Rapport de la première soutenance - OCR

Joric Hantzberg - Alexiane Laroye

Ekaterina Schiff Dit Sarmois - Abel Roffiaen

Promo EPITA 2026

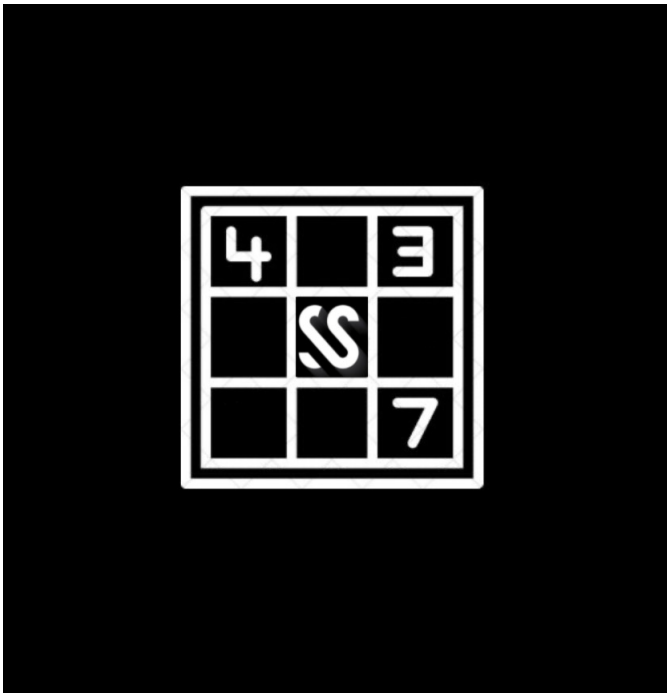


Table des matières

1 Introduction

Cela fait déjà plus d'un mois que notre projet d'OCR (Optical Character Recognition) a commencé. L'objectif de ce projet est de réaliser un logiciel de type OCR qui résout une grille de sudoku. En effet, le programme doit prendre en entrée une image représentant une grille de sudoku, de format quelconque et de dimension quelconque, et en resortir la-dite grille résolue.

2 Le Groupe

2.1 Scan&Solve

Nous devions donner un nom à notre groupe sachant que nous devons faire un OCR. C'est-à-dire nous devons scanner une image de sudoku et la résoudre. C'est pourquoi nous avons choisi « Scan » pour l'action de scanner l'image et « Solve » pour la résolution du sudoku de l'image.

2.2 Joric Hantzberg (Chef du groupe)

Etudiant en deuxième année de classe préparatoire à EPITA, je suis passionné d'informatique depuis tout petit. Grace à EPITA, j'ai l'occasion de réaliser des études dans un domaine qui me passionne et en pleine expansion. Ce nouveau projet proposé pour notre troisième semestre est l'occasion parfaite d'améliorer mes compétences de travail en groupe mais également d'acquérir de nouvelles compétences. C'est dans cette optique que j'ai décidé de m'occuper de la partie sur l'intelligence artificiel. C'est un sujet dont tout le monde a déjà entendu parlé mais dont peu de personnes connaissent le fonctionnement et c'est ce qui le rend si intéressant.

2.3 Ekaterina Schiff Dit Sarmois

En seconde, j'ai pu choisir une option reliée à l'informatique et le numérique et c'est comme cela que j'ai pu trouver ma passion. Coder dans différents langages, développer une certaine logique et être conduit à travailler en groupe pour mener à bien des projets m'ont donné envie d'en apprendre plus dans ce domaine et d'y travailler plus tard. C'est pourquoi je suis ravie de commencer ce projet avec mes camarades afin de résoudre un sudoku. Ayant envie d'en apprendre plus sur le traitement d'image, j'ai décidé de m'intéresser plus particulièrement à cette partie.

2.4 Alexiane Laroye

Actuellement en deuxième année de prépa à EPITA. Suite à ma première année, j'ai pu développer mes capacités en informatique mais surtout prendre goût à la programmation. En effet, derière le fait de coder, j'ai pu développer ma reflexion sur comment gérer un problème et le résoudre. c'est pourquoi, je trouve ce projet très intéressant à résoudre car il nous impose un problème et nous devons trouver la solution pour le résoudre. Ce n'est pas comme notre projet de l'année dernière, nous ne pouvons plus contourner le problème on doit trouver un moyen pour le résoudre car un cahier des charges nous est imposé. Je trouve ce projet très intéressant.

