Rapport de projet - OCR

Joric Hantzberg - Alexiane Laroye

Ekaterina Schiff Dit Sarmois - Abel Roffiaen

Promo EPITA 2026

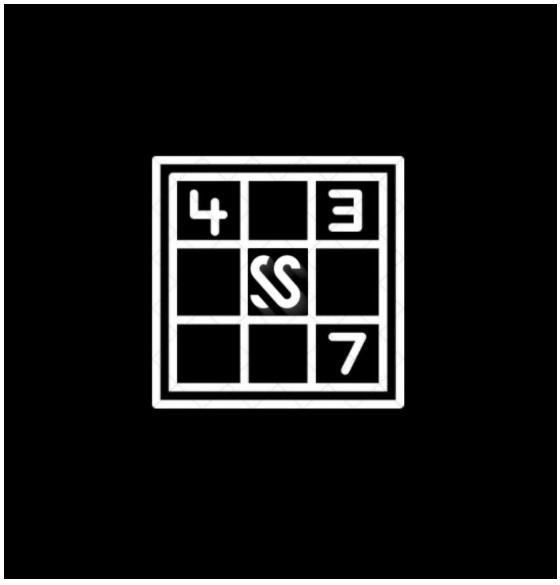


Table des matières

1 Introduction

Cela fait plusieurs 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.

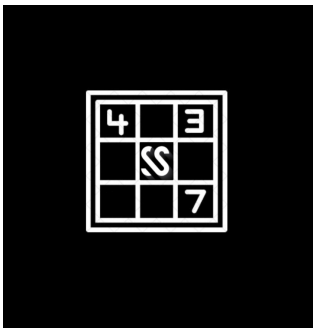


FIG. 1 – *Logo du groupe*

3 Cahier des charges

L'objectif de ce projet est de réaliser un logiciel de type OCR (Optical Character Recognition) qui résoutune grille de sudoku.

L'application prendra donc en entrée une image représentant une grille de sudoku et affichera en sor-tie la grille résolue.

Dans sa version définitive, l'application devra proposer une interface graphique permettant de chargerune image dans un format standard, de la visualiser, de corriger certains de ses défauts, et enfin d'afficherla grille complètement remplie et résolue.

La grille résolue devra également pouvoir être sauvegardée. L'application devra aussi posséder un aspect apprentissage, qui pourra être séparé de la partie princi-pale, et qui permettra d'entraîner votre réseau de neurones, puis de sauvegarder et de recharger le résultatde cet apprentissage.

4 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)				

5 L'avancement du projet

5.1 Première soutenance

5.1.1 Ekaterina et Abel

5.1.1.1 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.

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*

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*

5.1.2 Alexiane

5.1.2.1 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

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

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.

5.1.2.2 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.

5.1.2.3 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.

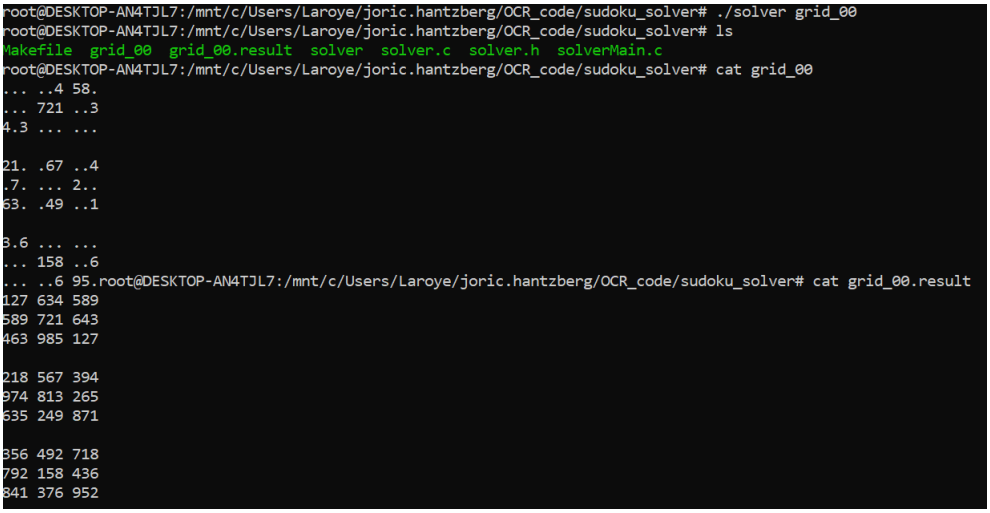


FIG. 7 – *Démonstration du solver*

5.1.2.4 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[2]. Cet outil est un ensemble de bibliothèques logicielles, c'est-à-dire un ensemble de fonctions permettant de réaliser des interfaces graphiques.

5.1.3 Joric

5.1.3.1 Reseau de neurones

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 dans le cahier des charges (lien[3].) 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

2. `https://www.gtk.org/`
3. `http://neuralnetworksanddeeplearning.com/chap1.html`

deux vidéos YouTube ainsi qu'un site web (lien[4],[5] et[6]). 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.

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 fonctionnement 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.

4. https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
5. https://www.youtube.com/watch?v=LA4I3cWkp1E
6. https://www.youtube.com/watch?v=w8yWXqWQYmU&t=573s

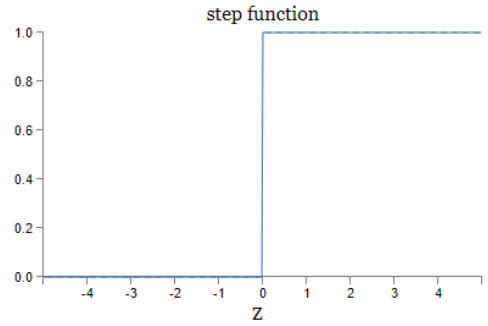


FIG. 8 – *Graphique de la fonction perceptron*

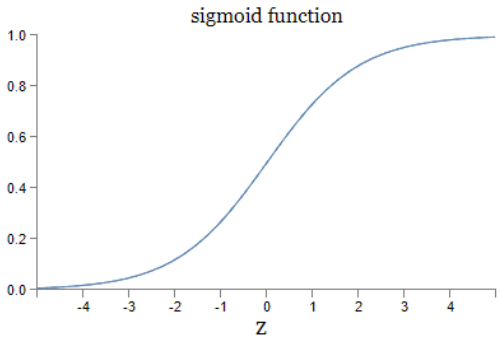


FIG. 9 – *Graphique de la fonction sïgmoide*

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.

