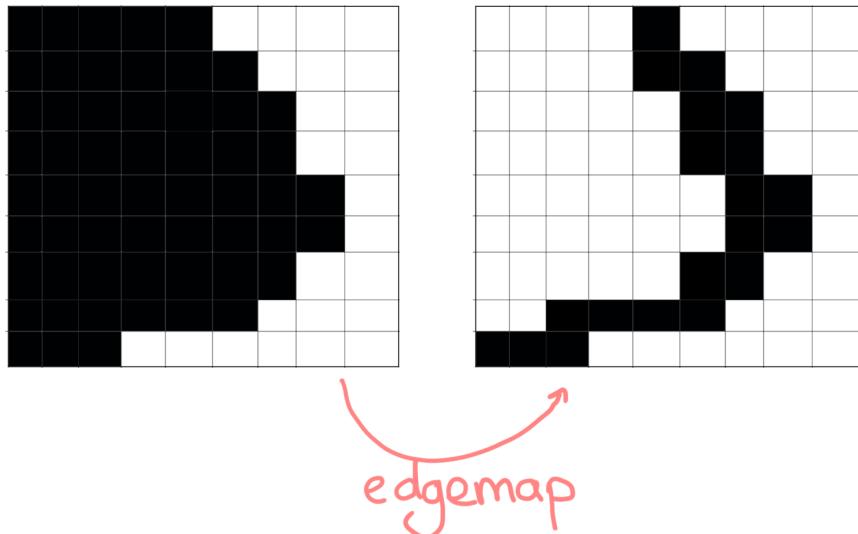


FIGURE 3 – Comment le filtre EdgeMap fonctionne

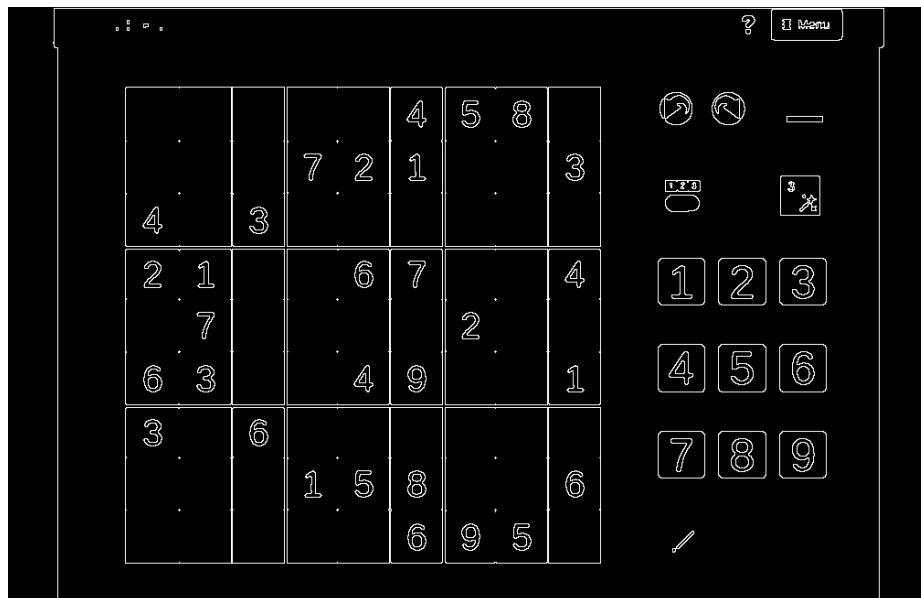


FIGURE 4 – Image 3 avec le filtre edgemap appliqué

On peut maintenant aisément voir les lignes du sudoku ce qui pourrait être très utile pour nos fonctions futurs.

3.2 Détection de la grille et des cases

(Dylan et Nicolas)

Dès lors qu'on a pu effectuer tous les traitements on peut commencer à détecter la grille de sudoku.

Notre objectif dans cette partie est de pouvoir automatiquement détecter,

découper et de dimensionner une image. On cherchera à ne rien obtenir autre que la grille de notre sudoku, ainsi retirer tous les contenus parasites qui peut se trouver à l'extérieur de notre sudoku.

Enfin dès qu'on pourra avoir notre grille seulement, il ne restera plus qu'à la découper selon les points d'intersection de notre image.

3.2.1 Détection de la grille

La première étape pour pouvoir découper notre image est de détecter notre grille. Comme durant la première soutenance, nous avions voulu développer notre transformée de Hough afin qu'elle puisse nous rendre seulement la grille de sudoku.

Rappelons brièvement comment la transformée de Hough fonctionne : 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 5 – 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.

Pour obtenir les fonctions qui le plus de points en commun, on utilisera le seuil suivant :

$$\text{Houghthreshold} = \text{maxline}/4$$

On prendra donc les lignes qui seront au minimum plus grande que le 1/4 de la plus grande. Ce choix a été fait car il vérifie la conditions sur toutes les images. Voici un exemple de à quoi ressemble la fonction Hough sur une grille données :

On voit bien qu'on arrive à distinguer les droites qui sont horizontales et celles qui sont verticales.

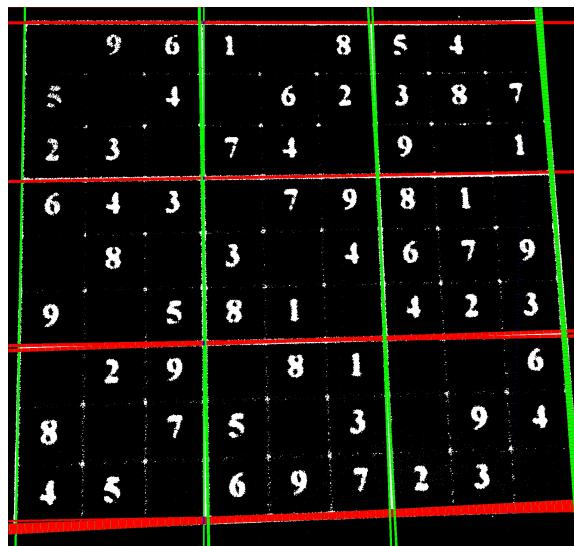


FIGURE 6 – Aperçu de la Transformée de Hough

La transformée de Hough possède plusieurs applications au sein du projet. Elle peut être utilisée dans la rotation automatique de l'image. Pour ce faire, il faut qu'on puisse détecter toutes les lignes de l'image, savoir différentier les lignes verticales et horizontales.

