Rapport de soutenance - Projet S3
Première soutenance

- Projet OCR -

Mikhaël DARSOT
Ethan GIRARD
Tom CASCHERA
Alexis MERLE

EPITA - Novembre 2022

Table des matières

Chapitre 1

Introduction

Dans ce premier rapport, le but sera d'expliciter de manière claire les avancées qui ont été faites sur le projet OCR par notre groupe, qui est composé de Mikhael Darsot, Ethan Girard, Tom Caschera et Alexis Merle.

Pour rappel, le but de ce projet est, comme son nom l'indique, de réaliser un logiciel type OCR (Optical character recognition), qui résout une grille de Sudoku.
Dans sa version définitive, notre application proposera une interface graphique permettant de charger une image dans un format standard, de la visualiser, de corriger certains de ses défauts, et enfin d'afficher la grille complètement remplie et résolue. La grille résolue pourra également être sauvegardée.

La présentation se fera de manière thématique. Nous allons présenter les différents éléments qui nous ont été imposés de terminer pour cette soutenance, c'est à dire : le chargement de l'image, la suppression des couleurs, la rotation de l'image, la détection de la grille et des cases, le découpage de l'image, le réseau de neurones, et enfin l'implémentation de l'algorithme de résolution du Sudoku.

Chapitre 2

Présentation des membres du groupe

2.1 Mikhael Darsot (Chef de projet)

Passionné par l'informatique depuis mon plus jeune âge, je suis toujours enchanté par le fait de travailler sur un nouveau projet. Le projet OCR est l'occasion pour moi de travailler sur quelque chose de tout nouveau. En effet, mis à part le TP de programmation que nous avons eu sur les images, je n'ai jamais fait de traitement d'image et c'est donc une première pour moi.

2.2 Alexis Merle

Etant premièrement fan de jeux vidéo, je me suis intéressé alors assez vite à l'informatique en étant jeune et plus particulièrement à la programmation. Ce projet se présente comme une opportunité pour progresser fortement en C étant complètement débutant dans ce language, mes seules expériences avec celui-ci sont les TP de début d'année proposés par l'école. Je n'ai donc que très peu de connaissances, que je pourrais facilement enrichir grâce au projet OCR.

2.3 Ethan Girard

Etant issu d'une terminale scientifique, j'ai été pris de passion par l'informatique en classe de première avec la spécialité "Numérique et sciences informatiques". Créer un projet de A à Z me plait beaucoup, avoir un outil fonctionnel qui permet de passer d'une image à une chose plus concrète me plaît encore plus. Il est vrai que le projet OCR est très enrichissant car il traite d'un sujet inconnu pour moi auparavant.

2.4 Tom Caschera

J'ai commencé à m'intéresser à l'informatique dès mon plus jeune âge. J'ai commencé à jouer à de plus en plus de jeux pendant mon temps libre par la suite et c'est progressivement devenu ma passion. Je commence à accumuler beaucoup d'expérience dans le domaine de la programmation, notamment avec C et Python. Ce projet me semble très intéressant pour apprendre le C et surtout pour mieux comprendre le traitement d'image.

Chapitre 3

Répartition des tâches

Afin d'être efficace dans notre travail, nous nous sommes mis d'accord sur une répartition de travail pour cette première soutenance, la voici :

	Mikhael	Ethan	Tom	Alexis
Chargement de l'image	X			
Suppression des couleurs	X			
Rotation de l'image	X			
Détection de la grille et des cases			X	
Découpage de l'image			X	
Réseau de neurones		X		
Algorithme de résolution du Sudoku				X

Chapitre 4

Chargement d'une image et supression des couleurs

4.1 Chargement de l'image

Avant de penser à travailler sur une image, il s'agit de la charger, c'est à dire de pouvoir l'afficher. Pour cela, nous nous sommes appuyés sur le TP de programmation qu'on a eu concernant l'affichage et le traitement d'image. Dans ce TP, nous avons appris comment manipuler des images en utilisant la librairie SDL. Il s'agit de la librairie que nous allons utiliser tout le long de notre projet.