2.5 Abel Roffiaen

C'est ma deuxième année à l'EPITA et j'ai déjà pu avoir un aperçu de ce dont un projet est constitué lors du projet de l'année passée. Cependant, le cadre de ce-dit projet était très libre et n'avait pas pour but de contraindre, juste d'apprendre aux étudiants les éléments nécéssaires à la production d'un projet informatique. Dans ce projet de Reconnaissance Optique de Caractères (OCR) l'objectif est différent. Répondre à une demande client avec des objectifs précis. Contrairement au projet précédent, celui-ci contient des attentes particulières. Je suis confiant quant au fait que nous serons capables de répondre à cette demande et de délivrer un produit qui répondra au besoin indiqué.

2.6 Le logo

Nous avons créé un logo pour représenter notre groupe Scan&Solve. Celui est artisanal et simpliste. En effet, il représente un sudoku avec deux « s » pour faire réference à notre nom de groupe.

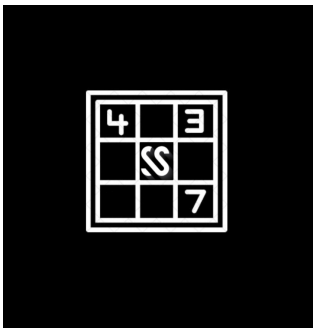


FIG. 1 – *Logo du groupe*

3 La répartition des charges

Tout d'abord, nous avons réparti les charges selon nos préférences et nos expériences pour faciliter la réalisation du projet. Vous trouverez, ci-dessous, le tableau des répartitions des charges.

Tâches / Personnes	Alexiane	Ekaterina	Abel	Joric
Chargement d'une image				
Suppression des couleurs				
Prétraitement				
Détection de la grille				
Détection des cases de la grille				
Récupération des cases à chiffres				
Reseau de neurones				
Reconstruction de la grille				
Résolution de la grille				
Affichage de la grille résolue				
Sauvegarde de la grille résolue				
Interface Graphique (GUI)				

4 L'avancement du projet

4.1 L'avancement de groupe

Comme vous avez pu l'observer, ce projet tourne autour du travail de groupe. Il est donc essentiel de s'entourer des bonnes personnes pour ce projet. Tout d'abord, nous obtiendrons des résultats supérieurs. Le travail de groupe est avant tout un bénéfice, car il nous permet de nous entraider, de fusionner nos forces. Dans un groupe, chaque personne a des qualités qui lui sont propres et qui sont un plus pour les autres membres.

Concernant l'avancement de groupe, nous nous avions fixé des dates limites pour chaque partie, nous sommes plus ou moins en avance sur certaines parties et en retard sur d'autres parties, cela fait que nous sommes dans les temps. En effet, des parties sont plus complexes que prévu. Cependant, comme il s'agit d'un travail de groupe, il y a toujours quelqu'un pour aider quelqu'un d'autre en cas de besoin. Pour l'instant, nous avons donc une bonne base de projet.

4.2 L'avancement de chaque partie

4.2.1 Chargement d'une image

Cette partie n'est pas encore commencé. En effet, nous nous intéresserons à cette partie pour la soutenance finale.

4.2.2 Prétraitements

La partie sur le prétraitement de l'image se découpe en trois parties principales. D'abord la suppression des couleurs puis la détection des éléments et enfin la rotation manuelle. Dans le code, le prétraitement se découpe en plusieurs fichiers, notemment les fichiers « grayscale.c » et « median_blur.c » respectivement responsable de la mise à gris et du floutage de l'image.