Mais un problème occure lorsqu'on utilise les images avec seulement le filtre d'Otsu appliqué. (cf. image n°7)

En effet, comme on peut le voir avec cette image. Plusieurs lignes ont été détectées car en effet dans un carré passe plusieurs lignes. Et cela biaise nos résultats, car le seuil de Hough sera plus élevé et ne prendra pas d'autres lignes qui pourraient être intéressante. Comme vous pouvez le voir avec l'image ci-dessus.

Comme dit précédemment, pour régler facilement à ce problème, on utilise la fonction edgemap et maintenant, comme vous pouvez le voir dans l'image n°9

Cependant, malgré toutes les lignes qu'on a réussi à enlever depuis qu'on a rajouté le filtre edgemap. Il reste que même beaucoup plus d'autres lignes à traiter et cela pourrait ralentir notre programme informatique que de devoir régler toutes ces lignes. Ainsi, on utilise une fonction qui permet de supprimer les occurrences de même droite, ou de droite ayant des composantes similaires. $\rho = x * \cos(\theta) + y * \sin(\theta)$

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

Comme les fonctions sont des fonctions affine chacune, on peut utiliser des simples méthodes d'algèbres classiques pour résoudre les équations. On cherche un x et un y , tel que (x,y) est un point d'intersection de lignes droites. Soit deux droites : $f(x), g(x)$ tel que $f(x) = g(x)$
Donc on cherche : $f(x) - g(x) = 0$. Cela correspond à la simple recherche du x , puis de l'utiliser dans l'une des deux fonctions.

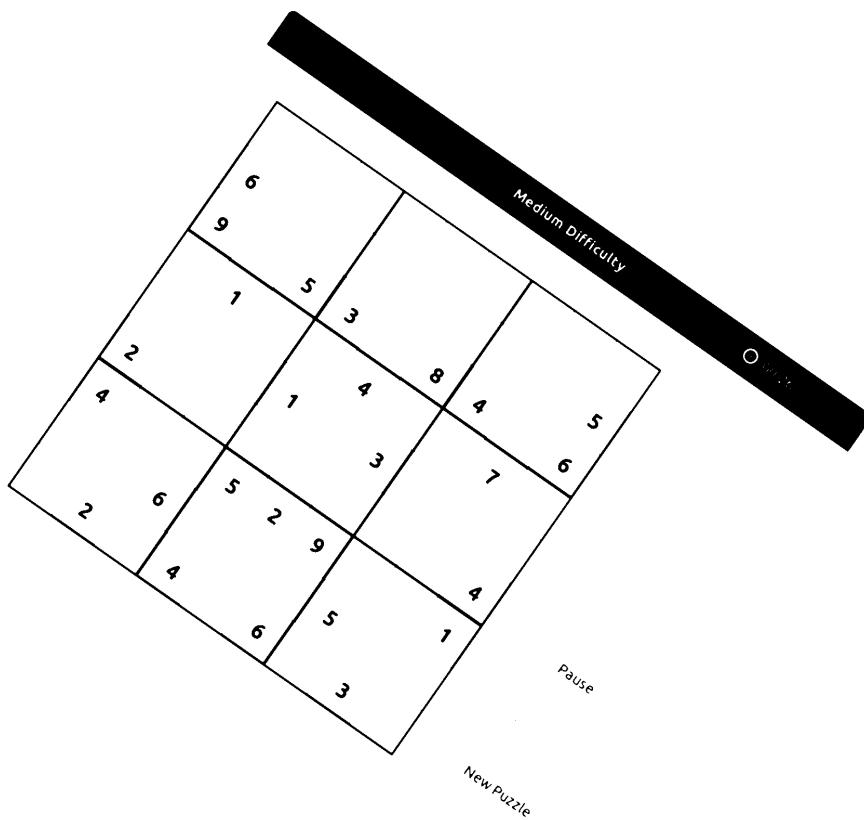


FIGURE 7 – L’Image 5 avec seulement le filtre d’Otsu en pré-traitement

De plus on cherche à

3.2.2 Détection de l'angle

Pour effectuer une rotation automatique de l'image, il faut d'abord détecter si l'image qu'on manipule est penché ou non. Pour savoir cela il y a une donnée très utile, ce sont les droites de la grille de sudoku.

En effet, grâce à l'utilisation des coordonnées polaires, on peut avoir l'angle dans lequel une droite est penchée par rapport à l'origine.

A l'aide de Hough, on peut récupérer l'angle d'une droite faisant partie de la grille de sodoku avec l'angle le moins prononcées.

Imaginons qu'on utilise une image I et que la transformée de Hough observe un angle α avec une des lignes du tableau. Alors on cherchera à tourner l'image de $-\alpha$, pour que l'angle de la droite que l'on a observer soit maintenant nul.

Par exemple, sur l'image ci dessus, la transforme de Hough va détecer une des droites les moins prononcées penchée de -35° . A l'aide de notre interface graphique, on va alors appliquer à cette image la rotation qui compense cet angle pour le rendre nul, c'est à dire que l'image soi hori-

