OCR
Rapport final

Julie Fiadino, Nicolas Dek, Roshan Jeyakumar, Oscar Chevalier

Prepa S3 – Décembre 2022

Sommaire

1 Introduction **4**

1.1 Répartition 4

1.2 Architecture de projet 6

1.2.1 Architecture du programme 6

1.2.2 Architecture des fichiers 7

2 Guide Utilisateur **8**

2.1 Manipulation de fichiers 8

2.1.1 Sélectionner une image 8

2.1.2 Sauvegarder l'image 8

2.2 Rotation 9

2.2.1 Rotation Manuelle 9

2.3 Réseau de Neurones 10

2.3.1 Entraînement du réseau 10

2.3.2 Chargement des poids 10

2.4 Analyse 10

2.4.1 Analyse de la grille 10

2.4.2 Analyse étapes par étapes 11

2.5 Résolution 11

3 Aspects Techniques **12**

3.1 Solveur 12

3.1.1 Algorithme de retour sur trace 12
3.1.2 Utilisations 13
3.2 Traitement d'image 14
3.2.1 Noir et blanc 14
3.2.2 Ajustement d'image 16
3.2.3 Détection de la grille 21
3.2.4 Interpolation 27
3.3 Réseau de neurones 28
3.3.1 Le XOR 28
3.3.2 L'utilisation de matrices 28
3.3.3 Le réseau 28
3.3.4 La reconnaissance de chiffre 29
3.3.5 L'entraînement 29
3.4 Interface 31
3.4.1 Menus 31
3.4.2 Api GTK 32
3.4.3 Connexion au programme 33
3.5 Sauvegarde et chargement de fichiers 35
3.5.1 Sauvegarde de la grille 35
3.5.2 Chargement de la grille 35
3.5.3 Sauvegarde des poids du réseau de neurones 35
3.5.4 Chargement des poids du réseau de neurones 36

4 Environnement de Travail 37

4.1 Automatisation de tests 37
4.2 Intégration Continue 38

4.3 Makefile 39
4.3.1 Options de compilation 39
4.4 Ambiance du groupe 40

Introduction

Chez Julie & Co, notre but est de proposer le meilleur outil de résolution de sudoku qui soit.
Notre équipe est constitué de Julie Fiadino, Nicolas Dek, Roshan Jeyakumar et Oscar Chevalier.

Répartition

Les 4 grandes parties du projet ont été réparties parmi chacun des membres. Cette répartition n'est pas non plus fixe car chacun peut aider une section en difficulté. Elle permet surtout d'associer un expert à chacune des sections qui sera capable d'en connaître l'avancement et l'aspect technique de manière précise.

- Julie Fiadino : En charge de l'interface graphique (cheffe de projet)
- Oscar Chevalier : en charge de l'exécutable 'solver'
- Nicolas Dek : en charge du traitement d'image
- Roshan Jeyakumar : en charge du réseau de neurones

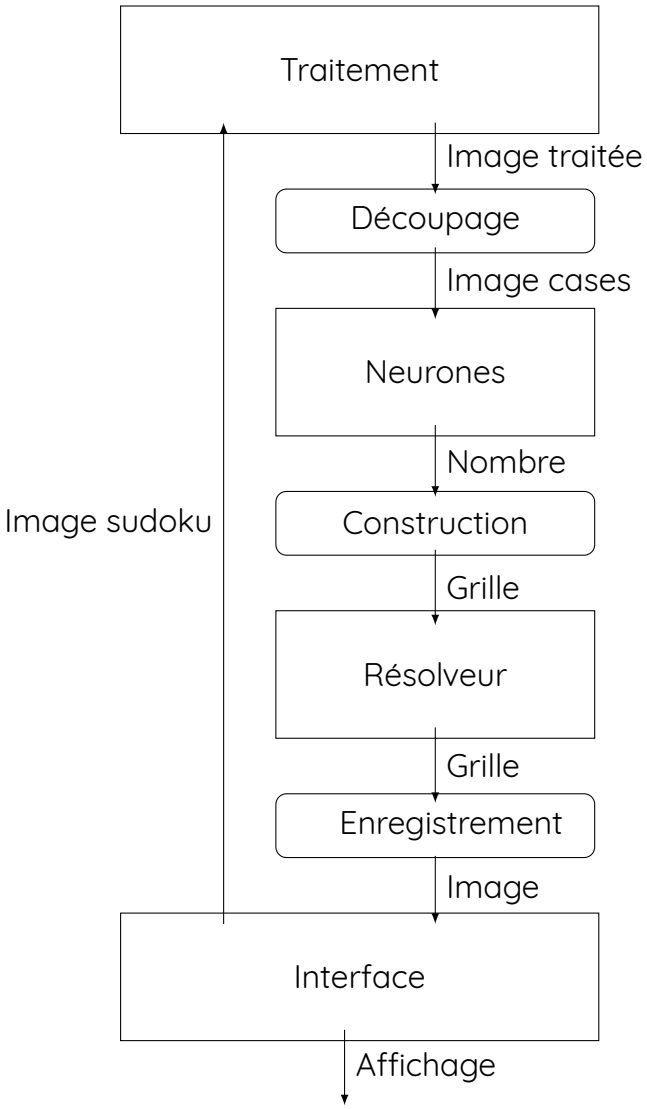
Pour cette seconde soutenance, notre attention s'est concentrée sur l'interface graphique, ainsi que résoudre les problèmes du traitement de l'image soulevés lors de la précédente soutenance.
Voici donc la répartition des tâches de cette fin de semestre.

Tâche	Répartition
Résolveur (algo)	Oscar Chevalier
Résolveur (executable)	Oscar Chevalier
Traitement d'image (luminosité)	Julie Fiadino
Traitement d'image (détection de composants)	Nicolas Dek
Traitement d'image (recadrage)	Nicolas Dek
Interface Graphique (traitement et réseau de neurones)	Julie Fiadino
Interface Graphique (rotation et résolveur)	Oscar Chevalier
Traitement d'image (otsu)	Nicolas Dek
Traitement d'image (détection de grille)	Nicolas Dek
Traitement d'image (rotation automatique)	Julie Fiadino
Réseau de neurones (banque d'image)	Oscar Chevalier
Réseau de Neurones (reconnaissance d'images)	Roshan Jeyakumar

Architecture de projet

1.2.1 Architecture du programme

Le programme est structuré comme ceci :



Sur ce schéma, chaque bloc rectangulaire représente une section importante du programme. Les blocs arrondis représentent une étape de conversion.