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

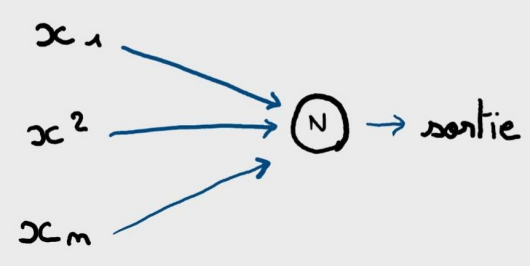


FIG. 10 – *Représentation graphique d'un neurone*

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.

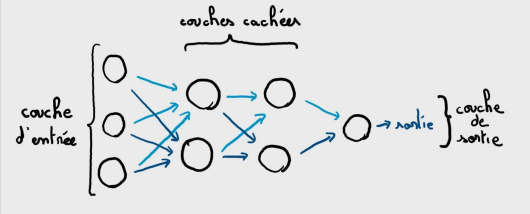


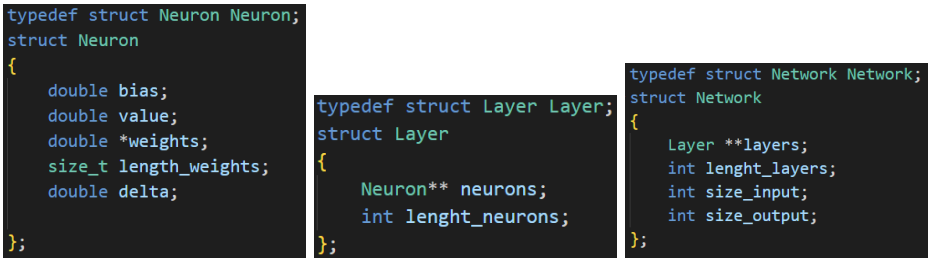
FIG. 11 – *Représentation graphique d'un réseau de neurones*

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. 12 – *Structures des éléments du réseau*

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.

« **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.

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 :

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

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.

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).

5.1.4 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.

5.1.4.1 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

5.1.4.2 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 ».

5.1.4.3 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 » .

5.2 Deuxième soutenance

5.2.1 Ekaterina

5.2.1.1 Réduction de bruit

Comme annoncé durant la première soutenance, nous avons choisi de faire une réduction de bruit à l'aide de la technique du flou médian. Nous n'avions que la fonction qui triait par ordre croissant les nuances de gris contenues dans un tableau de 9 valeurs et retournait la valeur médiane. Maintenant nous appliquons cette fonction sur l'ensemble de l'image ce qui donne le résultat de flou.

```
for (size_t i = 1; i < rows-1; i++){
    for (size_t j = 1; j < cols-1; j++){
        // get all surrounding pixels
            array[0] = changepixels[i*cols+j];
            array[1] = changepixels[(i-1)*cols+j-1];
            array[2] = changepixels[(i-1)*cols+j];
            array[3] = changepixels[(i-1)*cols+j+1];
            array[4] = changepixels[(i+1)*cols+j-1];
            array[5] = changepixels[(i+1)*cols+j];
            array[6] = changepixels[(i+1)*cols+j+1];
            array[7] = changepixels[i*cols+j-1];
            array[8] = changepixels[i*cols+j+1];
        // compute the median value and change the pixel color
        median = ClassicMedianFilter(array, 9);
        pixels[i*cols+j] = pixel_to_color(surface->format, median);
```

FIG. 13 – *Code appliquant le flou médian à tous les pixels*

J'ai cherché une image sur Internet contenant beaucoup de bruit afin de montrer la différence avant / après l'application de la fonction de réduction de bruit.

5.2.1.2 Binarisation

La binarisation utilisait un seuil fixe durant la première soutenance. Désormais, le seuillage dynamique est implémenté tout en suivant l'article de Derek Bradley et Gerhard Roth.[7]

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

(a) *AVANT* (b) *APRES*

FIG. 14 – *Image avec bruit AVANT/APRES flou médian*

5.2.1.3 Détection de la grille

Concernant la détection de la grille, j'ai créé une fonction auxiliaire qui pour des coordonnées de pixel données, va vérifier si ses 8 voisins sont noirs et pas déjà marqués. Si c'est le cas, le voisin est ajouté dans une pile et va être à son tour marqué avec la même valeur. Cela permet d'avoir des "blocs" de valeurs et le plus grand bloc sera donc la grille. les valeurs sont stockées dans une matrice de la même taille que l'image, initialement les valeurs sont à 0. La fonction principale parcours toute l'image, vérifie qu'un pixel n'est pas déjà marqué, est noir et augmente la valeur du marquage. J'ai donc fait une autre fonction qui va renvoyer la valeur de marquage la plus présente dans la matrice (hormis 0, qui signifie que le pixel est blanc) puis je reparcours toute la grille et dès qu'un pixel est marqué avec la valeur de marquage la plus présente, je la mets en rouge. Finalement, la grille s'affiche en rouge.

```
int valuemax = histo(mat, rows*cols, value);
for (size_t i = 0; i < rows; i++){
    for (size_t j = 0; j < cols; j++){
        if (mat[i*cols+j] == valuemax){ //we color it in red.
            pixels[i*cols+j] = pixel_to_colors(surface->format, 255, 0, 0);
        }
    }
}
```

FIG. 15 – *Code permettant de déterminer la valeur attribuée aux pixels constituant la grille et la mettant en rouge*

J'ai créé deux fonctions d'histogrammes: une en X et une en Y permettant d'aider à la suite, c'est-à-dire à la détection des coordonnées de la grille.

J'ai choisis d'utiliser une pile pour la détection de la grille car faire une fonction récursive aurait pris trop de temps et il y aurait eu sûrement une erreur. Pour créer une pile en C, j'ai utilisé une struct avec quatre paramètres: la position du pixel si la matrice est à une dimension (donc un tableau), ses coordonnées x et y si la matrice est à deux dimensions et un pointeur vers le prochain élement. x et y permettent de savoir si le pixel est sur un bord, pour

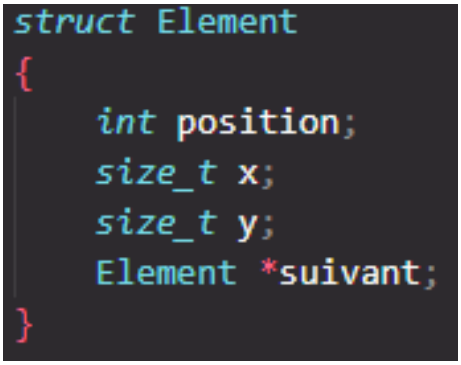


FIG. 16 – *Structure d'une pile*

éviter un dépassement lors de la vérification des 8 pixels, par exemple si x et y sont égaux à 0, cela signifie que notre pixel est en haut à gauche de l'image, nous ne pourrions donc pas vérifier les pixels au-dessus et à sa gauche. J'ai également créer une fonction « isempty » pour vérifier que la pile est vide et ne contient plus de pixels à marquer, une fonction « empiler » pour empiler le voisin coloré en noir et non marqué ainsi que la fonction « dépiler » pour traiter le pixel et lui attribuer la valeur de marquage.