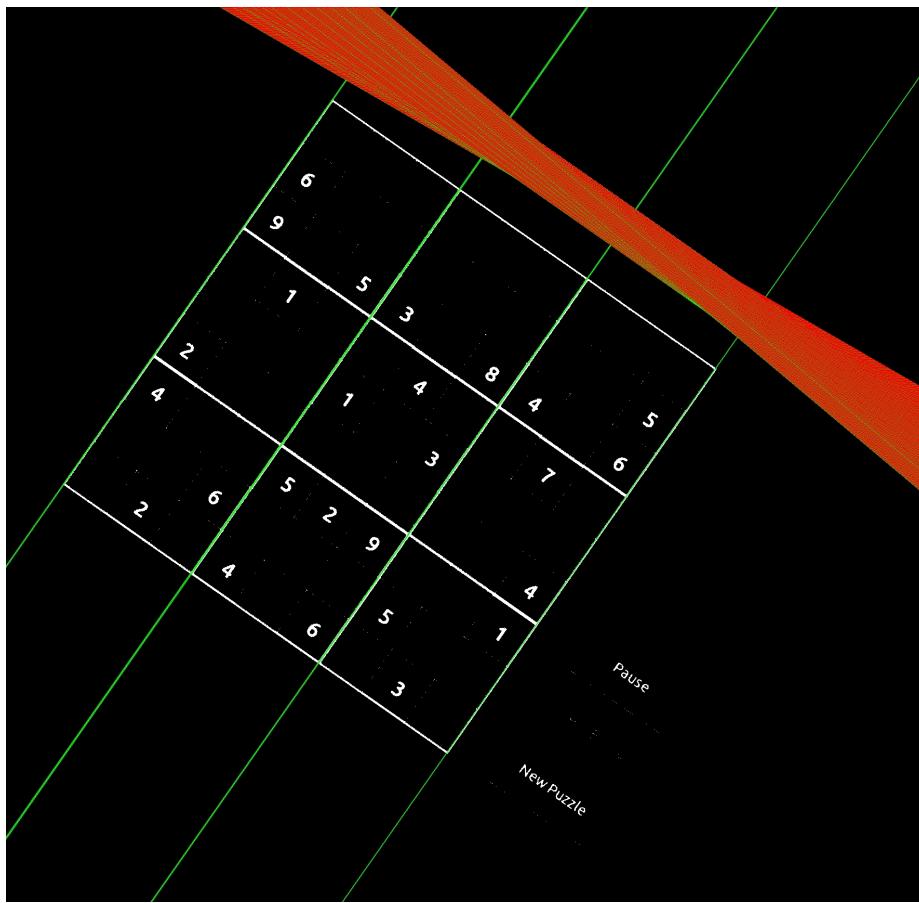


FIGURE 8 – Les lignes détectée de Hough de n°7 avec seulement le filtre d’Otsu en pré-traitement

zontale et dans le bon sens pour l’utilisateur et le réseau. On obtient grâce à la rotation automatique l’image suivante :

On observe que la rotation s’est bien passée, les lignes sont cohérentes et sont toutes soit horizontales, soit verticales, signe que la grille est dans le bon sens, et les chiffres également.

La qualité des résultats obtenus par la rotation automatique dépend alors de la précision de Hough sur cette image. C’est pour cela que cette rotation automatique s’effectue sur une image déjà bien traitée.

3.2.3 Détection d’un rectangle

A l’aide de la transformée de Hough, nous pouvons détecter plusieurs formes dont une forme qui nous sera très utile : le rectangle. En effet dans chacune des images on cherche à trouver la plus grande grille, ainsi en trouvant le rectangle le plus grand, on trouvera la grille de sudoku.

Nous nous sommes inspiré du travail de Claudio Rosito Jung and Rodrigo Schramm sur la détection de rectangle d’une image à l’aide de la transformée Hough.

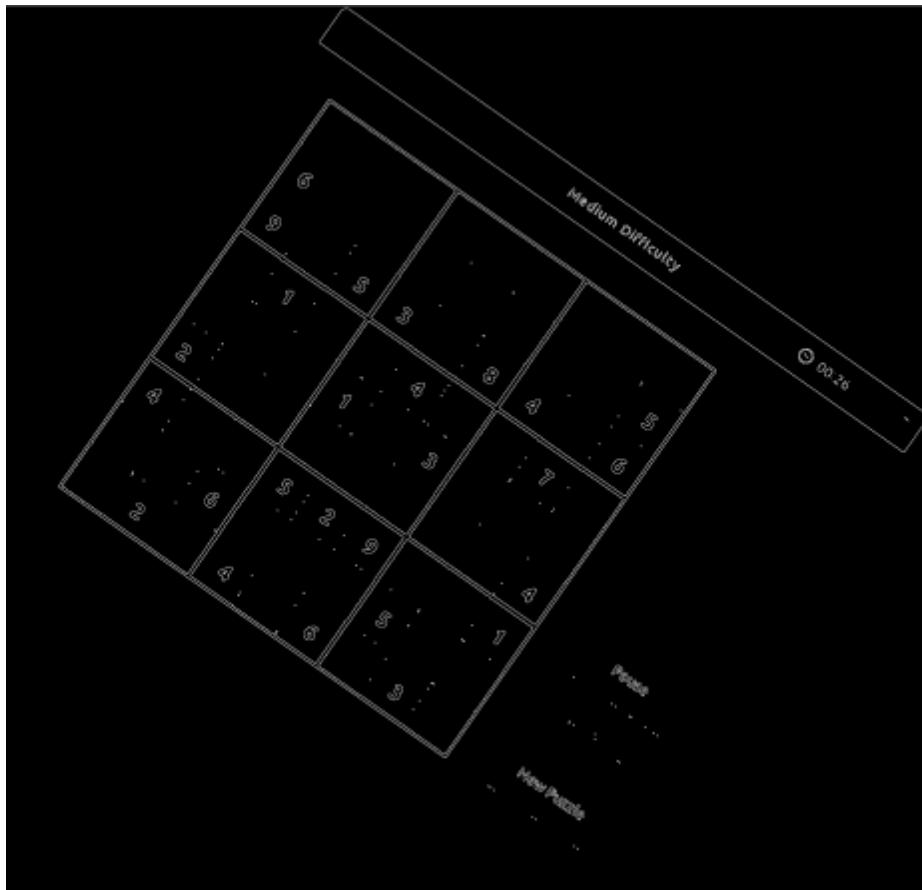


FIGURE 9 – image après un simple pré-traitement automatique

3.3 Découpe des images

Une fois que l'image est traitée, c'est à dire que le sudoku soit détecté et que les images des chiffres soient prêtes à être utilisées, il faut maintenant isoler les images. Le but est de prendre les images de chiffre et de les donner individuellement dans le réseau de neurone.