Les flèches représentent le type de donnée entrant et sortant.

Ce diagramme correspond aux étapes que va suivre notre programme. Chacun des blocs importants étant indépendants (pour leur permettre d'être facilement modifiables) une étape de conversion est donc requise pour passer de l'un à l'autre.

L'une des notions importantes est le fonctionnement circulaire du programme, comme indiqué avec la flèche 'Image sudoku'. En effet, l'interface va fournir l'image du sudoku et afficher sa résolution.

1.2.2 Architecture des fichiers

Le projet suit les conventions de structure d'un programme. Il contient :

- Un dossier `include`: contenant tous les fichiers .h du projet,
- Un dossier `src`: contenant tous les fichiers .c du projet,
- Un dossier `img`: avec toutes les images utilisées par le projet (réseau de neurones),
- Un dossier `tests`: avec toutes les fonctions de tests du projet,
- Un Makefile: permettant de compiler les différentes versions du code.

Pour faire écho à l'architecture du programme, le code pour chacune des sections importantes est séparé dans son propre sous dossier, que ce soit dans `include` ou dans `src`.

Guide Utilisateur

Manipulation de fichiers

2.1.1 Sélectionner une image

Pour ouvrir un fichier il suffit de cliquer sur « File » puis « Open... ». Une fenêtre de sélection de fichier apparaît alors. Les fichiers peuvent être ouvert dans les formats PNG, JPG et BMP.

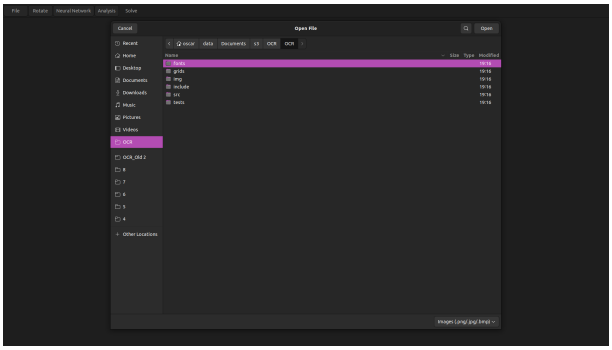


Figure 2.1: Fenêtre de chargement d'image

2.1.2 Sauvegarder l'image

Pour sauvegarder une image, il faut cliquer sur « File » puis « Save... ». Une fenêtre de sélection de fichier apparaît alors. Les fichiers sont sauvegardé dans le format PNG.

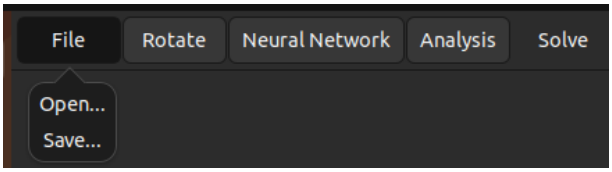


Figure 2.2: Menu « File »

Rotation

2.2.1 Rotation Manuelle

Pour faire une rotation manuelle, il faut ouvrir le menu « Rotate » et ouvrir le glisseur à droite de « Manual Rotation ».

La rotation manuelle permet de tourner l'image sur 360°.

Une image tournée peut être sauvegardée ainsi.

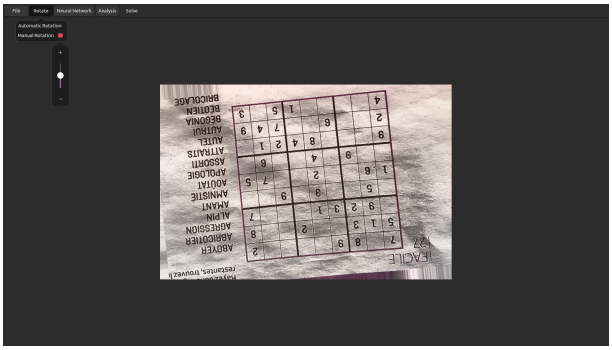


Figure 2.3: Rotation manuelle d'un sudoku

Réseau de Neurones

2.3.1 Entraînement du réseau

Pour démarrer l'entrainement du réseau de neurones, il faut aller dans le menu « Neural Network » et cliquer sur « Train ». Le programme va donc entrainer le réseau sur 10 000 générations.

2.3.2 Chargement des poids

Pour charger le réseau de neurones, il faut aller dans le menu « Neural Network » et cliquer sur « Load ».

Une fenêtre apparait donc pour permettre de choisir le fichier de sauvegarde à charger.

Analyse

2.4.1 Analyse de la grille

Pour analyser la grille, il faut aller dans le menu « Analysis » et cliquer sur « Analyse ».

Ce bouton fait toute les étapes de correction de l'image, détection des caractères et leur lecture.

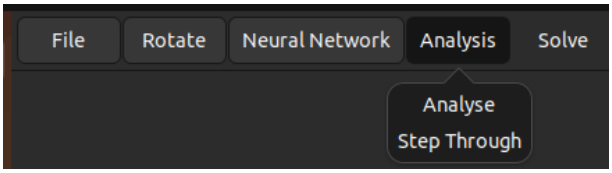


Figure 2.4: Menu de la résolution

2.4.2 Analyse étapes par étapes

Pour analyser étapes par étapes le sudoku, il faut cliquer sur « Analysis », puis sur « Step Through ».

Avec cette option, le logiciel s'arrêtera à chaque étape en indiquant la transformation faite.