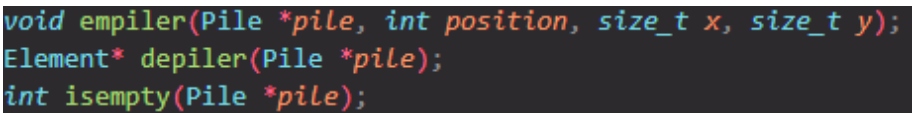


FIG. 17 – *Fonctions liées à la pile*

5.2.2 Abel

5.2.2.1 Binarisation à seuillage dynamique

Notre binarisation, lors de la première soutenance, était beaucoup trop simple. Elle vérifiait seulement si un pixel avait une valeur grisée supérieure ou inférieure à 127 (la valeur du gris parfait). Désormais, notre binarisation utilise la matrice intégrale de notre matrice de pixels et une somme calculée sur chaque pixel. Cette somme est ensuite pondérée par la moyenne des valeurs des pixels environnants et comparée à une valeur de seuil comme ci-après (*Fig. 18*).

```
// loop to get the new values
for(size_t i = 0; i < cols; i++){

    // columns
    x1 = max(i - s2, 0);
    x2 = min(i + s2, cols-1);

    for(size_t j = 0; j < rows; j++){

        // rows
        y1 = max(j - s2,0);
        y2 = min(j + s2, rows-1);

        count = (x2 - x1)*(y2-y1); // number of values
        sum = max(intImg[y2*cols+x2] - intImg[y1*cols+x2] - intImg[y2*cols+x1] + intImg[y1*cols+x1],0);

        if (get_pixel_red(pixels[j*cols+i],surface->format) * count <= (sum * (1 - threshold)))
        {
            pixels[j*cols+i] = pixel_to_color(surface->format, BLACK);
        }
        else
            pixels[j*cols+i] = pixel_to_color(surface->format, WHITE);
    }
}
```

FIG. 18 – *Itération dans l'ancienne matrice de pixel, somme, comparaison et attribution de la nouvelle valeur*

Cependant, cette implémentation n'est pas tout-à-fait satisfaisante. En effet, le seuil étant le même pour chaque image, certaines images avait un rendu trop noirci tandis que d'autre, au contraire, un rendu trop blanchi. Afin de palier ce problème, j'ai tenté d'implémenter une dynamisation de ce seuil en partant d'un seuil de 1% et en augmentant ce seuil de 1% si celui d'avant n'était pas suffisant (*Fig. 19*).

```c
void to_binary(SDL_Surface* surface)
{
    size_t len = surface->w * surface->h; //  array size
    Uint32 start_pixels[len]; // array of starting pixels
    get_pixels_values(surface->pixels, start_pixels, &len); // copy into new array
    float threshold = 1.0; // set threshold

    while(is_binarized(surface->pixels, &len) > MEAN){ // check good binarization
        get_pixels_values(start_pixels, surface->pixels, &len); // copy starting array into surface
        binarisation(surface, threshold/100.0); // do binarization
        threshold += 1.0; // increase threshold
    }
}
```

FIG. 19 – *Itération jusqu'à un seuil suffisant*

Toutefois, une erreur que je n'ai pas réussi à trouvé dans le code nous empêche d'utiliser cette dynamisation du seuil. L'implémentation fonctionnelle reste donc la binarisation dynamique à seuillage fixe à ce jour.

5.2.2.2 Rotation automatique

C'est trop peu de temps avant la soutenance que nous nous sommes rendu compte de notre erreur. En effet, pour la première soutenance nous avions implémenté la rotation manuelle grâce à la fonction « SDL_RenderCopyEx() » de la bibliothèque SDL. Néanmoins, il est impossible de transposer cette fonction d'affichage sur le GUI. Alexiane, en avance sur sa partie, a pu m'aider et a réussi à faire la rotation manuelle que j'ai ensuite pu réutiliser pour la rotation automatique. Pour cette rotation automatique j'ai décidé de trouver les sommets A, B, C et D de la grille grâce à la structure « struct Point » que j'ai créé et à la fonction « find_in_array() » (*Fig. 20 et 21*).

```c
/** Point type. */
struct Point{
    size_t x;
    size_t y;
};
```

FIG. 20 – *Implémentation de la structure Point*

```c
/** Find the first occurence of value in matrix and create a Point with its coordinates.
struct Point* find_in_array(Uint32* matrix, size_t rows,
    size_t start, size_t end, Uint32 value)
{
    struct Point* A = NULL;
    if(start < end){
        for(; !A && start < end; start++){
            if(matrix[start] == value)
                A = new_point(start/rows, start%rows);
        }
    }
    else{
        for(; !A && start > end; start--){
            if(matrix[start] == value)
                A = new_point(start/rows, start%rows);
        }
    }
    return A;
}
```

FIG. 21 – *La fonction « find_in_array() »*

Grâce à ces quatre points, on connait désormais quatre valeurs pour le sinus de l'angle de rotation (*Fig. 22*) et on peut donc calculer l'arcsin pour connaître la valeur de l'angle en question. On applique ensuite la rotation manuelle pour trouver l'image droite. Comme on a quatre valeurs, on utiliste la moyenne des quatre pour faire le calcul afin d'obtenir un résultat plus précis (*Fig. 23*).