3.3.1 cutter

Le cutter est une fonction qui à partir de coordonnées de deux points, va extraire l'image qui se trouve dans le rectangle formés par ceux-ci. Le but étant d'isoler chaque

3.3.2 redimension des images

Une dernière étape avant la reconnaissance des chiffres est encore nécessaire. Le réseau de neurone a été entraînée pour des images de taille 28 par 28 pixel. Nous avons alors mis en place une fonction qui dimensionne toutes les images à ces dimensions. De plus, avoir des images de taille réduite permet également de réduire le temps d'entraînement du réseau mais également de rendre la reconnaissance de chacune des 81 images plus rapide.

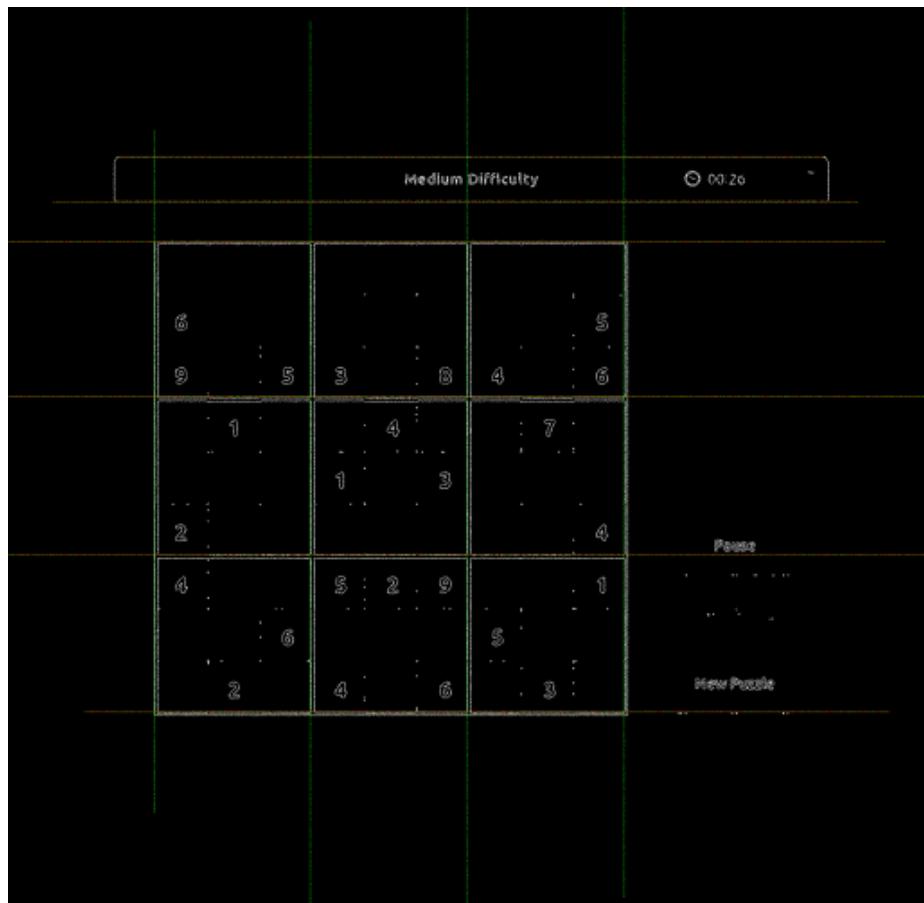


FIGURE 10 – image précédente après la rotation automatique

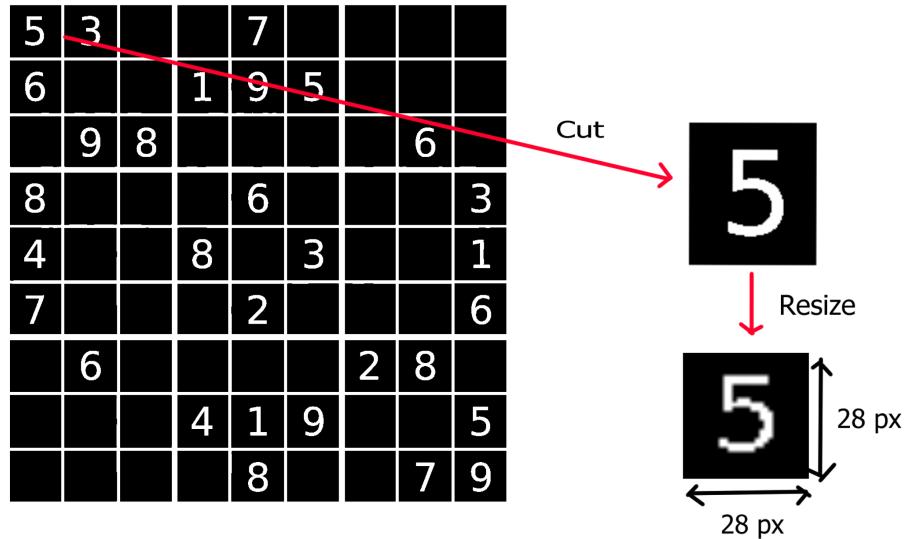


FIGURE 11 – exemple de l'extraction du chiffre 5

On répète ce procédé pour toutes les cases puis le réseau de neurone se charge de la reconnaissance de ceux-ci.

3.4 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.

3.4.1 Fonctionnement

Le principe du réseau de neurones est de donner en entrée 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 ses 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.

3.4.2 Implémentation

Passons maintenant à 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 trouvé 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 entraîner notre réseau de neurones en propagant 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 contiennent 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.

3.4.3 Reconnaissance de la porte XOR

Pour la 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 était fonctionnel. Nous devons 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.