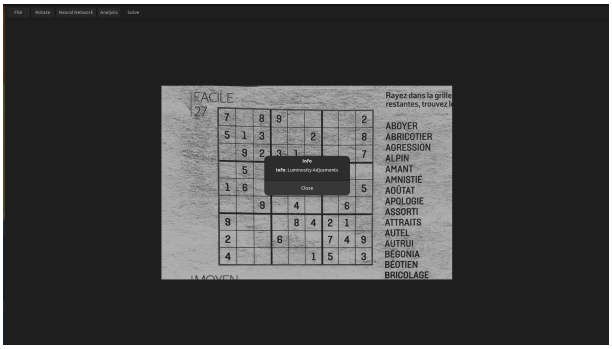


Figure 2.5: Exemple de résolution étapes par étapes d'un sudoku

Résolution

Pour résoudre la grille obtenue, il faut cliquer sur le bouton « Solve ».

Cela va remplir les cases vides par les chiffres calculés par le solveur. Les chiffres déjà présents sur la grille sont en noir et ceux affichés après la résolutions sont en bleu.

Aspects Techniques

Dans cette section, nous allons aborder tous les aspects techniques de la construction du programme.

Solveur

Le solveur (solver) utilise la méthode classique de résolution de sudokus de retour sur trace (backtracking).

3.1.1 Algorithme de retour sur trace

Cet algorithme consiste à tester récursivement si une proposition de solution est valide ou non.
Dans le cas du sudoku, il faut créer une fonction récursive qui prend en paramètre la grille.

Dans un premier temps, cette fonction attribue à la première case vide (d'abord de gauche à droite puis de haut en bas) toutes les valeurs entre 1 et 9 (inclus) les unes après les autres.

Une fois une valeur initialisée sur la case choisie, il faut vérifier que le sudoku est toujours correct. Pour cela il existe des fonctions vérifiant que la ligne, le carré et la colonne sont corrects.

Si le sudoku est valide, on appelle cette même fonction récursive.
Si sur un appel de la fonction récursive il ne reste plus aucune case vide, alors cela signifie que le sudoku est terminé et il renvoie la valeur 1 (vrai).

Si l'algorithme n'a pas réussi à trouver de solution, cela signifie que le sudoku n'est pas solvable, il renvoie alors la valeur 0 (faux).

Depuis la première soutenance, l'algorithme a été légèrement optimisé et la forme

général du solveur a été clarifié.
Notamment en enlevant une partie qui comptait et tenait le compte du nombre de cases encore vides.

3.1.2 Utilisations

Directe

Le solveur peut être utilisé avec des fichiers textes représentant la grille a résoudre. Il enregistre la grille une fois résolue dans un fichier avec les mêmes caractéristiques, uniquement si le sudoku est solvable.

Indirecte

Le solveur peut être utilisé par d'autres fonctions pour résoudre un sudoku sous la forme d'un tableau à 2 dimensions. Les cases vides sont représentées par les chiffres 0 ou -1.

Traitement d'image

Le traitement d'image est constitué de plusieurs étapes, allant de l'ajustement de luminosité à la rotation de l'image.

3.2.1 Noir et blanc

Grayscale

Avant tout traitement, l'image est convertie en grayscale. Pour calculer la valeur de chaque pixel, notre algorithme prend la valeur des couleurs (r, g, b) la plus claire et l'utilise pour définir le niveau de gris.

Le but de cette méthode est de filtrer les couleurs qui peuvent être perçues comme foncées par un grayscale 'normal' (avec calcul de moyenne).

Par exemple, le bleu de l'image 4 va être perçu plus clair à cause de sa faible teneur en rouge.

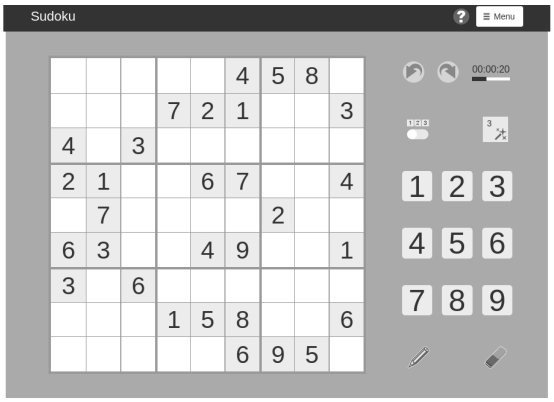


Figure 3.1: Image 4 avec grayscale 'normal'

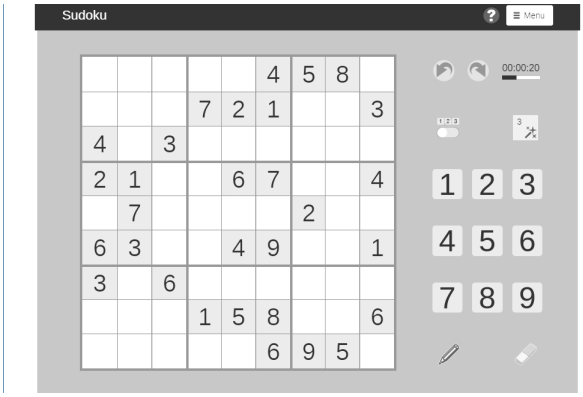


Figure 3.2: Image 4 avec grayscale maximal

Noir et Blanc

Pour la mise en noir et blanc, nous avons décidé d'utiliser l'algorithme de Otsu. Otsu a la particularité d'être plus efficace que les algorithme de noir et blanc basique, car il enlève en majorité les pixels parasite d'une image.

