Rapport De Soutenance
OCR

Clément FABIEN - Grégoire GENET
Gabriel CALVENTE - Cédric DAMAIS

Prépa S3
Novembre 2022 - Décembre 2022

Table des matières

1 Introduction **4**

1.1 Présentation du Projet 5

1.2 Avancement du projet 6

1.3 Répartition 7

2 Rotation de l'image **8**

2.1 Rotation Manuelle 8

2.1.1 Exatraction des Matrices Unicolonnes 8

2.1.2 Matrice De rotation 9

2.2 Hough Transform 10

2.2.1 Hough : Detection de Ligne 10

2.2.2 Algorithme Appliquer 11

2.2.3 Meilleur Parametre 12

2.3 Rotation Automatique 13

2.4 InterFace Graphique 13

3 Binarisation **14**

3.0.1 Augmentation Des contrastes 14

3.0.2 Algorithme De Otsu 14

3.0.3 Difficultés recontrées 15

4 Résoudre la grille **16**

4.0.1 Resolution en BackTracking 16

5 Rendre une image finale **17**

6 Filtres **19**
6.1 Grayscale 19
6.2 Erosion 21
6.3 Dilation 21
6.4 Filtre De Gauss 22

7 Détection Des Bords **24**
7.1 Algorithme De Sobel 24
7.2 Algorithme De Canny 25

8 Détection **27**
8.1 Hough Lines 27
8.2 Intersections 28

9 Découpage **29**

10 Réseau de neurones **30**
10.1 Documentation 31
10.2 La reprise du modèle initial 31
10.3 Les challenges 31
10.4 Implémentation 32
10.5 Les difficultés rencontrées 32
10.6 Sauvegarde des données après entraînement 33
10.7 outils développé pour la réalisation du réseau 34
10.8 fonctionnalités 34

11 Interface Utilisateur **35**
11.1 Glade 35
11.2 Boutons 35
11.3 Fonctionnement 36
11.4 Difficultés rencontrées 37

12 Ressentis **38**
12.1 Grégoire GENET 38

12.2 Cédric DAMAIS 39
12.3 Gabriel CALVENTE 40
12.4 Clément FABIEN 41

13 Mise en commun des programme/Fusion du travail 42

14 Conclusion 43

1 Introduction

En cette seconde année de classe préparatoire à l'EPITA, nous sommes de nouveau amené à réaliser un projet en groupe, afin de nous apprendre les bases du travail en groupe, de la recherche individuelle et surtout de nous apprendre à ne pas baisser les bras malgré les difficultés. Ce projet consiste à délivrer sous une date donnée, ce qui est appelé un OCR (Optical Character Recognition), permettant de résoudre un sudoku à partir d'une photo de ce dernier. Notre groupe de projet est constitué de quatre personne : Calvente Gabriel, Damais Cédric, Genet Grégoire et Fabien Clément, ce dernier étant le chef de projet. Ainsi, lors de ce rapport, nous vous expliquerons en quoi consiste concrètement le projet au global, son état d'avencement, les aspects techniques liés au projet, les difficultés rencontrées, et les attentes quant à notre rendu final.

1.1 Présentation du Projet

Le projet de ce troisième semestre à l'EPITA est de faire un OCR (Optical Character Recognition) permettant de résoudre un sudoku à partir d'une photo de ce dernier. Pour ce faire, nous devons effectuer un programme compilable dans le langage C, qui prend en paramètre une photo de sudoku, et qui renvoie la même photo avec la grille de sudoku résolue. Ainsi, notre programme terminé contiendra les éléments suivants : Une partie preprocessing, avec la binarisation de l'image, la rotation, et tout ce qui permettrait d'utiliser optimalement l'image donnée. Ensuite, une partie sur la détection des bords de la grille, puis des cases, mais aussi la sauvegarde des 81 cases séparées dans un dossier auxiliaire. Puis un programme de réseau de neurones, qui permettra de reconnaitre séparemment les numéros dans les cases différentes. Après avoir reconnu les numéros, ainsi que la disposition théorique de la grille de sudoku, il y a ensuite un programme qui résout le sudoku et renvoie la grille résolue. Puis, un dernier programme mettra tout les numéros sur les images des cases vides, puis recollera les morceaux des 81 cases différentes, en reverra enfin, la grille résolue. Ce programme compilé pourra, s'il reste du temps, être utilisé comme une petite application, afin d'être facile d'utilisation et compréhensible.

1.2 Avancement du projet

Le projet avance à grand pas depuis un mois, mais quelques difficultés se font ressentir, notamment sur la détection de la grille et des cases, et c'est pour nous la partie la plus difficile. Nous sommes donc majoritairement avancées sur les requis pour la première soutenance, cependant nous ne disposons pas de la détection des cases de la grille ainsi que de la grille elle même. Nous avons cependant la détection des bords de la grille, ainsi que tout les autres requis de la soutenance, c'est à dire la binarisation, la rotation manuelle, le chargement de l'image dans le programme, une aide visuelle pour voir les images modifiées, un programme de résolution de sudoku sous le format demandé, une manipulation de fichiers basique, ainsi qu'un début de base de données de sudoku afin de tester nos différents programme, puis pour finir un réseau de neurone fonctionnel sachant calculer un XOR, ainsi qu'une manipulation de fichier afin de sauvegarder différents poids et liaisons entre les différents neurones de ce réseau. La majorité des éléments demandés pour cette soutenance sont effectués, mais à cause de la difficulté rencontrées avec la détection des cases et de la grille, nous n'avons pas pu faire. Ainsi, notre prochain objectif est de terminer cela rapidement.