4.2.2.1 Suppression des couleurs (niveau de gris, puis noir et blanc)

Pour cette partie, nous avons récupéré le code d'un ancien TP afin de rendre notre image en nuance de gris. Celui-ci permettait de passer de l'image originale à l'image en gris à chaque fois qu'une touche était préssée. Nous avons modifié le code afin qu'à chaque fois que nous pressons une touche, une étape se déclenche: d'abord l'image en nuance de gris, puis la réduction de bruit et ensuite la binarisation.

```
// uses pixel_to_grayscale to convert the surface to gray
void surface_to_grayscale(SDL_Surface* surface)
{
    Uint32* pixels = surface->pixels; // array of pixel values
    int len = surface->w * surface->h; // height and width of the array
    if(SDL_LockSurface(surface) != 0){ // lock the surface for changes
        errx(EXIT_FAILURE, "%s", SDL_GetError());
    }

    for (int i = 0; i < len; i++){ // change all pixels to gray
        pixels[i] = pixel_to_grayscale(pixels[i], surface->format);
    }

    SDL_UnlockSurface(surface); // unlock the surface to finish
}
```

FIG. 2 – *La fonction « surface_to_grayscale »*

```
// If a key is pressed, get to the next step in the image pre-treatment
case SDL_KEYDOWN:
    if (booleen == 0){ // grayscale
        surface_to_grayscale(s);
        booleen = 1;
    }
    // missing step for blur !!
    else if(booleen == 1){ // black and white
        surface_to_black_and_white(s);
        booleen = 2;
    }
    else{ // back to begining
        s = surface;
        booleen = 0;
    }
    t = SDL_CreateTextureFromSurface(renderer, s);
    draw(renderer, t, alpha);
    break;
```

FIG. 3 – *Les étapes du prétraitement*

Pour la réduction de bruit, nous avons choisis un flou médian qui fonctionne en prenant la valeur de gris d'un certain nombre de pixels, nous trions par ordre croissant ces valeurs et finalement nous prenons la valeur médiane est l'appliquons à tous les pixels selectionnés.

```
// iterate in the cell to make the array of values
int startr = (i-1)*pgcd, startc = (j-1)*pgcd;
for(int k = startr; k < i*pgcd; k++){
    for(int l = startc; l < j*pgcd; l++){
        // get the value in pixels array
        array[k - startr + l - startc] = get_color(pixels[k + cols * l], surface->format);
    }
}

int median = ClassicMedianFilter(array, 9); // get the median value

// change all values in the cell to the median value
for(int k = startr; k < i*pgcd; k++){
    for(int l = startc; l < j*pgcd; l++){
        pixels[k + cols * l] = pixel_to_color(surface->format, median);
    }
}
```

FIG. 4 – *Utilisation de la médiane des valeurs*

Concernant la binarisation (noir et blanc) nous avons rendu une binarisation à seuil fixe, c'est-à-dire que si la valeur de gris du pixel est au dessus de 127, le pixel sera mis en blanc, sinon en noir. Nous avons commencé à travailler sur une binarisation à seuil dynamique utilisant l'image entière en suivant un article[1].

```
// iterate in the pixels array
for(int i = 0; i < length; i++){
    int gray = get_gray(pixels[i], surface->format); // get the grayscaled value
    if(gray < Limit) pixels[i] = pixel_to_black(surface->format, 0); // black pixel
    else pixels[i] = pixel_to_black(surface->format, 255); // white pixel
}
```

FIG. 5 – *La binarisation statique*

4.2.2.2 Rotation manuelle

Nous avons choisi de faire une rotation manuelle de la facon suivante: il suffit d'indiquer la source de l'image ainsi que son degré de rotation et l'image apparaitra dans une fenêtre avec le bon angle.

```
SDL_RenderCopyEx(renderer, texture, NULL, NULL, alpha, NULL, SDL_FLIP_NONE);
```

FIG. 6 – *La rotation manuelle se fait à l'affichage*

4.2.2.3 Détection des éléments et extraction des images de nombres

Nous n'avons pas encore pu traiter cette partie mais celle-ci sera complète lors de la prochaine soutenance.

4.2.3 Reseau de neurones

4.2.3.1 Recherche

Au début du projet, le fonctionnement d'un réseau de neurones m'était totalement inconnu. Afin de réaliser une intelligence artificielle capable de déterminer la valeur d'un chiffre en écriture manuscrite, il ma donc fallu commencer par de grandes recherches. Tout d'abord, j'ai suivi le lien donné

1. Article de Derek Bradley et Gerhard Roth `https://www.researchgate.net/publication/220494200_Adaptive_Thresholding_using_the_Integral_Image`

dans le cahier des charges (lien[2].) Grace à ce site web très riche en informations j'ai appris le fonctionnement de deux types de neurones : les perceptrons et les sigmoïdes neurones . Par la suite j'ai compris comment était constitué un réseau de neurones ainsi que sont fonctionnement. Cependant la partie du site consacré à l'apprentissage du réseau était un peu trop mathématique et j'avais du mal à comprendre l'algorithme de la descente de gradient utilisé pour l'apprentissage. Je me suis donc dirigé vers deux vidéos YouTube ainsi qu'un site web (lien[3],[4] et[5]). Grace à ces trois sources d'informations j'ai pu dans un premier temps comprendre le fonctionnement de cet algorithme mais également me faire une idée sur la façon dont je pouvais implémenter mon réseau de neurones.