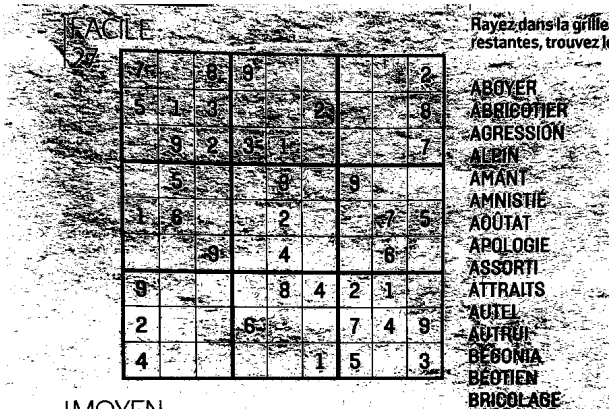


Figure 3.3: Algorithme basique de mise en noir et blanc

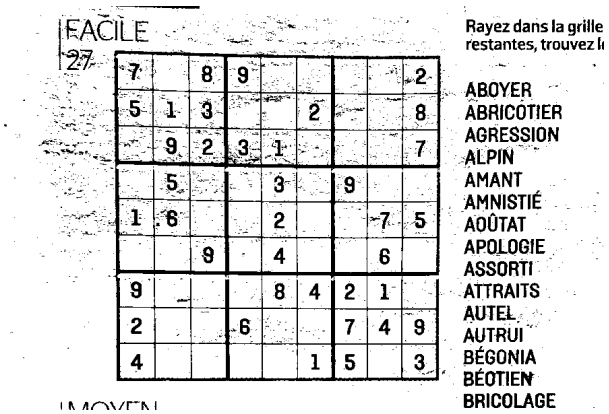


Figure 3.4: Algorithme d'Otsu

La première image a été réalisé avec l'algorithme que nous utilisions à l'origine, mais il ne facilite pas clairement la distinction de la grille dans l'image. C'est pour cette raison que nous avons adopté l'algorithme d'Otsu.

3.2.2 Ajustement d'image

Dilation et Erosion

Ces 2 opérations sont appelées 'opérations morphologiques'.

Elles prennent toutes les 2 un élément structurel qui va permettre de déterminer la nouvelle couleur des pixels de l'image.

Dans le cas de la dilation, on place l'élément structurel au centre de chacun des pixels, et on lui assigne la couleur la plus claire contenue dans l'élément.

Pour l'érosion, on y assigne la couleur la plus sombre.

Figure 3.5: Sudoku en grayscale

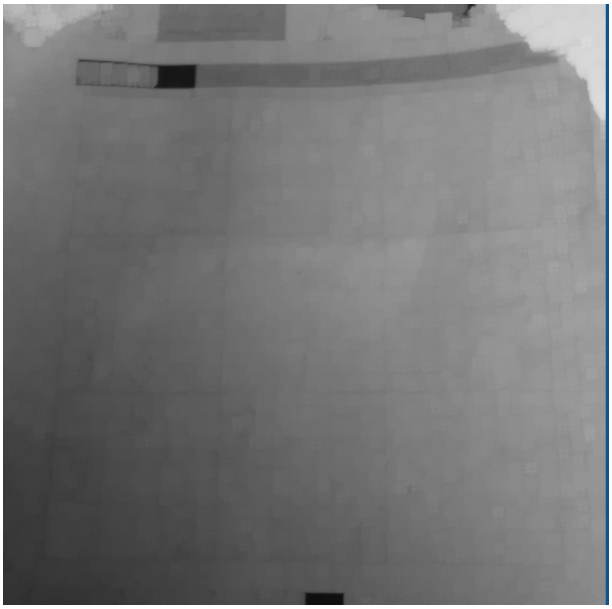


Figure 3.6: Dilation sur le sudoku

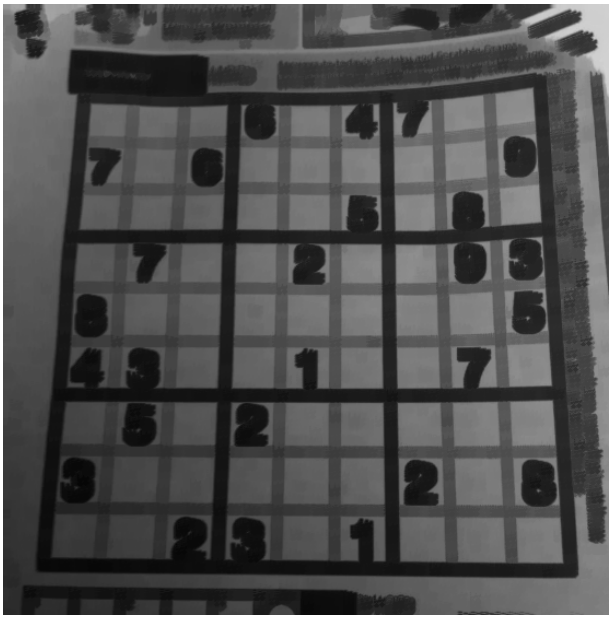


Figure 3.7: Erosion sur le sudoku

La dilation va donc permettre de faire ressortir le fond, et l'érosion les lignes noires.

Pour ajuster le niveau de luminosité de l'image, notre programme applique une 'closing operation'.

Cette opération va effectuer une dilation puis une érosion sur l'image.

Figure 3.8: Closing operation sur l'image 1

Le but de ces opérations va être d'obscurcir les lignes du sudoku.

Ajustement

En divisant l'image de base par celle obtenue, les tons trop sombres ou trop clairs vont donc s'annuler, ne laissant que les chiffres et les lignes de la grille visible.

Figure 3.9: Sudoku après ajustement

Cette technique est efficace car elle permet d'obtenir un résultat satisfaisant pour chaque type d'image. Cela évite donc de détecter la luminosité de l'image pour choisir le type d'ajustement.

De plus, elle permet d'ajuster des images avec un éclairage pas toujours uniforme. C'est par exemple le cas de l'image 7.

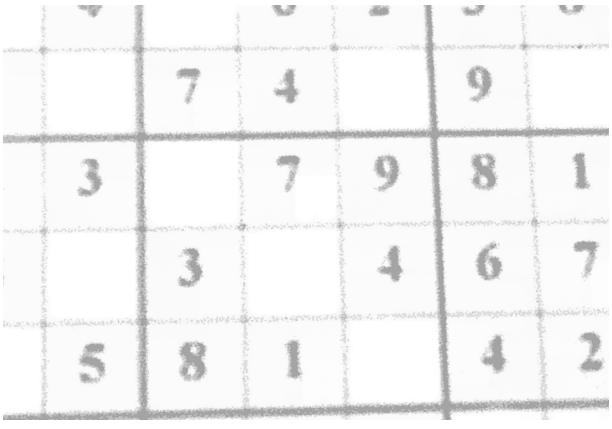


Figure 3.10: Image 7 après ajustement

Cette méthode est en revanche extrêmement coûteuse, avec $width^2/100 \times height$ opérations. Les images trop grandes doivent donc être rétrécies au préalable.