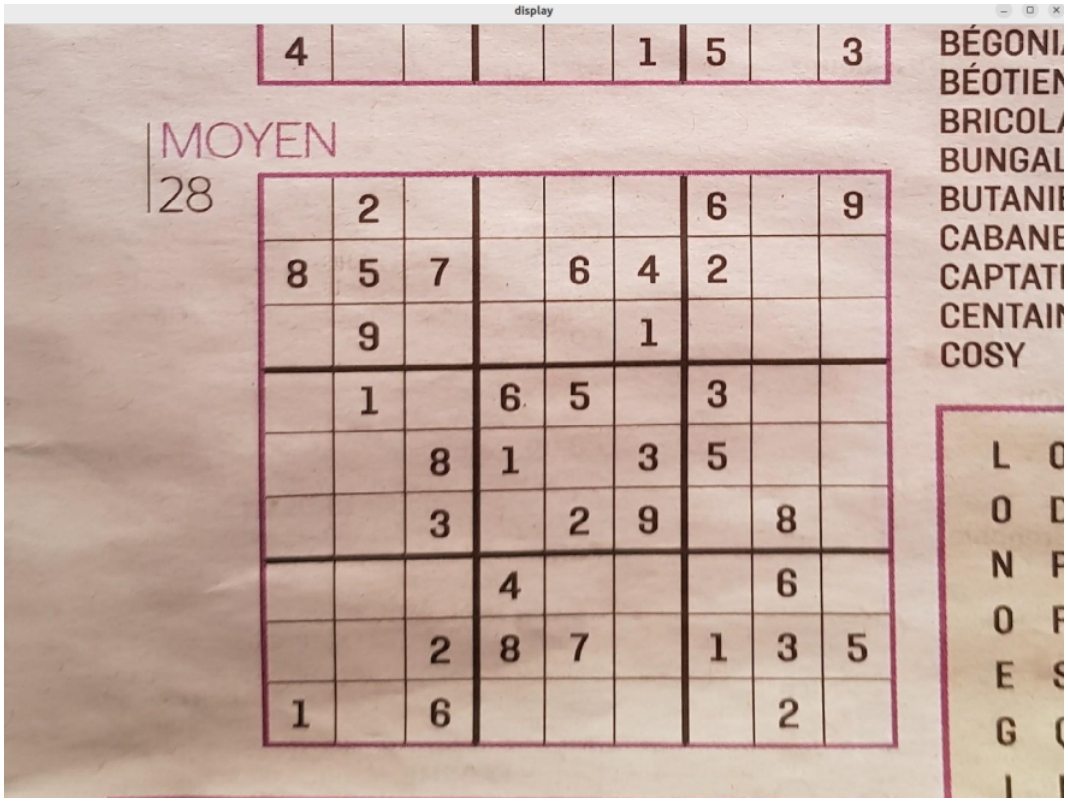
Concrètement, notre programme de chargement d'image va permettre d'afficher une image dans une fenêtre graphique afin de pouvoir la visualiser, de la redimensionner et de pouvoir refermer la fenêtre. Notre programme se compose en trois parties : la fonction draw qui va permettre de dessiner l'image, la fonction *event_loop* qui va permettre d'afficher l'image dans la fenêtre et de maintenir l'affichage jusqu'à temps d'avoir une commande qui lui dit de fermer la fenêtre. Enfin, nous aurons une fonction main qui aura pour but d'initialiser les différentes choses avant d'appeller les deux fonctions précédentes, puis les détruire après fermeture de la fenêtre.

Afin de charger une image, il y a plusieurs étapes : dans notre fonction Main, nous avons d'abord créé une fenêtre, cela est possible grâce à la fonction $SDL\_CreateWindow$, puis il a fallu créer un moteur de rendu, avec la fonction $SDL\_CreateRenderer$, et enfin une texture pour notre image via la fonction $IMG\_LoadTexture$.

Après avoir réalisé ces étapes, notre fonction main va appeler notre fonction *event_loop*, qui elle-même va appeler notre fonction draw, ce qui va permettre de dessiner la texture, puis de l'afficher dans la fenêtre. L'image restera affichée tant qu'on ne ferme pas

la fenêtre nous-même.

Finalement, lorsqu'on compile notre programme avec la commande "make", nous avons le fichier exécutable "display" qui se crée et nous pouvons l'éxecuter via la commande "./display images/source.jpeg", avec "source.jpeg" l'image qu'on souhaite afficher.



Annexe n°1 - Fenêtre graphique après avoir affiché notre image

4.2 Passage aux nuances de gris

Sur le même principe que notre programme qui permet d'afficher une image, nous devons cette fois-ci afficher une image qu'on aura passé en nuances de gris. Brièvement, pour cela, nous avons utilisé les surfaces SDL, afin de pouvoir cibler chaque pixel de notre image, et ainsi traiter notre image pixel par pixel pour effectuer des modifications.

Nous avons eu besoin de deux fonctions supplémentaires par rapport à notre fonction de chargement d'image, qui sont les fonctions $pixel\_to\_grayscale$ et $surface\_to\_grayscale$.

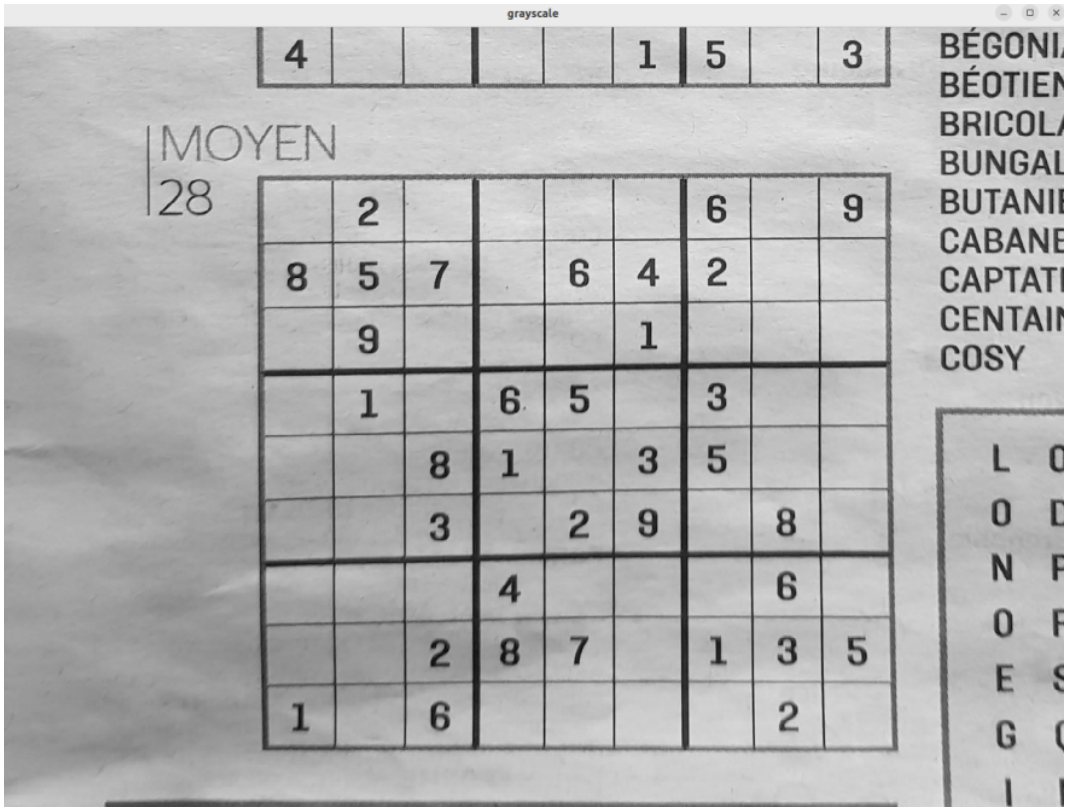
La fonction $pixel\_to\_grayscale$ va venir prendre les valeurs R,G,B (c'est à dire les valeurs correspondantes à la couleur) d'un pixel passé en paramètre, et à partir de ces valeurs, on va faire une moyenne des valeurs des couleurs (via une formule bien