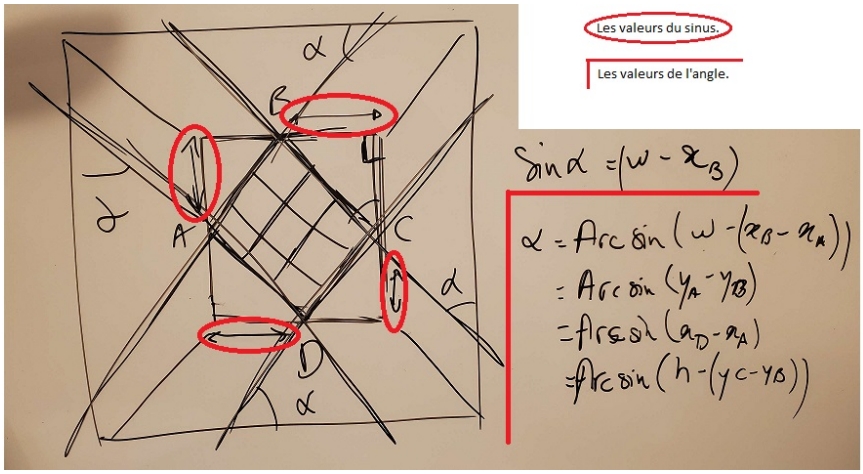


FIG. 22 – *Comment trouver l'angle de rotation (schéma personnel)*

```
// 4 values for the angle of rotation
int array[4] = {
    A->y - B->y,
    D->x - A->x,
    h - (C->y - B->y),
    w - (B->x - A->x)
};
```

(a) *Les quatre valeurs du sinus*

```
ASIN(get_mean(array, 4))
```

(b) *Calcul de l'arcsin*

FIG. 23 – *Calcul de l'angle*

Toutefois, une erreur que je n'ai pas réussi à trouvé dans le code nous empêche d'utiliser cette rotation automatique à ce jour (Segmentation fault).

5.2.2.3 Extraction des cases de la grille et appel du réseau de neurones

En ce qui concerne le découpage des sous-images des chiffres de la grille, c'est la fonction « cutting() » de Joric qui s'en charge à partir des coordonnées des points en haut à gauche et en bas à droite de la case à extraire. Je devais donc trouver ces coordonnées. Dans ce but, et en considérant l'image droite, j'ai utilisé des histogrammes sur les lignes et les colonnes de la matrice de pixel afin de trouver les pics et d'en déduire les coordonnées (indexes des pics dans les deux histogrammes). Ensuite, je n'avais plus qu'à itérer sur chaque case pour la découper, et lancer le réseau de neurones. (*Fig. 24*).

Enfin, il fallait retranscrire les résultats dans un fichier texte pour le solveur. Ainsi, la fonction stock les valeurs de retour dans un tableau et itère sur ce tableau rempli pour créer le fichier (*Fig. 25*).

Malheureusement, du fait de multiples contre-temps et d'une impossibilité de compiler, je n'ai pas réussi à tester cette partie de code et elle n'est donc pas implémenté dans le rendu final. Cette fonction étant celle qui lie le prétraitement avec le réseau de neurone et le solveur, notre rendu ne peut pas faire ce lien.

```c
// Get the 10 peaks to have X and Y coordinates
int maxX[GRID_SIDE+1], maxY[GRID_SIDE+1];
for(size_t i = 0; i < GRID_SIDE+1; i++){
    maxX[i] = get_max_removed(histoX, cols);
    maxY[i] = get_max_removed(histoY, rows);
}

// Get int values of sub_images
SDL_Surface* img = NULL;
int array[NB_VALUES];
for(size_t i = 0; i < GRID_SIDE; i++){
    img = cutting(maxX[i], maxY[i], maxX[i+1], maxY[i+1], surface);
    array[i] = compute(img, nn);
}
SDL_FreeSurface(img);
```

FIG. 24 – *Calcul des coordonnées, itération sur les cases, extraction des sous-images et lancement du réseau de neurones*

```c
// Print in file
FILE* fp = fopen(filename, "w");
for(size_t i = 0; i < NB_VALUES; i++){
    if(i % GRID_SIDE == 0)
        fputs("\n", fp);
    else if(i % SQR_SIDE == 0)
        fputs(" ", fp);
    array[i] ? fputc(array[i], fp) : fputs(".", fp);
}
fclose(fp);
```

FIG. 25 – *Rédaction des résultats dans un fichier texte*

5.2.2.4 Ségmentation du code

Afin d'éviter les répétitions de fonctions ou de macros, j'ai passé un peu de temps à organiser le code assez long du pré traitement de l'image. J'ai créé un fichier « Utils.c » qui contient la quasi totalités des fonctions et macros de notre code qui sont utilisées à plusieurs reprises (*Fig. 26 et 27*).

```c
// Double value of the pi math constant
#define M_PI 3.14159265358979323846

// Value for a black color
#define BLACK 0

// Value for a white color
#define WHITE 255

/** Get the max between x and y. ...
#define min(x, y) (x < y ? x : y)

/** Get the min between x and y. ...
#define max(x, y) (x > y ? x : y)

/** Get the absolute value of x. ...
#define abs(x) (x > 0 ? x : -x)

/** Get the sine value of x from degree to radian. ...
#define SIN(x) sin(2*M_PI*x/360)

/** Get the cosine value of x from degree to radian. ...
#define COS(x) cos(2*M_PI*x/360)

/** Get the arcsine value of x from radian to degree. ...
#define ASIN(x) 360*asin(x)/2*M_PI

/** Get the arccosine value of x from radian to degree. ...
#define ACOS(x) 360*acos(x)/2*M_PI
```

FIG. 26 – *Les macros du code*

```c
Uint8 get_pixel_red(Uint32 pixel_value, SDL_PixelFormat* format);
Uint32 pixel_to_color(SDL_PixelFormat* format, Uint8 color);
Uint32 pixel_to_colors(SDL_PixelFormat* format, Uint8 red, Uint8 green, Uint8 blue);
void get_pixels_values(Uint32* pixels, Uint32* values, size_t* len);
void printMat(size_t* M, size_t rows, size_t cols);
void printArr(size_t* array, size_t len);
void Swap32(size_t* a, size_t* b);
void Swap8(uint8_t* a, uint8_t* b);
void insertSort(Uint8* array, size_t size);
int get_mean(int* array, size_t size);
int get_max_removed(int* array, size_t size);
```

FIG. 27 – *Les fonctions du code*

5.2.3 Alexiane

5.2.3.1 Rotation Manuelle

Concernant la rotation manuelle pour réaliser celle-ci j'ai dû implémenter des fonctions annexes telles que pour récupérer un « pixel » pour remplacer un « pixel » et ensuite dans la fonction rassemblant tout cela appelé « rotate_angle » implémenter la rotation manuelle en utilisant donc les fonctions précédentes. Cette fonction prends en paramètre une « SDL surface » et un angle. dans un premier temps on pose la condition que l'angle ne peut pas tourner de plus de « 360° ». Ensuite, nous calculons le « sin » et le « cos »

de l'angle. Puis on calcule la taille de la surface passée en paramètre. Ensuite, on calcul la nouvelle surface en fonction de l'ancienne, puis on calcule la taille de la nouvelle surface. Par la suite, on crée la nouvelle surface. Il faut penser à verrouiller les surfaces à modifier et rechercher la nouvelle surface. Et enfin, avant de « free » l'ancienne surface et de déverrouiller après les changements les surfaces. Pour chaque « pixel » de la nouvelle image, j'associe le « pixel » correspondant de l'image d'origine.