3.2.3 Détection de la grille

Détection de composants

Pour la détection des composants dans une image on utilise l'algorithme Connected-component labeling, cela va nous permettre de facilement identifier la grille du sudoku dans une image.

Pour faire fonctionner l'algorithme il nous faut une image binarisée (c'est à dire une image avec seulement 2 couleurs).

Voici le fonctionnement de l'algorithme:

On met le compteur de label a 1.

On parcours chaque pixel d'une image de gauche vers la droite et de haut en bas.

Si le pixel de gauche contient un label alors le pixel courant récupère son label.

Si le pixel de gauche n'a pas de label mais que le label de haut en a un, alors on récupère le label de celui du haut.

Sinon, si le pixel de gauche et de haut n'ont pas de label on donne au pixel courant un label et on incrémente le compteur de label.

Ensuite après avoir fini de parcourir l'image, on la parcourt une deuxième fois et cette fois pour chaque pixel on remplace son label par le plus petit label de ses voisins.

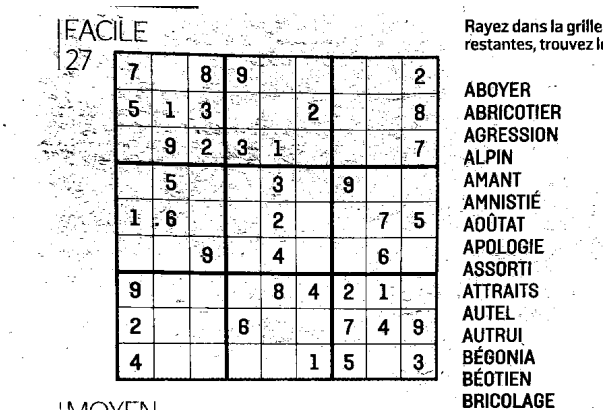


Figure 3.11: Image de base

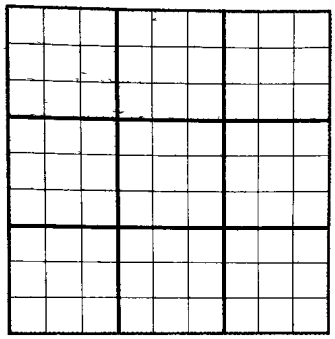


Figure 3.12: Image en ayant sélectionné le label le plus gros

En utilisant l'algorithme précédemment décrit, nous pouvons isoler la grille dans l'image, facilitant ainsi les opérations à appliquer sur celle-ci.

Rotation

Le calcul de la rotation se fait à l'aide d'une matrice de rotation.

Cette matrice permet d'obtenir les coordonnées suivantes : $x' = \sin(angle) * x + \cos(angle) * y$ et $y' = \cos(angle) * x - \sin(angle) * y$

Il suffit donc d'assigner la couleur actuelle aux nouvelles coordonnées.

Pour détecter l'angle de rotation, le programme utilise le point de la grille le plus proche des coordonnées (0,0) et celui le plus proche du centre du côté gauche. Il s'en sert pour calculer le vecteur parallèle au côté de la grille.

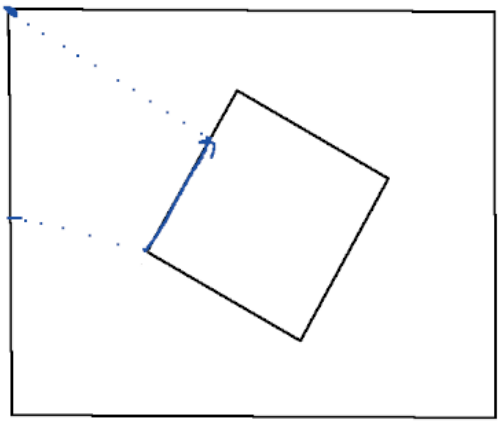


Figure 3.13: Calcul du vecteur parallèle au côté

On calcule ensuite le vecteur vertical qui part du point le plus bas et rejoint le haut de l'image.

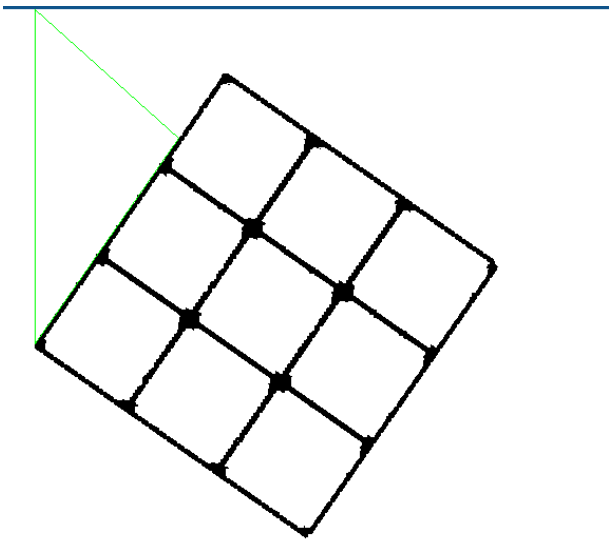


Figure 3.14: Calcul du vecteur vertical sur l'image 5

Une fois les 2 vecteurs obtenus, on utilise la formule suivante :

$$\alpha = arccos\left(\frac{parallele_x * vertical_x + parallele_y * vertical_y}{\|parallele\| * \|vertical\|}\right)$$

Cette formule est dérivée du produit scalaire défini ainsi : $u \cdot v = \|u\| * \|v\| * cos(\alpha)$

L'image étant définie comme un repère orthonormé, le produit scalaire peut être obtenu à l'aide d'une autre formule : $u \cdot v = u_x * v_x + u_y * v_y$

L'angle est alors obtenu en isolant le cosinus dans l'équation.