précise), et ainsi obtenir la couleur que prendra le pixel lors de la mise en nuances de gris. Ainsi, on va renvoyer le pixel avec sa couleur modifiée.

```
average = 0.3*r + 0.59*g + 0.11*b
```

La fonction *surface_to_grayscale* quant à elle aura pour but de parcourir tous les pixels de la surface qu'elle aura en paramètre, et de leur appliquer la fonction *pixel_to_grayscale*, afin de changer la couleur de tous les pixels de l'image et les passer en nuances de gris.

Finalement, lorsqu'on compile notre programme avec la commande "make", nous avons le fichier exécutable "grayscale" qui ce crée et nous pouvons l'éxecuter via la commande "./grayscale images/source.jpeg", avec "source.jpeg" l'image que l'on souhaite afficher en nuances de gris.



Annexe n°2 - Fenêtre graphique après avoir affiché notre image en nuances de gris

4.3 Binarisation (mise en noir et blanc de l'image)

La binarisation de l'image est une étape cruciale dans notre notre projet, puisqu'elle est essentielle pour la détection des lignes de la grille. En effet, lorsqu'on a une image avec énormément de couleurs, il est assez dur de détecter des élements (dans notre cas des lignes) afin d'identifier ce qui nous intéresse. Avec la binarisation, on n'aura que deux couleurs pour notre image et donc cela permettra de suivre la couleur que l'on souhaite afin de mieux identifier les choses.

Pour cela, nous avons repris le même principe que pour la mise en nuances de gris, c'est à dire que nous avons travaillé avec des surfaces et nous avons apporté nos modifications directement pixel par pixel afin de changer la couleur. Nous avons également fait le choix (par préférence et par lisibilité) de représenter les lignes et les chiffres en blanc et le fond de la grille en noir. Ainsi, un pixel qui est censé être noir sera représenté en blanc par notre binarisation, et inversement.

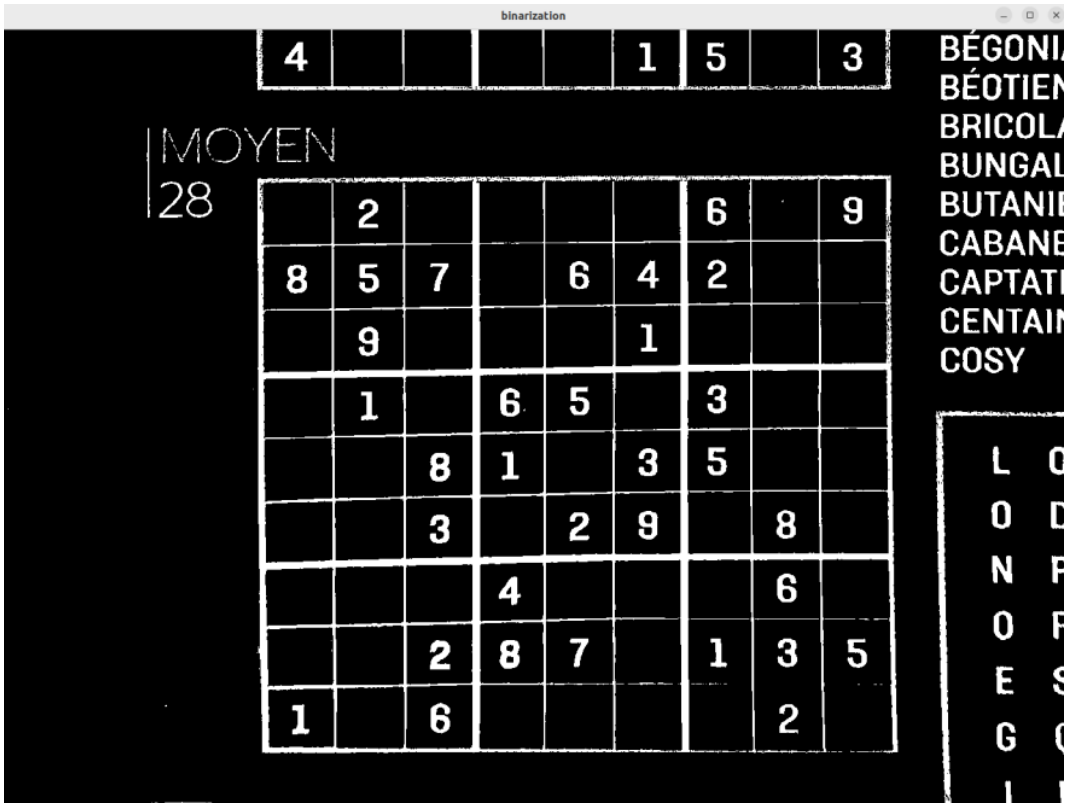
Notre programme est donc similaire à celui de la mise en nuances de gris, le changement est à la fonction $pixel\_to\_binary$, qui comme $pixel\_to\_grayscale$, effectue un changement de couleur sur un pixel passé en paramêtre. Cependant, à l'inverse de $pixel\_to\_grayscale$, nous n'avons pas de formule qui donnera la nouvelle couleur de notre pixel, mais un calcul encore plus simple : nous faisons la somme des 3 composantes R, G, et B, puis on divise par 3 et ainsi, on a la moyenne de la couleur du pixel. Le blanc étant à R=G=B=255, et le noir étant à R=G=B=0, si notre moyenne est supérieure à 127 (qui est la moitié de 255), alors le pixel est "plutot" blanc (on le mettra donc en noir) et si notre moyenne est inférieure à 127, alors le pixel est "plutot" noir (on le mettra donc en blanc).