Grace à tout ce processus on obtient une nouvelle image plus précisément une nouvelle surface tourné de l'angle passé en paramètre.

5.2.3.2 Résolution de la grille

La résolution de la grille avait été terminé à la première soutenance. Cependant nous avons dû régler quelques problèmes qui sont apparus lors de la mise en commun de cette partie avec l'interface graphique car nous avons eu des conflits. Nous avons dû donc améliorer la partie qui correspondait à la résolution de la grille. Les problèmes étaient dû à des conflits de nom de fonction ou à des includes. En effet il n'y avait pas eu de problèmes du côté du fonctionnement du « solveur ». Nos fonctions fonctionnaient individuellement mais elles ne fonctionnaient pas dans l'interface graphique dû aux causes que j'ai énoncé précédemment. Cette partie a donc été assez rapide à faire cependant elle était obligatoire.

Petite précision la résolution de la grille nous renvoie un fichier « .txt » qui contient la solution cette solution porte le même nom que le fichier reçu en entrée cependant il faut savoir que la solution sera effacée après un « make clean » contrairement à la solution finale contenu sous format image qui est renvoyé par la fonction qui affichage l'image résolue soit « Solution_show ».

5.2.3.3 Chargement d'une image

Concernant le chargement d'une image, en effet dans notre interface graphique nous devons tout d'abord charger une image sachant que toutes nos fonctions utilisent « SDL » il faut pouvoir transformer image reçue en une« SDL surface ». C'est pourquoi nous avons dû coder une fonction qui effectue cela et que nous utilisons donc dans notre « GUI » pour charger les images. Cependant nous avons aussi dû utiliser « SDL_SaveBMP » pour pouvoir retransformer l'image qui est une « SDL surface » en un fichier image soit un fichier avec l'extension « .bmp ». Toutes ces fonctions nous permettent donc de charger une image et de la transformer et de pouvoir

l'afficher et de la sauvegarder par la même occasion. Précision nous avons aussi dû faire une fonction qui extrait l'extension d'un fichier pour savoir s'il s'agit d'une image ou pas. Cette fonction nous sert à afficher dans notre « GUI » que des images avec les extensions « .jpg » et « .png » .

5.2.3.4 Reconstruction de la grille et affichage de la grille résolue

Cette partie correspond à la finalité de notre projet. Pour réaliser cette partie nous avons implémenté plusieurs fonctions. Tout d'abord, nous appelons la grille reçu par la fusion de la partie « Neural Network » et la partie prétraitement d'images. En effet ces deux parties nous renvoient un fichier « txt. » ce fichier à la même construction que les fichiers que nous devons résoudre dans la partie résolution de grille. C'est pourquoi au début de la fonction principale qui appelle toutes les fonctions utiles à l'affichage de la grille résolue nous devons avant tout convertir le fichier en une matrice de 9 par 9 puis de la copier et de la résoudre en remplaçant les zéros par les bons chiffres après ces étapes là nous avons donc une matrice contenant l'agrile résolue et une matrice contenant la grille non résolue.

Après cette étape, nous nous attaquons à la partie visuelle. En effet nous avons choisi d'afficher la solution sous la forme d'une image. Pour cela nous avons téléchargé une image de grille de sudoku vide ainsi que des chiffres de couleur noire. Il faut savoir que la grille fait du 720 par 720 et que les chiffres font du 80 par 80. Nous ne voulions pas avoir des problèmes de redimensionnement. Une fois que nous avons toutes nos images et que nous les avons chargés à l'aide de la fonction « load_image ».

Ensuite nous utilisons nos deux matrices pour déterminer quel chiffre nous devons mettre tout d'abord mais également la couleur que doit prendre ce chiffre. Pour pouvoir réaliser tout cela, j'ai du implémenter plusieurs fonctions sur les « pixels » tel que récupérer un « pixel » remettre un pixel. Ces fonctions sont également utilisées pour la rotation manuelle. j'ai également créer une fonction pour savoir si un « pixel » était déjà de couleur noire. L'assemblage de toutes ses fonctions m'a permis de créer une autre fonction qui va m'écrire le bon chiffre est de la bonne couleur dans la bonne case soit la fonction « printNumber ». J'appelle cette fonction dans la fonction principale de l'affichage de la grille résolue.

Petite précision pour savoir si un chiffre doit être en bleu ou en noir j'ai initialisé une variablee « inblue » que j'appelle en paramètre de la fonction « printNumber ». Cela nous permet donc d'obtenir une image plus

précisément une « SDL surface » qui contient la solution de la grille qui nous avait été renvoyée. C'est dans cette même fonction que j'ai effectué la sauvegarde de la grille résolue. Et grâce à cette sauvegarde qui me permet de convertir une « SDL surface » en un fichier avec l'extension « .bmp », je peux alors récupérer ce fichier pour l'afficher dans l'emplacement réservé à cela dans mon « GUI ».

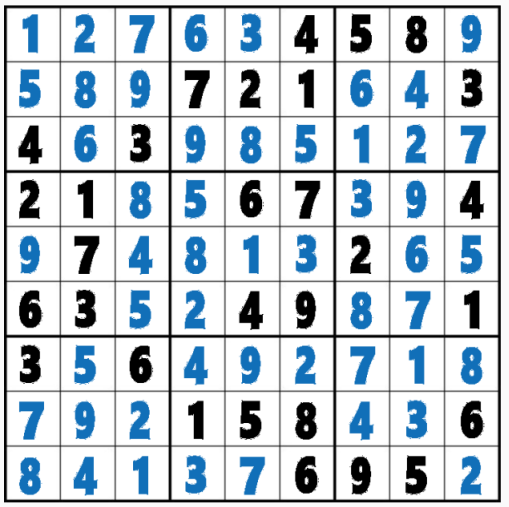


FIG. 28 – *Affichage de la solution*

5.2.3.5 Sauvegarde de la grille résolue

Comme dit dans la partie précédente c'est dans la partie de l'affichage de la grille que je sauvegarde la grille résolue, en effet pour cela je convertie simplement notre image qui est initialement en « SDL surface » en un fichier avec l'extension « .bmp ». Je sauvegarde ce fichier dans le dossier « Solution ».