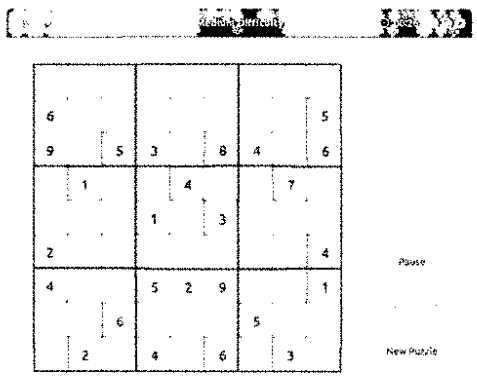


Figure 3.15: Image 5 après rotation

Récupération des lignes

Les lignes sont récupérées en dérivant l'image en x et en y.

Cette opération s'effectue grâce à une convolution, démarche visant à passer une matrice appelée 'kernel' sur chacun des pixels et à calculer la nouvelle couleur en fonction des poids inscrits.

Pour dériver l'image, le programme utilise un kernel de sobel :

$$K_{sobel} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

La dérivée en x de l'image revient donc à faire une convolution de l'image I par le kernel. Pour la dérivée en y, on utilise la transposée du kernel.

Cette matrice permet seulement de faire des dérivées de premier degré. Or, pour plus de précision, il est préférable de passer par une dérivée de second degré. La convolution étant associative, on peut donc faire une convolution sur le kernel avec le kernel.

On obtient alors le kernel suivant :

$$K_{sobel} * K_{sobel} = \begin{pmatrix} 1 & 0 & -2 & 0 & 1 \\ 4 & 0 & -8 & 0 & 4 \\ 6 & 0 & -12 & 0 & 6 \\ 4 & 0 & -8 & 0 & 4 \\ 1 & 0 & -2 & 0 & 1 \end{pmatrix}$$

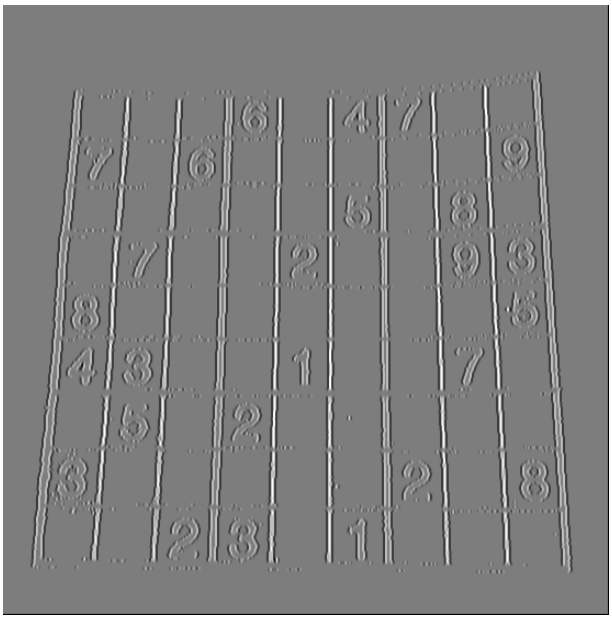


Figure 3.16: Dérivée en x du sudoku

Il suffit ensuite de binariser l'image et de lui appliquer une dilation pour rendre les lignes plus visibles.

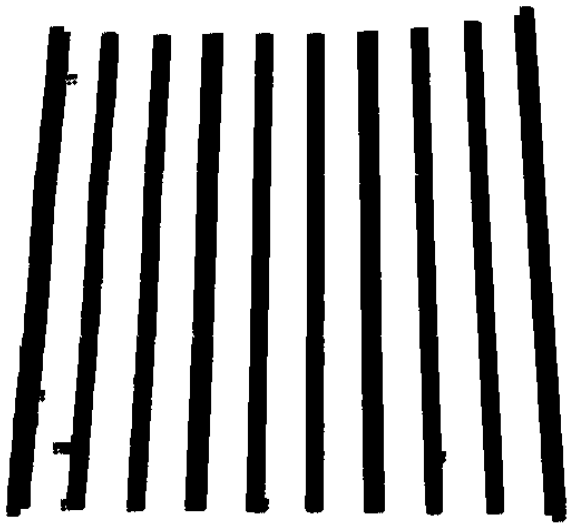


Figure 3.17: Dérivée en x après binarisation et dilation

En appliquant un ’et binaire’ entre les 2 dérivées, on obtient donc les intersections des lignes c’est-à-dire tous les coins des cases de la grille.

3.2.4 Interpolation

Après traitement, le sudoku est donc isolé et enregistré dans une nouvelle image de taille fixe qui sera ensuite découpée et envoyée au réseau de neurone.

Pour cela, le programme utilise de l’interpolation linéaire. L’image source est considérée comme une texture qui va être appliquée sur le carré que représente la nouvelle image.

Cette méthode va donc permettre d’étirer automatiquement l’image pour qu’elle rentre dans les dimensions demandées de manière uniforme.

Réseau de neurones

3.3.1 Le XOR

Pour la première soutenance, il nous est demandé de faire un réseau de neurones pouvant apprendre à simuler le principe de la porte XOR, dans le but de comprendre le fonctionnement du réseau dans un cas plus simple que la reconnaissance de chiffre.

C'est sur la base de ce réseau que la reconnaissance d'image a été faite.

3.3.2 L'utilisation de matrices

À travers les recherches, la méthode la plus simple est d'utiliser des matrices qui représentent les "Hidden layer". Cela permet de modifier les biais et les poids en parallèle sans passer dans chaque noeud.

Pour cela nous avons créé une structure "Matrix" qui permet de simuler le comportement d'une matrice en parcourant les adresses mémoire.

Elle est aussi associée à de nombreuses fonctions permettant de faire des opérations, telles que l'addition, la multiplication, etc.