4.2.3.2 Composition d'un résau de neurones et fonctionnement d'un neurone

Afin de comprendre l'implémentation du réseau de neurones et des algorithmes qui en découle, je vais d'abord expliquer le fonctionnement des différents neurones ainsi que la structure du réseau.
Tout d'abord, il faut discerner deux types de neurones. Les premiers sont les perceptrons, ce sont les premiers neurones qui ont été inventé. Ces neurones vont agir comme une fonction mathématique. Ils prennent en entrée autant de valeurs que l'utilisateur souhaite. Ensuite chaque branche reliant une valeur au neurone contient une deuxième variable : le poids. Et pour finir le neurone contient une dernière valeur appelée biais. Grace à ces trois types de variables différentes on va pouvoir déterminer la valeur de sortie du neurone à l'aide de cette fonction mathématique :

$$\text{output} = \begin{cases} 0 \text{ if } poid_i \cdot value_i + b \leq 0 \\ 1 \text{ if } poid_i \cdot value_i + b > 0 \end{cases} \text{, avec } i \text{ le numero de l'entrée}$$

Cependant, cette représentation du neurone comprend un gros soucis comme on peut le voir sur la figure 7 : la valeur de sortie est binaire, 1 ou 0. De ce fait, un léger changement dans les variables utilisées pour déterminer la sortie peut totalement inverser la sortie d'un réseau de neurones.

C'est pour cela que les sigmoïdes neurones ont été inventé. Le fonction-

2. http://neuralnetworksanddeeplearning.com/chap1.html
3. https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
4. https://www.youtube.com/watch?v=LA4I3cWkp1E
5. https://www.youtube.com/watch?v=w8yWXqWQYmU&t=573s

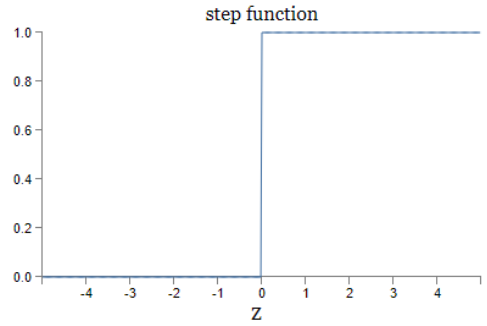


FIG. 7 – *Graphique de la fonction perceptron*

nement d'un sigmoïde est relativement le même que celui du perceptron. La grande différence est la fonction mathématique qui détermine la sortie du neurone en fonction des variables. Cette fonction s'appelle fonction sigmoïde et vaut :

$$\frac{1}{1+e^{-}z} \text{ avec } z = \sum poids_i.value_i + biais$$

Grâce à cette fonction, la sortie du neurone ne sera plus binaire mais comprise entre 0 et 1 comme on le voit sur la figure 8.

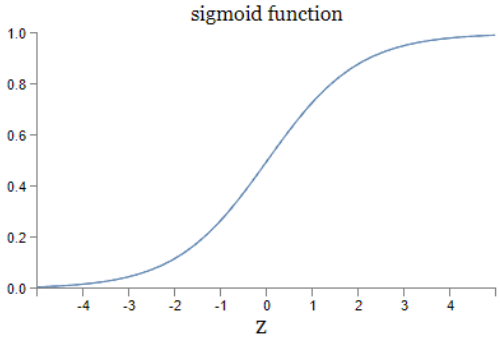


FIG. 8 – *Graphique de la fonction sïgmoide*

De ce fait, un petit changement dans les poids ou le biais aura une plus petite influence sur la sortie du neurone et il sera donc plus facile d'implémenter un algorithme de machine Learning.
Maintenant que nous avons défini le fonctionnement d'un neurone, il est temps de montrer comment est construit un réseau de neurones. Dans le schéma qui va suivre, les neurones seront représentés par des ronds et les

flèches seront les liens (sortie → entrée) entre les neurones.