Petite précision il s'agit du seul fichier qui n'est pas supprimé après un « make clean ». Cependant si on réeffectue le même processus l'image de la nouvelle solution écrasera l'image de l'ancienne solution car elles ont forcément les deux le même nom.

5.2.3.6 Interface Graphique (GUI)

Concernant cette partie il s'agit de la partie qui regroupe toutes les parties. Pour effectuer celle-ci nous avons utilisé GTK, ainsi que Glade[8]. il s'agit de

8. `https://glade.gnome.org/`

logiciels pour interface graphique.

- Glade :

Tout d'abord nous avons dû réfléchir au visuel de notre interface graphique. Une fois que cela a été fait nous avons utilisé Glade pour implémenter les boutons et autres outils dont nous avions besoin dans notre « GUI ». Notre interface se décompose en deux parties il y a la partie qui s'intéresse au traitement d'image et à la solution et il y a l'autre partie qui s'intéresse au réseau de neurones. Nous voulions implémenter également le réseau de neurones dans notre interface graphique car nous trouvions cela intéressant.

- GTK :

Tout d'abord j'ai dû effectuer des recherches sur cette partie car il s'agit de quelque chose que je ne connaissais pas. Nous avons cependant eu un TP sur cette partie mais en fin d'année donc j'avais déjà commencé cette partie cependant ce TP m'a également aidé. En effet avant toute chose nous avons dû appeler dans notre fichier « ocr.c » notre fichier Glade soit « ocr.glade », mais également tous les « includes » nécessaires.

Premièrement, nous avions déjà implémenter les boutons à l'aide de Glade, nous avions plus qu'à les relier a des fonctions. En effet le principe de ces boutons est d'effectuer une fonction en fonction du bouton appelé. Nous avons donc aussi implémenté quelques fonctionnalités tel que l'accès à un bouton que si un autre bouton avait déjà été cliqué. Par exemple nous pouvons pas appuyer sur le bouton « Solution » tant que nous avons pas appuyé sur le bouton « Solve ». Et par exemple nous pouvons pas accéder au bouton de la partie traitement d'image tant que le fichier passé en entrée n'est pas valide.

De plus, j'ai également apporter quelques détails supplémentaire tel qu'un titre, une taille minimum de fenêtre, un placement précis des boutons etc.

Précision sur l'aspect technique de GTK pour pouvoir utiliser les boutons nous avons tout d'abord dû les initialiser. Nous avons aussi du récupérer les widgets à l'aide de « gtk_builder_get_object » puis pour les appeler nous avons utilisé la fonction « g_signal_connect ». En effet l'utilisation de GTK nécessité beaucoup de recherches cependant une fois les recherches effectuées la prise en main de GTK devient très facile. Et aussi il ne faut pas oublie de

tout quitter à la fin de l'exécution de notre « GUI » soit dans le « main », je « free » le réseau de neurones et on quitte GTK et SDL.

En effet cette partie n'est pas la plus simple car elle dépend des autres parties. Tant que les autres parties ne sont pas faites je ne peux pas les implémenter dans l'interface graphique d'où le fait qu'il manque quelques parties dans l'interface graphique malgré que j'ai essayé de simuler ces partis. De plus, il faut que le code de nos camarades soit absolument commenté car dans la partie interface graphique j'ai du appelé chaque fonction correspondant au bouton voulu. Sachant qu'il peut y en avoir plusieurs pour un seul bouton.

Il faut savoir que toutes les fonctions qui ne reçoivent pas un bouton en entrée sont appelés que dans l'interface graphique dans les fonctions qui elles ont en entrée un bouton. En effet j'ai codé des fonctions supplémentaires pour faciliter la création de l'interface graphique. L'interface graphique n'appelle pas juste les fonctions issues des autres dossiers elle doit convertir des images, des fichiers mais également envoyer des messages d'erreurs faire attention aux extensions il y a plein de détails à regarder ce qui rend cette partie très complexe.

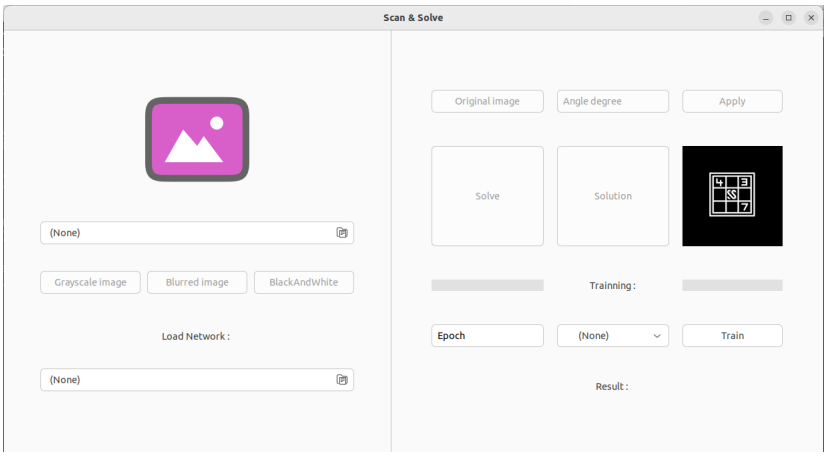


FIG. 29 – *Interface graphique*

- Traitement d'image :

Concernant la partie traitement d'images, tout d'abord nous avons implémenté un bouton qui permet de récupérer un fichier et qui affiche un message d'erreur si celui-ci n'est pas dans la bonne extension lorsque le fichier est validé alors ils s'affichent sur le bloc prévu pour l'affichage.

Ensuite nous avons accès à quatre boutons qui permettent de changer l'aspect visuel de l'image c'est à dire que nous pouvons la mettre en nuances de gris la flouter et la binariser soit la mettre en noir et blanc et nous pouvons aussi retourner à l'image originale.

Il y a également la partie pour tourner l'image pour cela nous avons implémenté un bouton qui récupère un texte donné ce texte est écrit par l'utilisateur dans ce bouton si ce texte n'est pas un chiffre ou plus précisément un nombre lorsque nous appuyons sur le bouton « Apply » à qui est relié donc au bouton contenant le texte alors nous avons un message d'erreur qui apparaît jusqu'à ce que l'utilisateur rentre un nombre. Lorsque le texte est validé alors lorsque nous appuyons sur le bouton « Apply » l'image tourne de l'angle donné. Petite précision le texte rentre par l'utilisateur plus précisément le nombre correspond à l'angle.