3.3.3 Le réseau

Le réseau possède 2 entrées, 5 neurones intermédiaires et une sortie. L'apprentissage se fait à travers 100 000 "epochs" (itérations) qui corrigent les poids et les biais des neurones à la fin d'une "epoch". À la fin de cet apprentissage, on teste les réseaux afin de vérifier les résultats

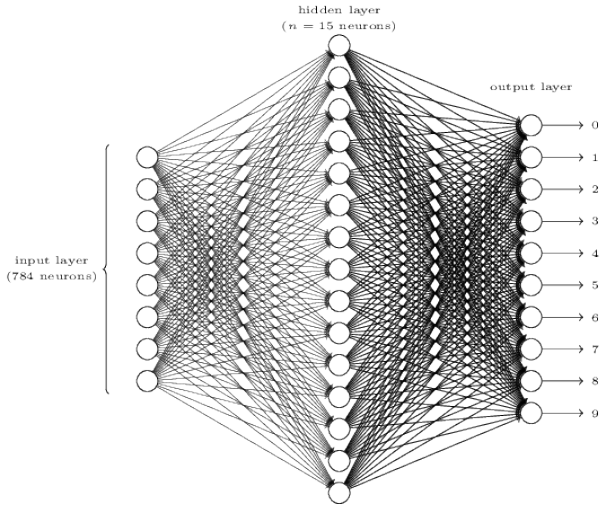


Figure 3.18: Schéma d'un réseau de neurones

3.3.4 La reconnaissance de chiffre

La reconnaissance de chiffre se fait de la même manière que le XOR mais le réseau a chaque pixel de l'image en entrée et 10 sorties. En effet, le nombre de neurones intermédiaires est beaucoup plus élevé à cause de la complexité liée au résultat attendu.

Les images en entrée ont toutes une taille de 28×28 pixels, ce qui permet de les comparer de la même manière que les autres.

Le réseau peut également reconnaître les cases vides, les associant à une valeur de 0.

3.3.5 L'entraînement

Le réseau est entraîné en lui donnant en entrée une image et en corrigeant les poids en fonction du résultat attendu.

Pour entraîner le réseau, nous avons collecté de nombreuses images que nous avons triées en fonction du nombre auquel elles correspondent.

Pour chaque 'epoch', le réseau va donc analyser chacune des images et donner

le résultat qu'il pense être correct. À la fin de l'analyse, les poids et les biais sont ajustés, avant de démarrer une nouvelle 'epoch'.

Le réseau pour ce cas précis contient 784 noeuds en entrée, 120 noeuds sur une couche cachée et 10 noeuds en sortie. L'entraînement se fait sur toutes les cases données par les sudokus d'exemple avec 25 000 générations.

Le réseau a une réussite de 100/100 parmi les cases d'entraînement. L'efficacité en cas réel est plus faible mais étant donné que ce réseau est entraîné en fonction du pré-traitement, il a une efficacité élevée dans le cadre du projet.

Interface

Notre interface est faite à l'aide de la bibliothèque GTK et de l'outil Glade.
La fenêtre est ajustable et se comporte de manière ergonomique.

3.4.1 Menus

Le logiciel propose 4 menus :

1. File : Ce menu permet de charger ou de sauvegarder les images à traiter ou traitées.
2. Rotate : Il propose de faire des rotations en automatique ou en manuel.
3. Neural nework : C'est ici que l'on charge ou entraîne le réseau de neurone.
4. Solve : On peut résoudre le sudoku d'un seul coup ou en affichant chaque étape les unes après les autres.

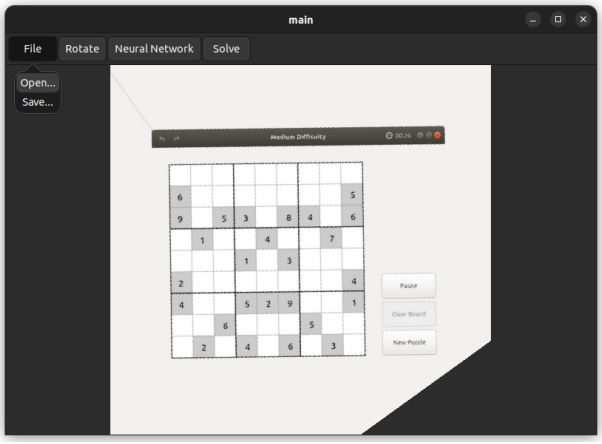


Figure 3.19: Interface avec une rotation d'image

3.4.2 Api GTK

Barre menu

La barre en haut de l'écran sert de menu pour accéder aux différentes fonctionnalités. Les différents boutons font apparaître des 'dropdown' permettant d'accéder aux sous-catégories de chaque option.

Affichage

L'interface utilise l'API GTK pour charger les images. Toutes les images trop grandes sont rétrécies en cas de besoin.

L'image de fin est générée à partir d'une grille vide, qui est ensuite remplie avec les chiffres qui ont étés reconnus : les chiffres étant déjà présents en noir et ceux renvoyés par le solveur en bleu.

En mode étapes par étapes, l'image est mise à jour en fonction de l'état actuel du preprocess.

Gestion des fichiers

Pour charger et enregistrer les fichiers, l'interface fait apparaître un gestionnaire de fichier pour que l'utilisateur puisse sélectionner l'image de son choix.

Il peut définir différents filtres permettant d'afficher les images, les fichiers grilles, ou tous les fichiers.

Le chargement du réseau de neurones fonctionne également ainsi mais permettant d'afficher les fichiers de sauvegarde.

Pour la sauvegarde des fichiers, l'utilisateur peut préciser le nom de sortie, le fichier étant sauvegardé en PNG. (plus de détails sur la gestion des fichiers dans la section 3.5)

3.4.3 Connexion au programme

Struct Interface

L'interface peut accéder aux différentes données nécessaires grâce à la struct 'Interface'. Voici sa définition :

```
typedef struct
{
    UI ui;
    Data data;
} Interface;
```

Elle est composée de 2 sous classes : UI et Data, contenant chacune les différents boutons de l'interface et les informations sur le statut de l'analyse.