Ainsi, avec cette implémentation, et avec la fonction $surface\_to\_binarized$, on pourra donc mettre toute notre surface en noir et blanc.

Finalement, lorsqu'on compile notre programme avec la commande "make", nous avons le fichier exécutable "binarization" qui se crée et nous pouvons l'éxecuter via la commande "./binarization images/source.jpeg", avec "source.jpeg" l'image qu'on souhaite afficher en noir et blanc.

Le résultat nous semble satisfaisant, la mise en noir et blanc fait bien apparaître les lignes et ca rendra les choses bien plus facile par la suite. Cependant, nous remarquons qu'il y a parfois des petits soucis et il y a des endroits où les pixels ne sont pas toujours bien représentés. Ce détail devrait être corrigé par la suite lorsqu'on continuera le pré-

traitement et qu'on travaillera sur l'élimination des bruits parasites (grain de l'image, tâches etc..)



Annexe n°3 - Fenêtre graphique après avoir affiché notre image en noir et blanc

Chapitre 5

Rotation manuelle de l'image

5.1 Début de l'implémentation

Lorsqu'on doit traiter une image pour pouvoir détecter une grille de sudoku, il faut que l'image soit dans le bon sens. C'est pour cela qu'un algorithme de rotation d'image est essentiel afin de permettre de faire une rotation sur l'image et ainsi la remettre droite pour faciliter le travail.

Dans un premier temps, il s'agit juste d'implémenter un algorithme qui effectue la rotation à partir d'un angle donné en argument, mais par la suite il faudra trouver un moyen de déterminer l'angle avec lequel on doit effectuer la rotation.

Encore une fois pour la rotation de l'image, nous nous sommes penchés sur les surfaces SDL afin de traiter les pixels un par un et ainsi pouvoir récupérer les informations pixels par pixels pour notre traitement.

Le principe de l'algorithme est le suivant : on va parcourir tous les pixels de notre surface, et pour chaque pixel que l'on rencontre, on va prendre sa valeur (c'est à dire sa couleur) et on va calculer les nouvelles coordonnées relatives à ce pixel en fonction de l'angle de rotation, puis on va mettre la couleur du pixel aux nouvelles coordonnées. En reproduisant cela pour tous les pixels, on aura donc fait une rotation sur tous les pixels de la surface, et donc on aura fait une rotation complète de la surface.

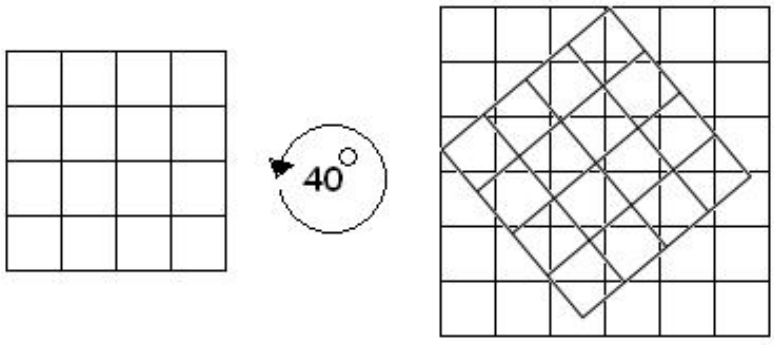
Pour l'implémentation de l'algorithme, nous avons donc eu besoin de deux fonctions intermédiaires, qui sont $Read\_Pixel$ et $Write\_Pixel$, et qui comme leur nom l'indique, ont pour but de respectivement lire et ecrire un pixel sur une surface donnée. Plus exactement, la fonction $Read\_Pixel$ a en paramètre une surface ainsi que les coordonnées x et y du pixel que l'on veut lire, et retourne la couleur du pixel (Uint32). La fonction $Write\_Pixel$ a quant à elle en paramètre une surface, deux coordonnes

(x,y) ainsi que un pixel (Uint32) qui devra être placé sur la surface.

Grâce à ces deux fonctions, nous avons pu écrire notre algorithme de rotation. Nous créons une surface vide (grâce à la fonction $SDL\_CreateRGBSurface$) de la même taille que la surface que l'on doit rotationner, et on place les pixels au bon endroit en appliquant la rotation de l'angle considéré sur les coordonnées des pixels de la surface de départ.

5.2 Problèmes rencontrés

Cependant nous avons rencontré un petit problème. En effet, nous créions d'abord une surface vide de la même taille que notre surface initiale et nous placions nos pixels dessus en fonction du traitement que l'on faisait sur la première surface. Or la taille prévue n'était pas toujours la bonne. En effet, lorsqu'on fait une rotation d'un carré de 40 degrés par exemple, la place que prendra le carré verticalement et horizontalement sera plus importante après la rotation qu'avant. Ainsi, nous avions des dépassements et donc toute la grille ne s'affichait pas vraiment comme nous le souhaitions.