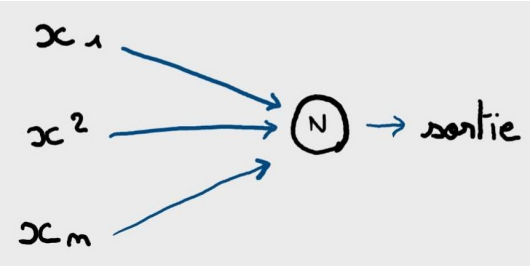


FIG. 9 – *Représentation graphique d'un neurone*

L'implémentation d'un réseau de neurones est relativement simple. On retrouve trois grands composants. Le premier est la couche d'entrée. C'est une liste de neurones qui servira de stockage pour les valeurs d'entrée données par l'utilisateur. Ensuite nous avons les couches cachées. Leur nombre peut varier mais dans notre cas, une couche suffira. Ces couches sont de nouveau constituées d'une liste de neurones mais ces neurones contiendront aussi des poids et un biais utilisés pour calculer la valeur de sortie du neurone. Et enfin, il reste la dernière couche de notre réseau de neurones qui contient de nouveau une liste de neurones avec des poids et un biais. Cette couche est similaire aux couches cachées à la différence que les valeurs de chacun des neurones sont les résultats du réseaux. Pour mieux comprendre cette structure, vous pouvez regarder la figure 11 qui est une représentation graphique d'un réseau de neurones.

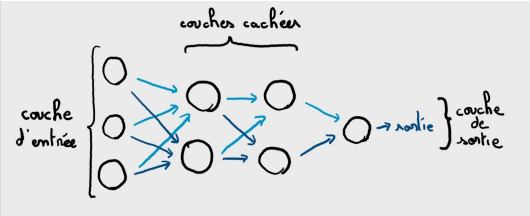


FIG. 10 – *Représentation graphique d'un réseau de neurones*

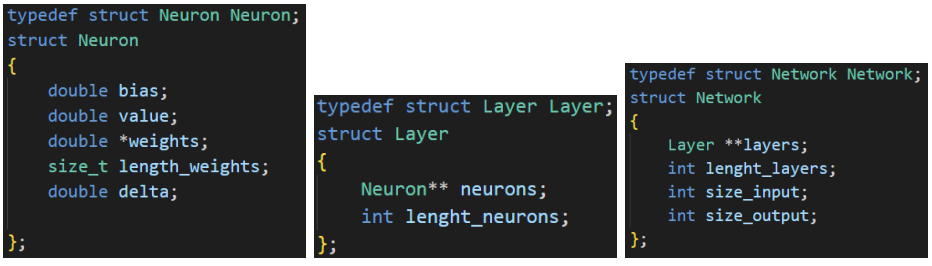
4.2.3.3 Implémentation

Maintenant que nous nous sommes bien documenté, il est temps de passer aux choses sérieuses : le code. Pour implémenter notre réseau de neurones, plusieurs choix s'offrent à nous. Tout d'abord pour des raisons évidentes nous avons optés pour des « sigmoïdes neurones » et non des perceptrons. Ensuite,

pour ce qui est de la structure du réseau, deux choix étaient possibles: les matrices et les structures. Les structures sont en réalité des classes et nous avons choisi pour notre implémentation de les utiliser car cette méthode nous semblait plus intuitive que l'utilisations de matrices. Lors de notre premier essai, nous avons fais une unique structure contenant la liste des biais ainsi que la liste des poids de tous le réseau. Cependant cette implémentation a vite montré ses limites. Premièrement à cause des difficultés rencontrées à déterminer quels poids sont liés à quels neurones mais également lors de l'implémentation de l'algorithme d'apprentissage. C'est pour cette raison que nous avons modifié notre implémentation pour ne plus avoir une mais trois structures.

Tout d'abord, les neurones : cette structure contient une liste de poids (de type « double ») de la longueur de la couche précédente, un flottant « biais », un flottant « delta » (utilisé pour l'apprentissage), un flottant « valeur » et un entier « longueur » indiquant la longueur de la liste de poids. Ainsi, il est simple de récupérer les informations pour chaque neurone.
Ensuite, les « layer » (couches en français) : cette structure contient simplement une liste de neurones ainsi qu'un entier indiquant sa longueur.