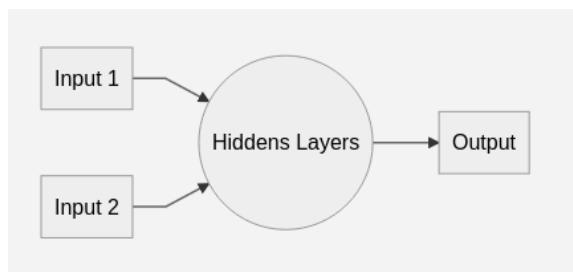


FIGURE 12 – Schéma du réseau pour le XOR

3.4.4 Reconnaissance des chiffres

Pour le projet final, il nous a fallu implémenter un réseau de neurones capables d'apprendre à reconnaître tous les chiffres. Nous avons repris le principe du XOR en le modifiant. Les images en entrée sont de taille

28x28, ainsi les neurones d'entrées seront tous les pixels de l'image qui vont correspondre à la valeur 1 si le pixel est blanc et a 0 si le pixel est noir. Ensuite nous avons choisi d'avoir une couche cachée et 15 neurones dans cette couche cachée. Ces valeurs ne sont pas choisies aléatoirement, elles sont connues pour être la meilleure configuration pour la reconnaissance de chiffre. Une fois passer dans tous les neurones nous avons 10 neurones de sorties qui correspondent aux valeurs de 0 à 9 (0 étant la case vide). Plus la valeur du neurone de sortie est grande plus cela veut dire que le neurone est sur de la valeur.

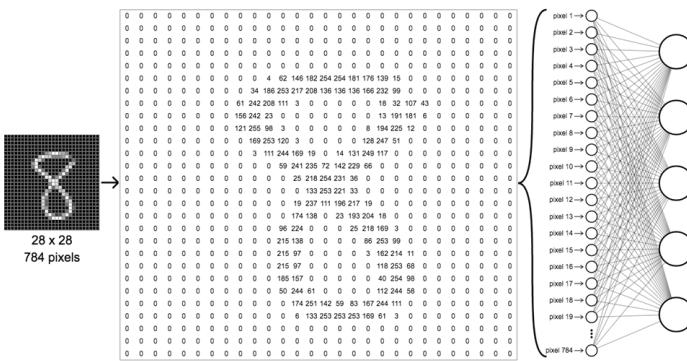


FIGURE 13 – Représentation du réseau de neurones

3.4.4.1 Entrainement

Dans ce cas nous allons appliquer une propagation arrière pour permettre de corriger les neurones suivant la valeur attendu pour cette image. Il faut savoir que nous disposons d'une dataset de 26k d'images fait à partir d'un script Python qui les à générer et stocker dans un fichier .dataset. Il faut savoir qu'il faut faire environ 100 interactions pour que le réseau de neurones commence à reconnaître les chiffres qu'on lui donne. Sur nos PC il nous faut environ 1h pour calculer des valeurs de ce réseau pour qu'il fonctionne correctement. Il faut donc en conclure que la sauvegarde dans un fichier des valeurs de neurones est obligatoire pour avoir un logiciel agréable à utiliser.

3.4.4.2 Utilisation

Lors de l'utilisation du réseau de neurones qui a déjà été entraîné au préalable. Il nous suffit juste de faire passer dans le réseau les pixels de l'image puis de prendre le neurone de sortie avec la plus grande valeur de sortie et ainsi ce neurone correspond au chiffre reconnu par l'IA.

3.5 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 réentraîner les neurones à chaque redémarrage du programme.

3.5.1 Sauvegarde

Pour sauvegarder les données de notre réseau nous allons utiliser des fichiers (au format .data) 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 :

```

1 nb_lignes nb_colonnes | nb_input nb_hiddens_layers nb_hiddens nb_output
2 poids_1_neurone_1_layer_1 poids_2_neurone_1_layer_1 ...
3 poids_1_neurone_2_layer_1 poids_2_neurone_2_layer_1 ...
4 ...
5 poids_1_neurone_1_layer_2 poids_2_neurone_1_layer_2 ...
6 ...

```

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.

3.5.2 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 à notre place. En effet, la conversion en `double` aurait été fastidieuse avec les virgules. Pour revenir à 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.

3.5.3 Base de données

Nous avons généré des images pour permettre d'entraîner nos neurones. Ces images ont été générées à l'aide d'un script Python qui à partir de plein de polices différentes génère une base de données d'images. Une fois les images générées elles sont stockées dans un seul fichier pour des raisons d'optimisation et de simplicité de déplacement. Pour s'entraîner le neurone récupère ligne par ligne l'image et ainsi il fait passer l'image dans son réseau. Il nous a fallu donc intégrer une fonction pour convertir une ligne de "01010011100..." en `SDL Image`.

3.6 Recréation de l'image

Une fois l'image traitée, les images séparées et reconnues par le réseau de neurone, il faut maintenant rendre l'image à l'utilisateur de façon la plus agréable possible. Il y a deux façons de rendre l'image, une image recréée à partir de rien mis à part du Sudoku, et une autre qui replace les numéros sur l'image au niveaux des vides.

3.6.1 Recréation depuis le .sudoku seul

Cette partie est la plus simple, car pour tout les sudokus, les cases seront toujours aux mêmes positions. De plus, la taille de sortie de l'image par cette méthode est fixe. Il suffit de "coller" les images aux bons endroits, sans même avoir besoin de changer la taille des chiffres. Il suffit alors de mettre toutes les images de chiffres aux bons endroit. En utilisant le sudoku suivant :

14 58.	127 634 589
2	... 721 ..3	589 721 643
3	4 .3	463 985 127
4		
5	21. .67 ..4	une fois resolu 218 567 394
6	.7. ... 2..	======> 974 813 265
7	63. .49 ..1	635 249 871
8		
9	3.6	356 492 718
10	... 158 ..6	792 158 436
116 95.	841 376 952

On obtient l'image suivante :

De plus, les chiffres issus de la résolution sont en rouge, pour aider à compétir la grille originelle. C'est la finalité du programme d'OCR, donc l'affichage doit être parfait et agréable pour l'utilisateur.

1	2	7	6	3	4	5	8	9
5	8	9	7	2	1	6	4	3
4	6	3	9	8	5	1	2	7
2	1	8	5	6	7	3	9	4
9	7	4	8	1	3	2	6	5
6	3	5	2	4	9	8	7	1
3	5	6	4	9	2	7	1	8
7	9	2	1	5	8	4	3	6
8	4	1	3	7	6	9	5	2

FIGURE 14 – gridresolved.jpeg

3.7 L’interface graphique

(Dylan et Alexandre)

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.