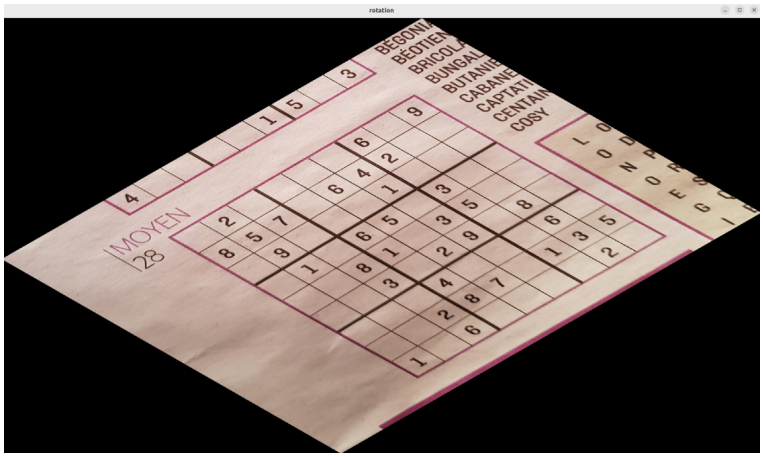


Annexe n°4 - Rotation d'un carré

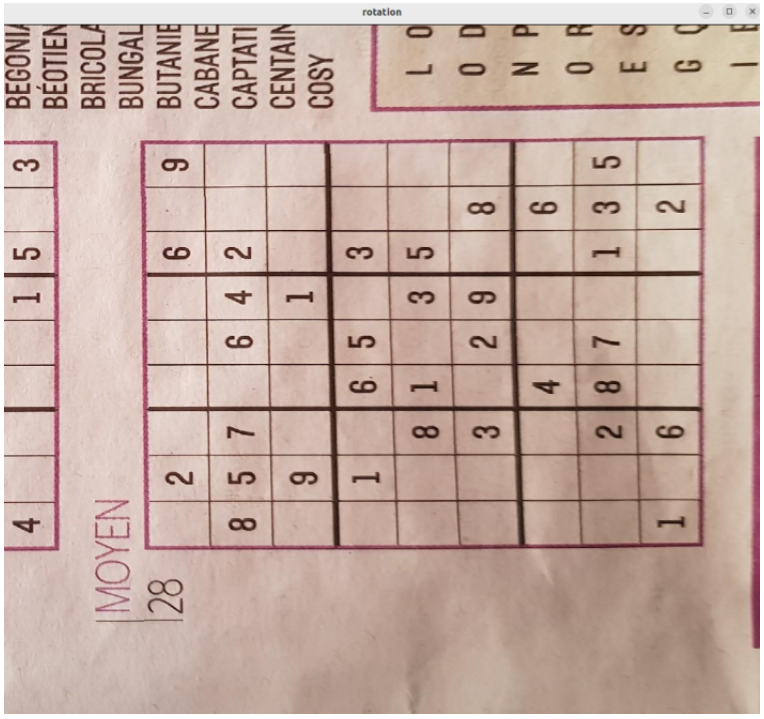
Pour palier à ce problème, nous avons du faire un calcul supplémentaire afin de créer une surface qui soit de la bonne taille et ainsi avoir toute l'image qui soit bien rotationnée (sans perte d'informations).

5.3 Fin

Finalement, lorsqu'on compile notre programme avec la commande "make", nous avons le fichier exécutable "rotation" qui se crée et nous pouvons l'éxecuter via la commande "./rotation images/source.jpeg angle", avec "source.jpeg" l'image qu'on souhaite rotationner et "angle" l'angle avec lequel on souhaite faire la rotation.



Annexe n°5 - Rotation de 45° sur notre image



Annexe n°6 - Rotation de 90° sur notre image

Chapitre 6

Détection de la grille et de la position des cases

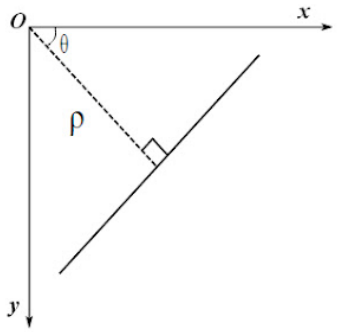
6.1 Detection des lignes

Pour détecter et récupérer les coordonnées de la ligne sur l'image du sudoku, nous avons utilisez l'algorithme de Hough.

Un algorithme très puissant mais aussi très complexe. Son fonctionnement est basé sur le fait que chaque ligne d'une image peut etre représentée par :

$$\rho = xcos(\theta) + ysin(\theta)$$

Avec ρ distance perpendiculaire de l'origine à la ligne.
Et θ l'angle formé par cette perpendiculaire et l'axe horizontal.

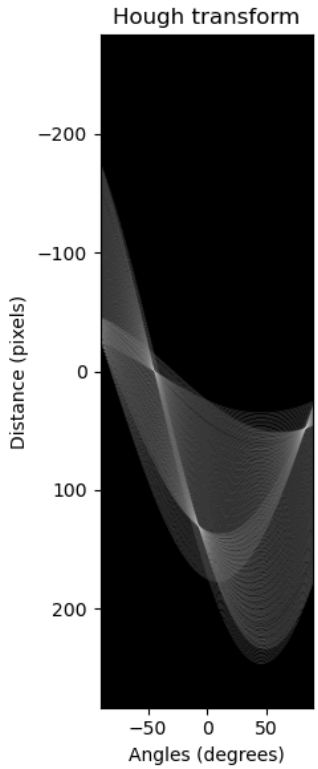


Annexe n°7 - Equation d'une ligne

Maintenant que nous savons que chaque point peut satisfaire cette equation, alors si nous possèdons chaque paire (ρ ,θ), nous devrions pouvoir tracer une ligne.

Nous allons donc créer une matrice, un accumulateur (Acc), (2D array) pour stocker chaque valeur de Rho et Theta.

Prenons par exemple un point A(x1,y1) sur une ligne. Alors on aura $\rho 1 = x1cos(\theta 1) + y1sin(\theta 1)$ et on pourra donc augmenter la valeur de Acc[ρ1][θ1] de +1.