La classe Data est principalement utilisée pour déterminer l'avancement du programme dans la résolution, notamment pour empêcher l'utilisateur d'analyser la grille avant que le réseau de neurones ne soit entraîné.

Cette structure est donc passée en paramètre de chacun des signal des boutons.

Connexion des boutons

Chaque bouton est associé à un signal gtk, et donc à une fonction spécifique. Certaines de ces fonctions vont faire apparaître une boite de dialogue supplémentaire, pour la sélection de fichier par exemple.

Le slider de la rotation manuel est lui aussi attaché à un signal, qui est activé à chaque changement de valeur.

Message d'erreur

En cas de comportement inattendu, le programme va afficher un message d'erreur pour signaler l'utilisateur. Cela empêche entre autre de résoudre une grille non analysée, analyser lorsqu'aucune image n'a été sélectionnée ou bien que le réseau de neurones n'est pas entraîné, etc.

Ces vérifications sont possible grâce à la classe Data, qui stocke l'avancée du traitement au fur et à mesure.

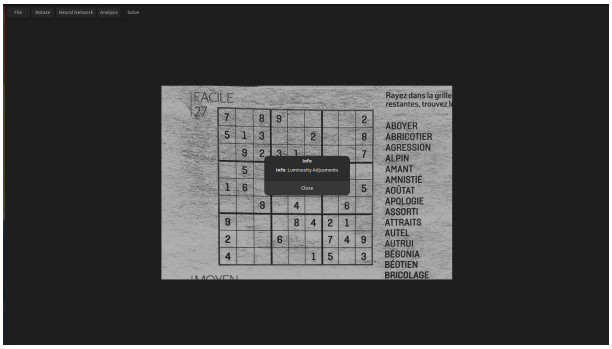


Figure 3.20: Exemple de message

Sauvegarde et chargement de fichiers

3.5.1 Sauvegarde de la grille

La grille obtenue peut être enregistrée en appuyant sur le bouton correspondant. L'utilisateur peut sélectionner le chemin et le nom de l'image. Elle est automatiquement sauvegardée au format PNG.

3.5.2 Chargement de la grille

La grille peut être chargée de différentes manières :

1. À l'aide d'un fichier image (.png, .jpg ou .bmp), ce qui nécessitera une analyse avant sa résolution.
2. À l'aide d'un fichier grille (comme définit par le cahier des charges), qui pourra être résolue immédiatement.

3.5.3 Sauvegarde des poids du réseau de neurones

La sauvegarde des poids et des biais se fait dans un fichier nommé 'save' qui indique les dimension de la matrice suivi des valeurs des poids et des biais.

Cette sauvegarde se fait automatiquement après l'entraînement du réseau de neurones.

```
120
2.527344
1.891529
4.151075
2.782586
3.064435
2.034117
3.015968
2.628228
2.537616
1.625201
2.229885
2.667618
2.352563
2.319380
2.339197
2.298457
1.956742
2.403504
2.626231
2.877664
2.894320
```

Figure 3.21: Aperçu d'un fichier 'save'

3.5.4 Chargement des poids du réseau de neurones

En appuyant sur le bouton du menu correspondant, il est possible de charger les fichiers 'save' obtenus dans le réseau.

Ce fichier va initialiser le réseau avec les valeurs décrites dans le fichier.

Environnement de Travail

Notre travail est principalement effectué sur un dépot GitHub, qui est ensuite synchronisé avec le dépôt GitLab de l'école. Cette démarche apporte de nombreux avantages.

Automatisation de tests

Chaque branche du projet est testée avant de pouvoir être fusionnée. GitHub va compiler l'exécutable du dossier tests et lancer le programme.

Ce programme est construit afin que chaque test non concluant renvoie une `EXIT_FAILURE`. Ainsi, en cas de code de retour non valide, la branche ne sera pas déployable.

Intégration Continue

Toutes ces opérations sont possibles grâce à l'intégration continue. Elle est programmable grâce à un fichier yaml décrivant sont comportement.

La CI effectue 3 étapes :

1. Elle compile l'exécutable et vérifie qu'il ne contient pas de warnings,
2. Elle lance les tests et vérifie qu'il n'y a pas d'erreurs,
3. Elle déploie le code du dépôt GitHub vers GitLab.

Chacune de ces étapes ne peut être effectuée si les étapes précédentes n'ont pas été validées. Ainsi, le code sur le dépôt GitLab est donc toujours valide et fonctionnel.

Makefile

Le Makefile est utilisé pour compiler ces différentes versions. Il comporte 5 entrées différentes :

1. Une entrée `main`, qui va compiler l'exécutable principal,
2. Une entrée `debug`, qui compile l'exécutable principal avec des options de debugging,
3. Une entrée `solver`, qui compile l'exécutable solver,
4. Une entrée `test`, qui compile l'exécutable de test (avec des options de debugging),
5. Une entrée `clean`, qui supprime tous les fichiers .o et exécutables.

Les options de debugging, autre que le classique `-g`, contiennent la définitions de variables telles que DEBUG. Ces variables sont utilisées dans le programme pour définir des fonctions dites de debug. Ces fonctions sont précédées de `DEBUG_` et entourées d'instructions `#if DEBUG ... #endif`. Elles ne seront donc pas compilées lors de la création de l'exécutable principal.

4.3.1 Options de compilation

Le Makefile permet de compiler différents exécutables à la fois :

1. L'exécutable `solver`, qui implémente une version du résolveur utilisable sur le terminal
2. L'exécutable `network`, qui implémente une version du réseau de neurones sur terminal
3. L'exécutable `test`, qui permet d'exécuter les différents tests.
4. L'exécutable `debug`, qui va compiler le projet en activant les diverses options de debugging
5. L'exécutable `main`, qui correspond à l'exécutable du projet.

Ambiance du groupe

Le groupe avait dans l'ensemble une bonne ambiance de travail.

Un serveur discord a été créé pour partager les informations et demander de l'aide si besoin. Il servait également à partager aux autres les ressources trouvées par chacun.