3.7.1 Charger une image de sudoku

Cette interface va permettre à l’utilisateur de charger une image de sudoku via un bouton qui va ouvrir l’explorateur de fichier. L’utilisateur n’aura plus qu’à sélectionner l’image de sudoku de son choix.

Une fois l’image chargé, l’utilisateur va avoir accès à plusieurs fonctionnalités qui sont disponible dans les différentes pages de l’interfaces, ces fonctionnalités sont l’application des filtres, la rotation de l’image, le découpage de l’image, le lancement d’un réseau de neurone et le résolveur de sudoku qui correspondent à ce qu’on a réalisé pour le projet.



FIGURE 15 – choix du sudoku

3.7.2 L'interface pour les filtres

Dans la page sur les filtres, l'utilisateur a le choix d'appliquer à l'image les filtres qu'il désire, plusieurs types de filtre sont disponible, il y a le niveau de gris, le flou, l'otsu et la carte de bord. Chaque filtre peut être activé ou régler grâce à des boutons, de plus, l'utilisateur peut également faire le choix d'activer le filtre automatique qui va calculer tout seul les valeurs optimales pour l'image sélectionnée.

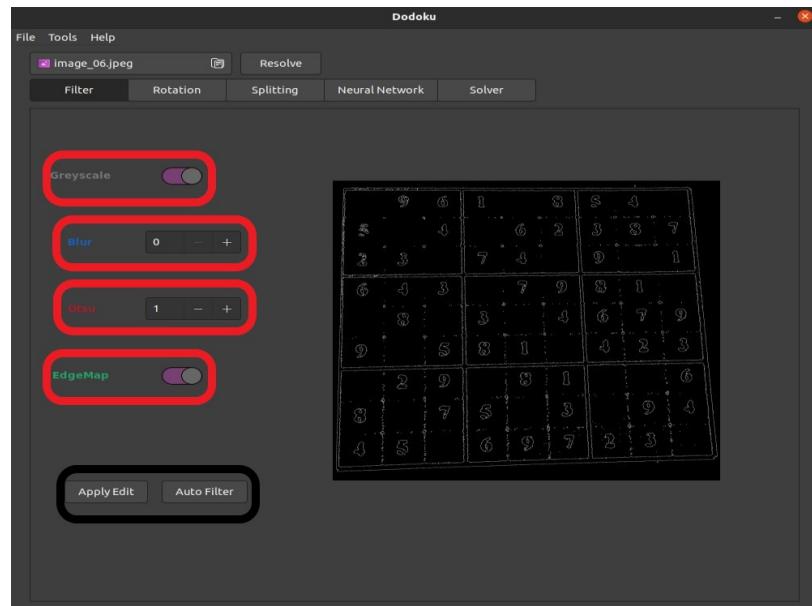


FIGURE 16 – Les filtres

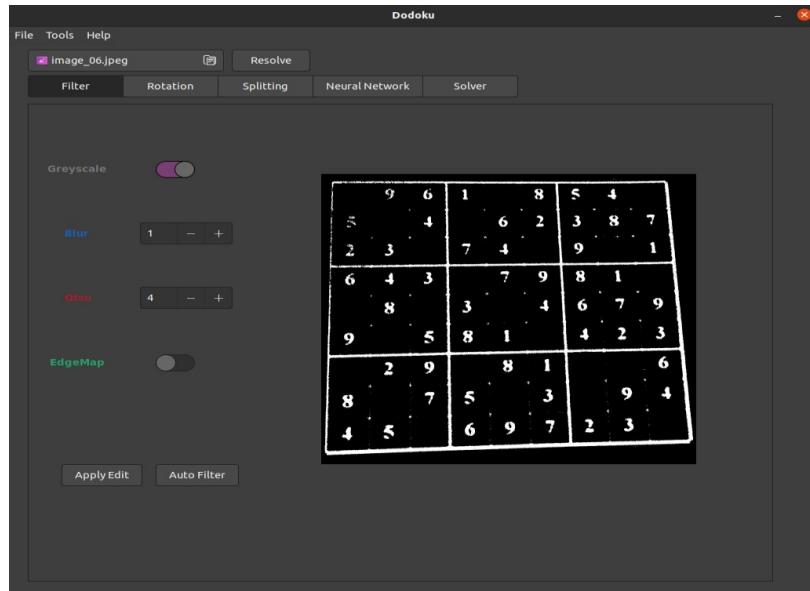


FIGURE 17 – Filtre automatique

3.7.3 L'interface pour la rotation de l'image

La page sur la rotation va permettre à l'utilisateur de d'appliquer une rotation à l'image avec un angle compris entre -180° et 180°, cet angle est réglable grâce à un bouton d'échelle situer en dessous de l'image, de plus, nous avons également inclus un bouton qui effectue une rotation automatique de l'image, en effet, les images que nous choisissons ne sont pas toujours bien orientés, de plus, une image bien orientée nous permettra d'avoir de meilleur résultat.