Annexe n°8 - Hough Transform

Apres avoir tracé les points de notre accumulateur, nous nous retrouvons avec une image comme celle-ci où nous pouvons apercevoir des points qui ressortent plus blanc que les autres. Ce sont ces points qui sont donc nos lignes dominantes de notre image et que nous allons donc devoir récupérer pour ensuite tracer les veritables lignes sur l'image de base.

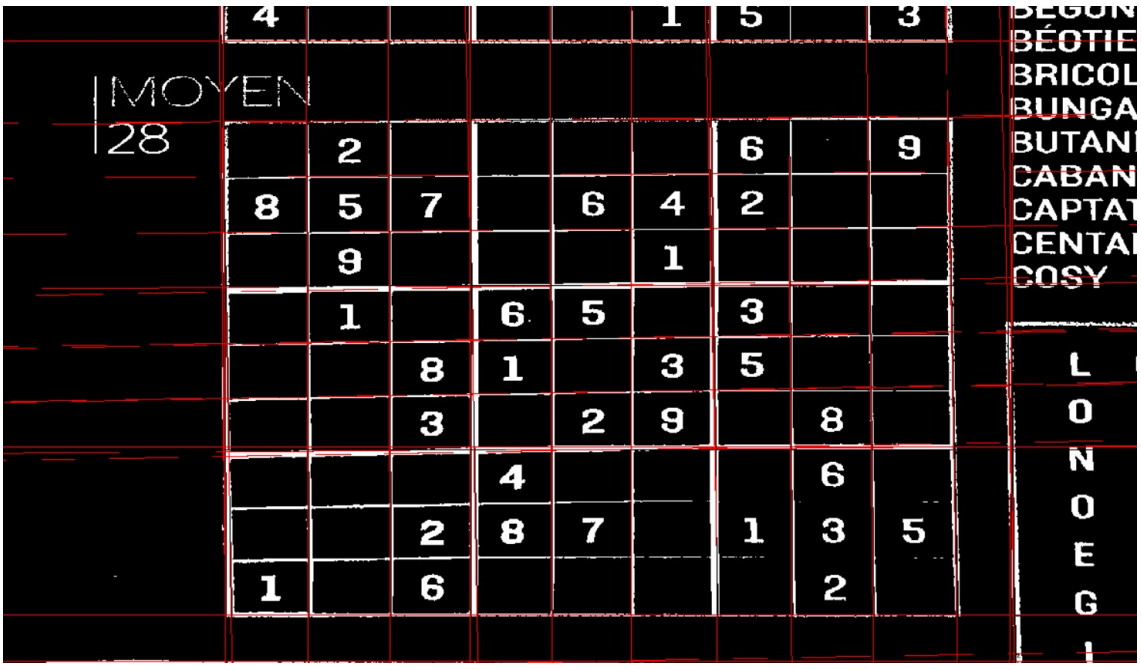
Par la suite les formules suivantes peuvent permettre de trouver les coordonnées cartésiennes (x0, y0) d'un point situé sur la droite :

$$x0 = \rho cos\theta$$
$$y0 = \rho sin\theta$$

Nous pouvons alors ajouter une distance arbitraire et le même theta pour trouver un deuxième ensemble de coordonnées situées sur la ligne :

$$x1 = x0 + d - sin\theta$$
$$y1 = x0 + d * cos\theta$$

Le problème est que cela nous renvoie trop de lignes. Nous allons donc poser une constante de seuil pour diminuer le nombre de lignes présentes. Chaque valeur de l'accumulateur est donc comparée à ce seuil pour savoir si notre valeur est suffisament grande. La valeur de seuil étant calculé par rapport à un pourcentage du maximum de notre accumulateur.
Nous pouvons donc ensuite tracer les lignes restantes.

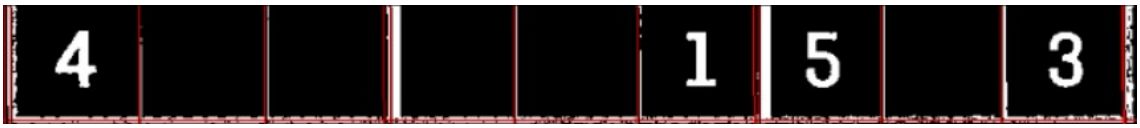


Annexe n°9 - Traçage des lignes sur une image de sudoku

Chapitre 7

Découpage de l'image

Pour cette partie nous devions nous occuper de récuperer chaque case de l'image. Sur une image où la grille étant parfaitement cadré cela était relativement simple, mais pour réussir à trouver l'emplacement exact de la grille cela est plus dur. Nous n'avons donc malheureusement pas encore réussi à trouver l'emplacement de la grille (nous devons donc nous en occuper pour la prochaine soutenance).
Cependant, l'algorithme de découpe des cases fonctionne très bien avec seulement l'ajout manuel des coordonnées de la grille.
Pour ce faire nous allons donc découper la grille ligne par ligne pour ensuite pouvoir récuperer chaque case.



Annexe n°10 - Lignes du sudoku

Annexe n°11 - Lignes du sudoku

Par la suite ces lignes vont nous permettre de récupérer chaque case pour ensuite les comparer a l'aide du XOR créé par Ethan, avec une image de chiffre de notre base de données.