Pour finir, le réseau : cette dernière structure est le regroupement des deux structures précédentes. Elle contient une liste de layer et un entier indiquant sa taille. Mais attention, il ne faut pas oublier que chaque layer présent dans la liste contient une liste de neurones qui contiennent elles-mêmes une liste de poids. Nous nous retrouvons ainsi avec une grande structure contenant toutes les informations sur notre réseau de neurones.



(a) *Structure neurone* (b) *Structure layer* (c) *Structure réseau de neurones*

FIG. 11 – *Structures des éléments du réseau*

4.2.3.4 Initialisation du réseau

Une fois que notre réseau est implémenté, il faut l'initialiser. Pour ce faire nous avons utilisé trois fonctions. La première initialise les neurones. Pour ce faire la valeurs du biais est générée aléatoirement entre 0 et 1. La liste de poids est créée grâce à « Malloc » et la taille de la couche précédente. Pour finir il ne nous reste qu'à initialiser la liste de poids avec des valeurs aléatoires entre 0 et 1.
La deuxième fonction sert à initialiser les couches de neurones. Cette partie est assez simple. Il nous suffit en effet de créer une liste de neurones avec « Malloc » puis d'initialiser tous les neurones de la liste grâce à la fonction précédente. La dernière fonction sert à initialiser notre réseau. Elle prend une liste en entrée indiquant le nombre de neurones dans chaque couches du réseau. Grace à cette liste, nous pouvons créer une liste de layer et utiliser la fonction d'initialisation de layer sur chacun. Cette fonction nous retourne un pointeur vers l'emplacement mémoire où est stocké notre réseau. C'est ce pointeur que nous passerons en paramètre de toutes les fonctions utilisant notre réseau.
Pour finir sur cette partie, il ne faut pas oublier de créer des fonctions pour libérer l'espace réservé par « Malloc » lors de l'initialisation du réseau.

4.2.3.5 « FeedFoward »

La prochaine étape logique dans la création de notre réseau de neurones est simplement le calcul du résultat de notre réseau en fonction des paramètres entrés. Pour ce faire nous avons d'abord implémenter la fonction sigmoïde qui va renvoyer le résultat de

$$\frac{1}{1 + e^{-}z}$$

Ensuite, nous avons créé la fonction « FeedForward ». Cette fonction calcule la valeur de chaque neurone pour chaque couche en commençant par la première couche cachée. Cette opération se fait en deux parties, la première consiste a calculer z pour notre fonction d'activation. La formule de ce calcul est

$$\sum poids_{ik}.value_{i-1k} + biais$$

Avec i le numéro de la couche de notre neurone et k le numéro du neurone

Grace à notre implémentation, chaque neurone contient déjà la liste des poids qui arrive sur lui donc ce calcul est d'autant plus simple à implémenter. Il suffit ensuite d'utiliser la fonction sigmoïde avec en paramètre la somme précédemment calculée pour avoir la valeur de sortie du neurone.

4.2.3.6 Apprentissage

La phase d'apprentissage est réalisée grâce à l'algorithme de la descente de gradient. Cette algorithme se décompose en deux parties : le calcul des taux d'erreurs et la modification des poids et biais.
Pour calculer les taux d'erreurs (variable *delta*) des neurones de la couche de sortie, il suffit de faire *delta = expected value - computed value*.

Ensuite, il faut remonter le réseau en commençant par la dernière couche cachée. La formule de calcul des deltas est légèrement différente. Pour cela nous utiliserons la formule suivante :

$$d = (sigmoïde(value))' \cdot \sum poids_{i+1kj} \cdot delta_{i+1k}$$

Avec i le numéro du layer, k le numéro du neurone de la couche $i+1$ et j le numéro du neurone j

Ensuite, il faut mettre à jour les poids et les biais du réseau grâce aux deltas calculés. Pour ce faire on utilise la formule suivante :

$$w_k = w_k + delta \cdot learning\ rate \cdot value\ neuron_{i-1k}$$

Pour la modification du biais, la formule est encore plus simple :

$$biais = biais + delta \cdot learningrate$$

4.2.3.7 Train network

Pour finir avec la partie apprentissage du réseau de neurones. Il faut réaliser une fonction qui va appliquer n fois l'algorithme de la descente de gradient afin d'avoir les meilleurs résultats possibles. Cette fonction est très importante car si on n'applique qu'une fois notre algorithme d'apprentissage sur le réseau de neurones, les changements seront trop minimes pour réellement influencer les valeurs de sortie. C'est pourquoi on répète en boucle l'algorithme pour corriger de façon contrôlé les poids et biais du réseaux.