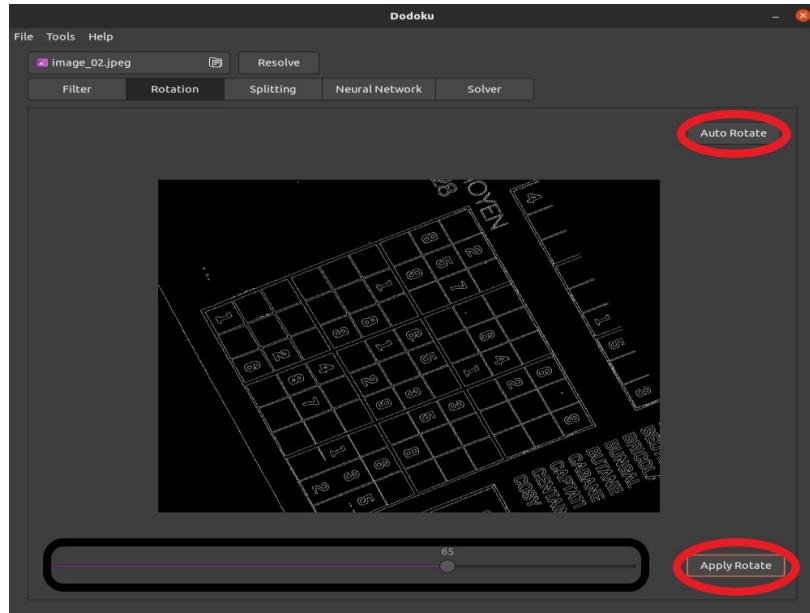


FIGURE 18 – Rotation de l'image

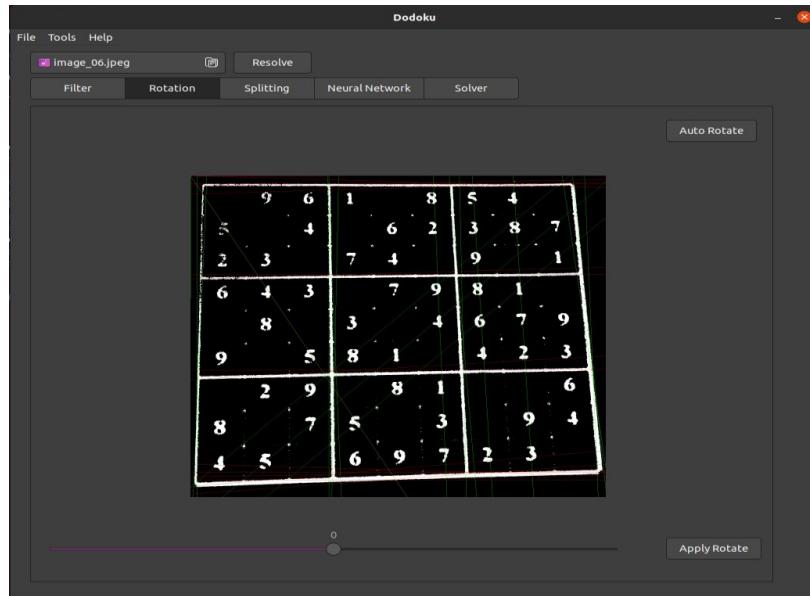


FIGURE 19 – Rotation automatique

3.7.4 L'interface pour le découpage de l'image

Pour le découpage de l'image, nous avons implémenter un bouton permettant de découper l'image en 81 images qui correspondent aux cases de sudoku. Il y a également des 4 boutons qui permettent de générer des traits de couleurs pour visualiser le découpage

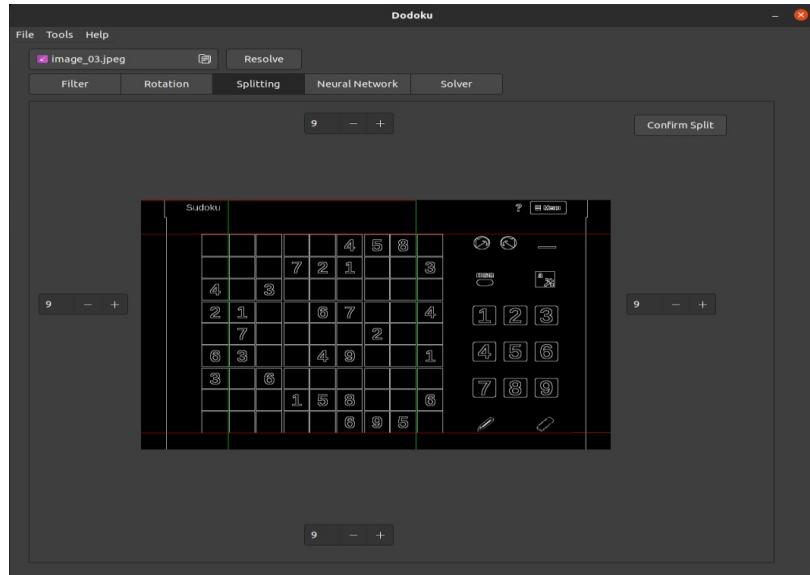


FIGURE 20 – découpage de l'image

3.7.5 L'interface pour le réseau de neurone

Concernant la page sur le réseau de neurone, nous avons créer une petite fenêtre appelé "configuration d'entraînement" accessible dans "outil" dans la barre de menus. Cette fenêtre permet de paramétriser le nombre d'entraînement que le réseau de neurone va effectuer et le sauvegarder dans un fichier data afin qu'il puisse être utilisable pour reconnaître les différents chiffres dans les cases du sudoku.

Pendant l'entraînement, nous avons également implémenté une affichage permettant à l'utilisateur de connaître le taux d'erreur et le nombre d'itération de l'entraînement pour que l'utilisateur puisse savoir s'il faut continuer à entraîner le réseau de neurone dans le cas où le taux d'erreur est trop élevé.