Chapitre 8

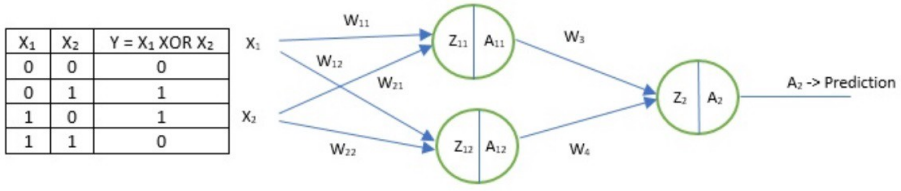
Réseau de neurones

8.1 Qu'est ce qu'un réseau neuronal

Pour commencer, il est important de rappeler ce qu'est un réseau neuronal. Les réseaux de neurones sont un sous-ensemble de "machine learning" et sont au cœur des algorithmes de "deep learning". Leur nom et leur structure sont inspirés du cerveau humain, imitant la façon dont les neurones biologiques se signalent les uns aux autres. En effet, un réseau neuronal reflète le comportement d'un cerveau humain. Le reséau de neurones est surtout utilisé pour l'intelligence artificielle, dans ce cas il s'agit d'un type particulier d'algorithme d'apprentissage automatique.

8.2 Commencement du réseau

Pour savoir par où commencer, il a fallût se renseigner en se documentant. Nous avons donc remarqué que la tache n'allait pas être si simple. Pour ne pas se tromper et bien comprendre comment un réseau de neurones marchait nous avons regardé des vidéos Youtube. Comme exemple nous pouvons citer "Dévellopeur Libre" qui a su faire une vidéo simple et compréhensible. De plus, pour mieux comprendre, des schémas explicatifs nous ont été de grand aide.



Annexe 12 - Image expliquant le principe de réseau de neurones pour un "XOR"

8.3 Avancement

Pour ce qui est de l'avancement du code, il a fallu commencer par déclarer les variables intemporels tel que le nombre d'entrées (inputs), le nombre de sorties (outputs), le nombre de "noeux cachés" (hiddenNodes) ou encore le nombre d'entrainements (TrainingSets) qui nous permettera de savoir combien de tests sont fait par boucle. Nous avons donc initialisé la variable "nbTrainingSets" à 4 car nous voulons que chaque série de tests (loop) soit faite avec 4 tests : **(0 0) (1 0) (0 1) (1 1)**
A chaque tour de boucle c'est avec ces 4 "inputs" que vont être efféctué les tests.

Nous savons que dans les réseaux de neurones, on se sert de poids. Pour le rappeler, un poids est appliqué à l'entrée de chacun des neurones pour calculer une donnée de sortie (output). Les réseaux de neurones mettent à jour ces poids de manière continue pour améliorer l'efficacité du réseau. Il nous fallait donc un moyen d'initialiser ces poids. Nous avons donc créé la fonction **"initweight()"**. Cette fonction va initialiser au hasard les poids de notre réseau de neurones pour nous renvoyer un nombre au hasard compris entre 0 et 1.
Nous avons de plus eu a créer la fonction **"sigmoid"** qui est la fonction d'activation la plus connue. Cette fonction va retourner : $\frac{1}{1+\epsilon^{-x}}$, avec x défini en paramètre.

Nous avons aussi créé la fonction **"dSigmoid"** qui est la dérivée de la fonction **"sigmoid"**. Cette dérivée permet d'interpréter la sortie du neurone comme une probabilité.

Pour être plus efficace nous avons besoin d'une fonction **"Shuffle"**. Cette même fonction permet de réduire les pertes (les erreurs). Cette fonction nous sera utile pour le deuxième partie du projet. Elle est crée mais pas utilisée entierement car lors de la suite du projet, nous aurons à sauvegarder et charger des poids du réseau de neurones. Nous avions donc commencé à créer certaines fonctions non utiles immédiatement.

Par la suite nous avons défini la fonction **"int main()"** qui nous permettera de nous donner les outputs finaux.
Pour commencer, nous avons initialisé les "hiddensWeights" et les "outputsWeights" grâce à la fonction **"initweight()"**.

Nous avons aussi initialisé les "hiddenLayerBias". La fonction d'activation des réseaux neuronaux utilise une entrée 'x' multipliée par un poids 'w'. Le biais nous permet de déplacer la fonction d'activation en ajoutant une constante (c'est-à-dire le biais donné) à l'entrée. Dans un scénario sans biais, l'entrée de la fonction d'activation est 'x' multi-

plié par le poids de connexion 'w0'. Dans un scénario avec biais, l'entrée de la fonction d'activation est 'x' multiplié par le poids de connexion 'w0' plus le biais multiplié par le poids de connexion pour le biais 'w1'. Cela a pour effet de décaler la fonction d'activation d'une quantité constante.

Nous sommes maintenant dans la partie "principale" du code. L'endroit où les résultats seront détérminés.
Nous avons donc calculé les "hiddens layer" et les "outputs layer" grâce à la fonction d'activation **"sigmoid"**
Grâce à cela nous pouvons obtenir toutes nos sorties ("output") qui évoluent suivant l'avancement de la boucle. Plus la boucle augmente plus la précision des sorties se précisent.

8.4 Fin

Pour compiler notre code, nous avons mis en place un MakeFile qui nous permet d'exécuter le fichier plus simplement. Avec la commande "make" cela nous crée notre exécutable nommé "XOR"

```
ethandage@ethandage-VirtualBox:~/SPE/mikhael.darsot/neuralnetwork$ ./XOR 1 0
0.988093
ethandage@ethandage-VirtualBox:~/SPE/mikhael.darsot/neuralnetwork$ ./XOR 0 0
0.0130028
ethandage@ethandage-VirtualBox:~/SPE/mikhael.darsot/neuralnetwork$ ./XOR 0 1
0.98873
ethandage@ethandage-VirtualBox:~/SPE/mikhael.darsot/neuralnetwork$ ./XOR 1 1
0.0115438
```