4.2.3.8 Prochaines réalisations

Pour ce qui est des prochaines réalisations liées au réseau de neurones, il ne reste plus beaucoup de fonctions à implémenter étant donné que toute la partie « machine learning » est terminée. La partie de notre OCR lié au réseau de neurones est donc presque terminée. Pour la prochaine soutenance

il reste donc uniquement le chargement et la sauvegarde des poids ainsi que la conversion du mnist (jeu de données d'entraînement du réseau).

4.2.4 Reconstruction de la grille

Nous n'avons pas encore pu traiter cette partie mais celle-ci sera complète lors de la prochaine soutenance.

4.2.5 Résolution de la grille

Cette partie est une des bases de notre projet. En effet, elle va nous permettre de résoudre le sudoku.

Cette partie n'est pas la plus compliquée car l'année dernière nous avions déjà fait un TP de programmation pour résoudre un sudoku. Cependant, nous voulions un programme plus optimisé. De plus, cette année notre projet est en C et non en C# contrairement à l'année dernière.

Pour commencer, nous avons créé un fichier « solver.c » qui est constistué de deux fonctions. La première permet de savoir si un chiffre peut être positionné à la ligne et à la colonne indiquées sinon il cherchera récursivement où le placer à l'aide de la deuxième fonction qui résout le sudoku en faisant appel à la première fonction. Nous appelons la fonction pour résoudre le sudoku avec le fichier qui nous renverra notre fonction qui parse le fichier rentré par l'utilisateur.

Ensuite, nous avons créé un fichier « solverMain.c » qui regroupe plusieurs fonctions. La fonction « main » fait appel aux fonctions présentes dans ce fichier qui tout d'abord fait appel à la fonction qui parse notre fichier d'entrée. C'est-à-dire le rôle du parseur est de remplir une matrice statique de 9 par 9 à l'aide du fichier rentré par l'utilisateur en argument. Comme nous devons respecter un format spécifique de la grille de sudoku, notre parseur doit prendre en compte que horizontalement, il y a un espace entre chaque groupe de 3 cases et verticalement, il y a une ligne vide entre chaque groupe de 9 cases. Il remplace également les points qui correspondent aux cases vides par le chiffre 0. Ensuite, la fonction « main » appelle la fonction pour sauvegarder la grille résolue, celle-ci écrit le résultat dans un fichier qui suit les mêmes caractéristiques que le fichier que nous avons parsé. Celle-ci crée donc un fichier qui contient le résultat. Celui-ci se nomme avec une extension « .result ». Sans oublier que notre fonction « main » contient un cas d'erreur si les éléments en paramètre ne sont pas valides. La fonction « main » permet

donc d'assembler toutes nos fonctions pour que notre « Solver » fonctionne.

Bien évidemment, il ne fallait pas oublier de créer un « Makefile » pour la compilation des fichiers.

Cette partie est donc fini, cependant nous pensons effectuer quelques améliorations si nous avons du temps.