Ensuite nous avons un bouton « Solve » qui permet de résoudre le fichier reçu une fois qu'il sera passé par le réseau de neurones et l'image prétraitement. Ce bouton n'affiche rien cependant lorsque celui-ci est cliqué le bouton « Solution » est activé et lorsque nous cliquons sur le bouton « Solution » notre solution s'affiche et par la même occasion elle est sauvegardée dans le dossier « Solution » sous format « .bmp ».

Il y a aussi une partie où nous affichons notre logo.

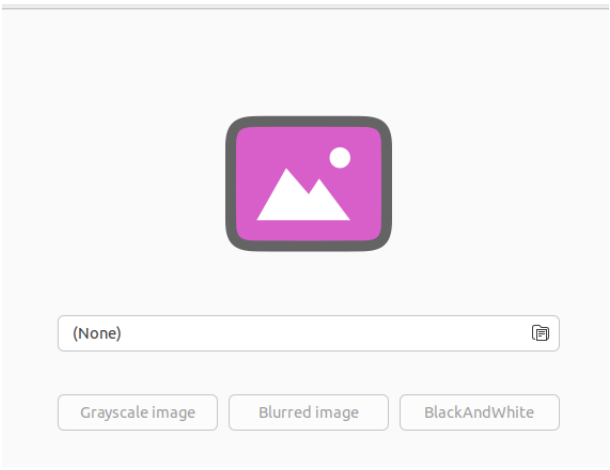


FIG. 30 – *Le traitement et le chargement d'image*

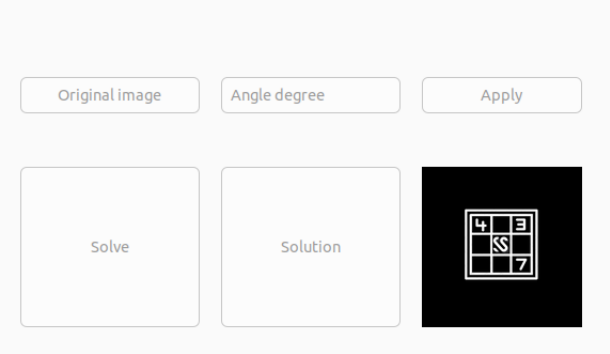


FIG. 31 – *Le traitement d'image, l'affichage et la résolution du sudoku*

- Réseau de neurones :

Concernant la partie réseau de neurones qui se situe en bas de l'interface graphique nous avons tout d'abord une partie qui récupère un fichier donné par l'utilisateur si ce fichier n'est pas en extension « .nn » alors un message d'erreur apparaît jusqu'à ce que le fichier donné soit valide. Lorsque celui-ci est validé rien ne s'affiche sur l'interface graphique cependant dans la console nous avons tous les résultats du réseau de neurones.

Ensuite sur la partie droite de l'interface graphique nous avons plusieurs boutons nous avons tout d'abord un bouton où il y a noté « Epoch » il s'agit du nombre d'époque que va effectuer le réseau de neurones. Ensuite petite précision comme dans la partie image prétraitement il faut que le texte rentré sur ce bouton soit un nombre sinon un message d'erreur apparaît. Ensuite

il y a un autre bouton pour récupérer un dossier il s'agit du dossier dans lequel lorsque nous appuierons sur le bouton « Train » le fichier de résultat ira se stocker. Le fichier résultat apparaît sous le nom « result.nn ». De plus nous ne pouvons pas appuyer sur le bouton« Train » si il n'y a pas de dossier donné et si le bouton correspondant aux nombres d'époques n'a pas un nombre en entrée. Ensuite nous avons tout en bas de l'interface graphique à droite un texte où y a écrit « Résult : » . En effet lorsque nous appuyons sur le bouton « Train » lorsque la fonction a fini d'effectuer son travail alors le résultat s'affichera à côté du texte « Résult : ».

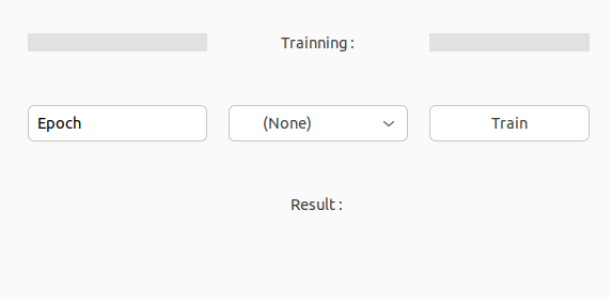


FIG. 32 – *La partie « trainning » du réseau de neurone*

- Makefile :

Pour finir, il ne faut pas oublier le fichier indispensable qui compile tout notre projet le « Makefile ». Tout d'abord, celui contient tous les librairies dont nous avons besoin. Puis il compile tous les fichiers nécessaires à l'utilisation du « GUI ». Et enfin, il supprime à l'aide d'un « make clean » tous les fichiers créer lors de l'utilisation du « GUI » et de sa compilation.

5.2.4 Joric

5.2.4.1 Création d'un « dataset »

Pour la partie « dataset » du réseau de neurones, nous avons commencé par utiliser le MNIST. Le MNIST est un « dataset » comprenant 60 000 images de chiffres dessinés à la main ainsi qu'un « dataset » d'entraînement de 10 000 images. Grâce à se « dataset », j'ai ainsi pu tester mon réseau afin de vérifier qu'il était fonctionnel et efficace. Ensuite, j'ai dû me lancer dans la

création de notre propre « dataset » dans le but d'avoir un réseau de neurones plus efficace. Pour ce faire, j'ai réalisé un programme python qui prend toutes les images d'un dossier, les affiches et demande à l'utilisateur le chiffre sur l'image. Ainsi, je pouvais créer mon propre « dataset » à l'aide des images extraite de sudokus. Cependant, tout ne s'est pas passé comme ce qui était prévu. En effet, pour avoir un « dataset » efficace, il est important de le créer à partir d'image provenant de notre traitement d'image. Or, la détection des cases de la grille n'ayant pas été finit, je n'ai pas pu réaliser de « dataset » personnalisé. Néanmoins, on peut voir que le réseau fonctionne, car en un entraînement de 15 secondes sur le MNIST, on obtient une précision de 85 % ce qui est très intéressant étant donné le peu de temps d'entraînement.

5.2.4.2 Sauvegarde et chargement des poids

Pour la partie concernant le chargement des poids et biais ainsi que leur sauvegarde, cela fut très rapide à réaliser, car il suffit de parcourir tous nos neurones et d'écrire dans un fichier les valeurs des variables. Pour le chargement, cela pourrait sembler plus compliqué, puisque nous avons enregistré des nombres à virgule, mais grâce à la fonction « fscanf » il suffit de préciser le type de valeur à lire dans le fichier et la fonction fait les conversions pour nous.