Annexe 13 - Image du résultat du XOR avec 100 000 tours de boucle

Pour exécuter notre réseau de neurone, nous avons à utiliser la commande "./XOR" avec juste après le test que nous voulons tester. Par exemple, comme montré ci-dessus, nous pouvons tester seulement avec des valeurs correctes **(0 0, 0 1, 1 0, 1 1)**. Si les valeurs misent en entrée ne sont pas comprises dans les valeurs indiquées ou que aucune valeurs n'est mise en entrée alors un message d'erreur apparait.

```
ethandage@ethandage-VirtualBox:~/SPE/mikhael.darsot/neuralnetwork$ ./XOR 1 2
XOR: error
ethandage@ethandage-VirtualBox:~/SPE/mikhael.darsot/neuralnetwork$ ./XOR
XOR: error
```

Chapitre 9

Algorithme de résolution du Sudoku

9.1 En quoi consiste ce solver ?

Simplement, le solver prend en paramètre un fichier texte sous le format suivant avec un sudoku non résolu avec les points qui représentent les cases à remplir pour résoudre le sudoku.

```
trookiz@trookiz-VirtualBox:~/OCR/mikhael.darsot/solver$ cat grid_00
... ..4 58.
... 721 ..3
4.3 ... ...
21. .67 ..4
.7. ... 2..
63. .49 ..1
3.6 ... ...
... 158 ..6
... ..6 95.
```

Annexe 14 - Fichier d'origine à résoudre

Finalement, le solver crée un nouveau fichier texte sous le même format et le même nom avec un .result en plus, mais cette fois-ci avec les valeurs du sudoku. Une fois celui-ci résolu, il n'y a donc plus de '.' qui représentent les cases à compléter pour résoudre le sudoku du fichier d'origine.

```
trookiz@trookiz-VirtualBox:~/OCR/mikhael.darsot/solver$ cat grid_00.
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

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

Annexe 15 - Fichier attendu après l'éxécution du solver

9.2 Implémentation du solver

Tout d'abord, il faut transformer le fichier texte en un tableau de chiffre de taille 9X9 en y mettant des 0 pour les cases du tableau à compléter. Ce tableau permettra ensuite de résoudre le sudoku et sera passé en paramètre d'une fonction qui s'occupera de cette partie. Une fonction a été crée spécialement pour cette implémentation de texte vers un tableau 9x9 d'entiers qui sera passé directement en paramètre de la fonction résolvant le sudoku et modifiant ce tableau à 2 dimensions pour enlever les 0 et mettre les valeurs de résolution.
Ce solver nécessite un algorithme de résolution basé sur le backtracking qui détecte dans quel carré de 3x3 se situe la case qu'on essaie de remplir et ensuite vérifiant chaque ligne, colonne et ce carré de 3x3 où se situe le 0 détecté. Cet algorithme sera efféctué pour chaque case où il y'a un 0 pour savoir quel chiffre peut être mis dans la case à compléter. Cet algorithme peut également détecter si la grille passé en paramètre est valide. Dans le cas contraire elle renverra une valeur indiquant la non validité de ce sudoku et qui permettra par la suite de ne pas créer de nouveau fichier ainsi que d'afficher une erreur.

Cet algorithme fonctionne uniquement si le sudoku passé en paramètre est résolvable. Dans le cas contraire un message d'erreur sera renvoyé et aucun fichier supplémentaire ne sera créé. Cela a été ajouté pour ne pas créer de problème si la grille n'est pas valide et ne pas créer de fichier supplémentaire vide ou avec des valeurs érronées.

```
trookiz@trookiz-VirtualBox:~/OCR/mikhael.darsot/solver$ ./solver grid_00
solver: The sudoku has no solution
```

Annexe 16 - Exemple d'une grille non valide

A la fin de ce processus, si la grille est correcte alors le tout est implémenté dans un nouveau fichier créé par le fichier solver qui suivra le même format que le précedent. Par la suite, une fonction s'occupe d'écrire directement les valeurs dans le tableau 9X9 une fois le sudoku résolu par la fonction utilisant le backtracking pour obtenir le fichier final contenant le sudoku résolu.

9.3 Création du solver et fin

Pour finir, un MakeFile a été créé permettant de générer le fichier solver grâce à la commande "make" qui permet ensuite de résoudre n'importe quel sudoku valide implémenté sous le format présenté précédemment.

Il suffit simplement de le passer en paramètre de ce fichier "solver" pour qu'un nouveau fichier soit alors crée sous le même nom mais avec un .result ajouté à la fin de celui-ci. Ce fichier contient donc finalement le sudoku résolu présenté sous le même format.

```
trookiz@trookiz-VirtualBox:~/OCR/solver$ ls
grid_00  solver
trookiz@trookiz-VirtualBox:~/OCR/solver$ ./solver grid_00
trookiz@trookiz-VirtualBox:~/OCR/solver$ cat grid_00.result
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

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

Annexe 17 - Exemple de résolution d'une grille

Si le nombre de paramètres passé n'est pas celui attendu, alors une erreur se produit et nous avons un message d'erreur qui l'indique directement dans le terminal.

```
trookiz@trookiz-VirtualBox:~/OCR/mikhael.darsot/solver$ ./solver
solver: Only one parameter is required
trookiz@trookiz-VirtualBox:~/OCR/mikhael.darsot/solver$ ./solver grid_00 grid_00
solver: Only one parameter is required
```

Annexe 18 - Message d'erreur lors du mauvais nombre de paramètre

Chapitre 10

Conclusion

Enfin, nous pouvons constater que notre projet OCR a connu une avancée importante durant ces deux premiers mois de travail. En effet, la quasi-totalité des objectifs du cahier des charges ont été atteints et nous sommes donc dans les temps vis à vis de l'avancement "idéal" du projet. Durant la deuxième partie du semestre, notre objectif sera de garder la même allure et de continuer d'avancer sur le projet afin de pouvoir présenter un produit fini lors de la soutenance finale, c'est à dire un logiciel complet permettant de renvoyer une grille de sudoku résolue à partir d'une image de départ.