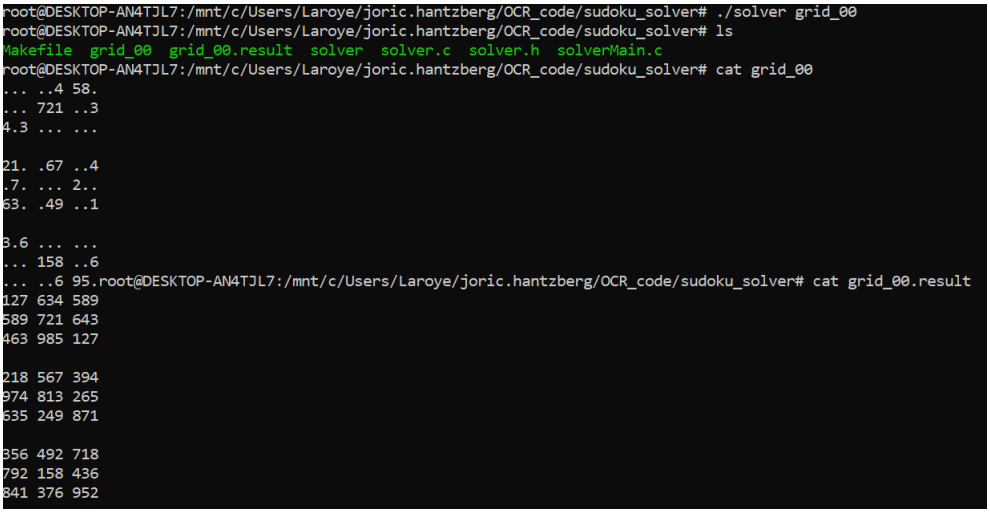


FIG. 12 – *Démonstration du solver*

4.2.6 Affichage de la grille résolue

Nous n'avons pas encore commencé cette partie, cependant nous y avons déjà réfléchi. Nous envisageons d'afficher le résultat sous la forme d'une image. Les cases remplies par l'algorithme, celles qui étaient initialement vides s'afficheront d'une couleur différente de celles initialement remplies.

4.2.7 Sauvegarde de la grille résolue

Concernant cette partie, nous avons simplement écrit une fonction dans la partie résolution de la grille qui permet d'écrire le résultat dans un fichier lorsque nous compilons le fichier correspond à la résolution de la grille. Nous n'avons pas encore sauvegardé le résultat sous la forme d'une image.

4.2.8 Interface Graphique (GUI)

Cette partie n'est pas vraiment commencée, nous avons juste commencé à nous renseigner sur comment l'effectuer. En effet, nous nous intéresserons à cette partie pour la soutenance finale. Nous ferons cela à l'aide de l'outil: GTK[6]. Cet outil est un ensemble de bibliothèques logicielles, c'est-à-dire un ensemble de fonctions permettant de réaliser des interfaces graphiques.

4.3 L'assemblage du projet

Concernant l'assemblage du projet, pour l'instant nous n'avons pas spécialement mis notre travail en commun car cela se fera à la fin quand nous aurons réalisé la partie qui va rassembler toutes nos parties.

4.3.1 Prétraitements

Afin de compiler les programmes de prétraitement, nous avons un fichier « Makefile » qui fait la compilation de tous les fichiers de cette partie du projet. Pour l'utiliser, il faut se placer dans le répertoire « /OCR_code/image_pretreatment/ » et écrire les commandes qui suivent.

La commande qui permet de compiler tous les fichiers du répertoire en un unique executable:

$ make

La commande qui permet de nettoyer le répertoire courant en ne gardant que les fichiers source:

$ make clean

4.3.2 Test de l'intelligence artificiel

Pour tester l'intelligence artificiel, vous avez à votre disposition un petit programme mettant en place un XOR. La fonction initialise un réseau, l'entraine et test ensuite la fonction avec toutes les combinaisons possibles du XOR. Pour exécuter ce test, il faut se placer dans le dossier « /OCR_code/neural_network/src » et entrer la commande « gcc -Wall -Wextra -std=c99 -lm *.c » pour ensuite exécuter le fichier « ./a.out ».

6. `https://www.gtk.org/`

4.3.3 Résolution de la grille

Pour compiler, il faut entrer la commande « make » dans le dossier « sudoku_solver ». Après avoir exécuté cette ligne, il faut écrire « ./solver (le nom du fichier). » Si vous regardez vos fichiers, vous trouverez un nouveau fichier qui contient le résultat. Pour supprimer les fichiers qui sont apparus suite à l'exécution du « make » , il faut entrer la commande « make clean » .

5 Conclusion

En conclusion, nous avons pris goût à la conception de ce projet et à ce travail de groupe qui nous permet de découvrir de nouvelles choses et d'acquérir de l'expérience. Pour la prochaine soutenance nous allons finir le traitement d'image et l'IA. Nous comptons également créer une interface graphique et si le temps nous le permet un site web. C'est pourquoi, nous attendons avec impatience de vous présenter la fin du développement de notre projet à la soutenance finale.