5.2.4.3 Conclusion de la partie réseau de neurones

Pour ce qui est du réseau de neurones, il est totalement fonctionnel et donnent de très bons résultats. Cependant, dû au retard pris par la partie détection de la grille, cette partie n'est pas utile, car elle n'a pas pu être implémentée dans le projet.

5.2.4.4 Découpage de la grille

À cause du retard pris par la partie détection de la grille, je me suis occupé du découpage de la grille et du passage des images dans l'IA afin de réduire la charge de travail de mes coéquipiers. Cette partie n'est pas très complexe, il faut toutefois faire attention au fait que les images extraites ne soient pas collées au bord. Une fois le décalage calculé, il reste simplement à parcourir l'image en tenant compte de ce décalage puis copié la valeur des pixels dans une nouvelle image. Concernant le passage des images dans le réseau, nous avons décidé de passer toutes les images extraites dans la fonction. Elle doit donc convertir l'image en liste et calcule ensuite le pourcentage de pixels noirs présents dans l'image. Si cette valeur est inférieure à 10 %, alors la case

est vide et la valeur retournée est 0. Sinon, on revoit le résultat du passage de l'image dans l'IA.

5.2.5 L'assemblage du projet

Concernant l'assemblage du projet, nous avons mis notre travail en commun par le biais de l'interface graphique principalement ou dans chaque partie individuellement selon les besoins.

- GUI :

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

$ make

Pour pouvoir lancer le « GUI » nous utilisons la commande suivante :

$./ocr

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

$ make clean

6 Récit de la réalisation

6.1 Ekaterina

Dès le début du projet, je savais que la partie que j'avais choisie serait assez complexe à réaliser toute seule, c'est pour cela que dans le groupe nous nous étions mis d'accord pour travailler à deux sur le traitement de l'image et des détections. Je suis très fière des tâches que j'ai accomplies avec le temps que j'ai passé dessus bien que toute la partie n'ait pas été terminée et ce sûrement à cause du choix du binôme. Ce projet m'a permis de comprendre l'importance des recherches que l'on doit effectuer et de l'autonomie que l'on doit apporter.

6.2 Abel

En dépit de certaines difficultés pratiques, je pense que ce projet aura été mon préféré depuis mon entrée à l'EPITA. En effet, durant ce projet et malgré plusieurs tentatives, je n'ai pas eu la possibilité de tester mes codes directement sur mon ordinateur personnel et devait donc utiliser les ordinateurs de l'école ou appeler un autre membre du groupe à l'aide. Ceci a donc beaucoup diminué mon efficacité de travail. De plus, il a fallut consacrer un temps important à la recherche d'algorithmes déjà utilisé pour les différentes étapes du pré traitement de l'image.

Cependant, j'ai trouvé plus intéressant de suivre le cahier des charges d'un projet concret au projet libre. Si le travail de groupe a été très important, surtout avec Katia, j'ai néanmoins trouvé le travail individuel sur ce projet assez conséquent, l'organisation de chacun y étant probablement pour quelque chose.

Grâce à ce projet, j'ai eu la possibilité de travailler sur le traitement d'image avec Katia et d'aider Joric et Alexiane sur le réseau de neurones et l'interface graphique, ce que j'ai trouvé très instructif et espère pouvoir renouveler à l'avenir.

6.3 Alexiane

Concernant la réalisation de ce projet celui-ci n'etait pas de tout repos. En effet comme il s'agit d'un projet imposé donc avec un cahier des charges à suivre on ne peut pas contourner tous les problèmes il faut donc se débrouiller pour trouver des solutions. C'est pourquoi ce projet nous oblige à faire des

recherches précises.

Il faut donc savoir bien se renseigner en choisissant les bons mots clés par exemple. De plus malgré qu'il s'agit d'un travail de groupe on ne peut pas toujours demander de l'aide à ses camarades car eux aussi ont leur partie à réaliser et d'un autre côté peut-être qu'il n'ont pas toujours la réponse car en effet chaque partie du projet ne fait pas référence forcément à des éléments que nous avons vus en cours. C'est pourquoi avant de pouvoir commencer une partie du projet il faut faire des recherches. C'est pour cela que les autres membres du groupe ne sont pas forcément aptes à nous aider. Nos camarades sont plus là pour nous aider dans les parties rédactions de code, réglages de problèmes, soit dans tout ce qui correspond à l'aspect technique du code et non à la partie connaissance d'un domaine.

D'un autre côté, malgré tous les problèmes rencontrés, ce projet est un projet très intéressant car tout d'abord il insiste sur le fait d'être autonome mais également de savoir travailler en groupe. Concernant l'aspect plus technique ce projet est très intéressant car tout d'abord il regroupe plusieurs parties tel que le réseau de neurones soit l'IA mais également toute la partie interface graphique. Pour moi, ce projet est donc un projet très complet sur lequel il était très plaisant de travailler car tout d'abord les parties que j'ai effectué sont des parties qui correspondent à des domaines qui m'intéressent. En effet tout ce qui correspond à du code ayant un rendu visuel me plaît beaucoup c'est également pour cela que j'ai choisi de réaliser l'interface graphique et aussi le rendu de la solution avec le contraste de couleurs entre les chiffres déjà présents et les chiffres issues de la solution.

Outre l'aspect scolaire, ce projet nous montre ce que nous devons réaliser dans le monde du travail soit d'avoir des projets à faire et à rendre dans un délai donné. Enfin, la réalisation de ce projet m'a beaucoup plu malgré qu'il n'a pas été de tout repos.

6.4 Joric

Malgré le fait que le projet n'a pas pu être fini dans les temps, je suis relativement satisfait de ce que j'ai pu produire. Le réseau de neurones est parfaitement fonctionnel et j'ai acquis de nombreuses connaissances sur le sujet. J'ai également compris que le choix de ces coéquipiers et une des parties la plus importante du projet. En effet, s'ils ne sont pas motivés ou que l'on ne s'entend pas entre nous, il est presque impossible de réussir à finir le projet. Cela reste tout de même une bonne expérience que je n'oublierais pas

et qui m'aura appris de nombreuses choses.

7 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. Certes, cette aventure n'a pas été de tout repos mais il s'agit d'une expérience enrichissante. Pour finir, La plus belle chose de ce projet est de voir celui-ci se concrétiser au fur et à mesure du temps. Ce sentiment de satisfaction est incroyable.