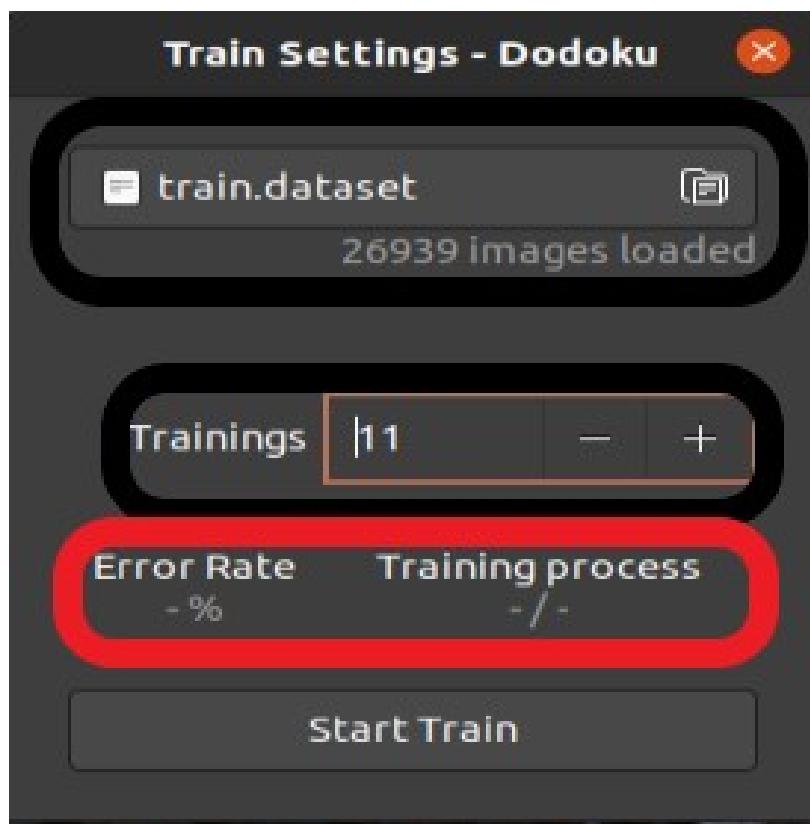


FIGURE 21 – Fenêtre d'entraînement du réseau de neurone

Une fois que l'entraînement est terminé, nous allons pouvoir charger notre réseau de neurone qui a été entraîner avec de préférence le moins de taux d'erreur possible dans la page correspondant au réseau de neurone et l'appliquer à notre image de sudoku qui a été découper par case pour reconnaître les chiffres de chaque case de la grille.

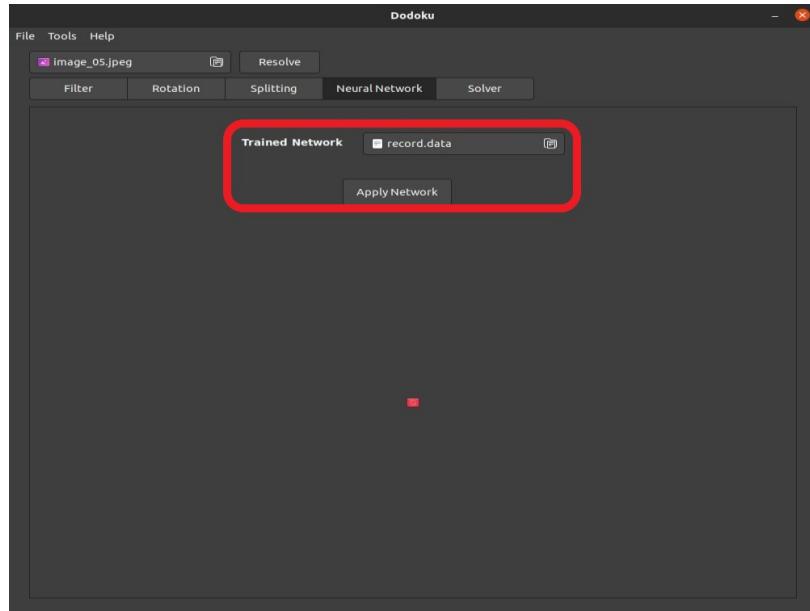


FIGURE 22 – Lancement du réseau de neurone

3.7.6 L'interface pour la résolution du sudoku

Et pour finir la page sur le résolveur va nous afficher une image du sudoku qui a été résolu par l'algorithme de résolution de sudoku qui utilise le backtracking. L'utilisateur a la possibilité de sauvegarder l'image dans l'emplacement de son choix.

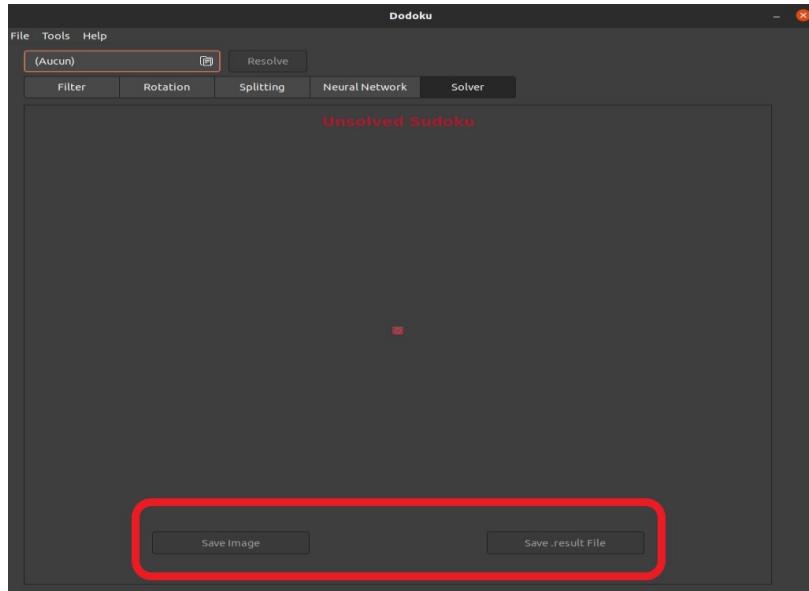


FIGURE 23 – Sauvegarde de l'image de sudoku

3.7.7 Autre

Nous avons aussi conçu une fenêtre supplémentaire accessible dans "à propos" dans la barre de menus qui affiche les membres du groupe ainsi que la version du projet et quelque information supplémentaire.



FIGURE 24 – à propos

Nous avons également remplacer le logo de l'application GTK glade par le logo de notre projet d'OCR.



FIGURE 25 – Logo 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, ce projet nous a ouvert l'esprit sur plusieurs domaines qui nous était pour ce jour inconnu. Nous avons dû faire par nous-mêmes un réseau de neurones, un solveur du sudoku, des détections de lignes grâce à des théorèmes. Cela nous sera très utile dans nos années futur. De plus, ce projet qui était en majeure partie composée des membres du projet de S2, a permis de consolider notre groupe et notre façon de travailler ensemble. Nous avons au début eu beaucoup de mal à trouver le temps de travailler ce projet en parallèle des cours, mais au fur et à mesure du semestre nous avons pu bien avancer. Notre projet contient donc une interface graphique réunissant tous les outils utiles pour partir d'une photo d'un sudoku à la grille de ce sudoku résolu. Tout en incluant un outil pour entraîner le réseau de neurones.

L'Equipe de Dodoku