1.3 Répartition

Pour ce qui est de ce projet, nous nous sommes donnés chacun des tâches différentes, afin de maximiser le temps de travail. Ainsi, Gabriel s'est occupé de faire le réseau de neurone, Cédric et Grégoire se sont occupés du pre-processing de l'image, c'est à dire le chargement de l'image, la rotation, la binarisation. Cédric à la rotation manuelle, puis automatique, et Grégoire au chargement de l'image ainsi que la binarisation. Grégoire s'est aussi occupé de la résolution de la grille de sudoku. Finalement, Clément s'est occupé de la détection des bords de la grille, et du découpage des cases.

2 Rotation de l'image

Pour utiliser l'image donnée par l'utilisateur à notre programme, il faut pouvoir la reconnaitre en toutes circonstances. Ainsi, il faut retourner l'image, pixels par pixels pour faire en sorte qu'elle soit droite et utilisable par le programme (notamment le réseau de neurones).

2.1 Rotation Manuelle

2.1.1 Exatraction des Matrices Unicolonnes

La rotation manuelle, c'est à dire faire tourner l'image d'un certain degré. En théorie, c'est une transformation très facile à comprendre et elle sera très utile plus tard pour la rotation automatique. Toute la diffculté était de réussir à bouger chaque pixel de notre image à leur nouvelle bonne position en fonction de l'angle donné. Pour parvenir à faire cette transformation de l'image nous avons considéré les coordonnées (x,y) du pixel dans le tableau comme une matrice unicolonne.

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

2.1.2 Matrice De rotation

Nous avons realisé cette transformation des coordonées pour pouvoir ensuite faciliter le changement de position en fonction d'un angle à l'aide d'une matrice de rotation.
En deux dimensions, les matrices de rotation ont la forme suivante :

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Avec cette matrice de rotation, trouver la nouvelle position (x',y') des pixels n'est qu'une simple multiplication de matrice :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Une fois les coordones (x',y') du pixel trouve il ne suffie que de le placer à sa nouvelle position. Puis repeter se processus sur tout les pixels de notre images.

2.2 Hough Transform

La transformation de hough etait necessaire pour arriver à faire la detection de ligne et trouver l'angle necessaire pour avoir une image a la bonne position.

2.2.1 Hough : Detection de Ligne

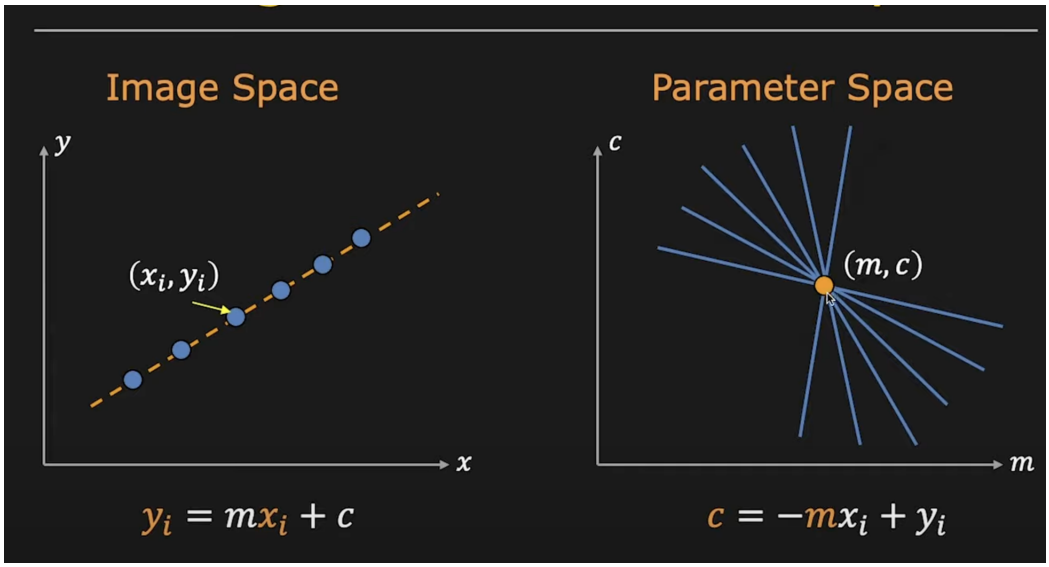
Apres la Binarization de l'image nous nous retrouvons avec des milliers de pixels blanc ou noir. Dans ces pixels, se trouve les lignes que l'on cherche a detecter, et la a ete la plus grosse difficulte rencontrer retouver les pixels qui forment les lignes qui constituent les lignes. On se retrouve donc à chercher un droite qui pour equation :

$$y = mx + c$$

dans un tableaux de pixels a des positions (x_i, y_i).
Grace à la Transformation de hough on arrive a resoudre se probleme en regardant le probleme dans deux espace different. En representant la droite sur deux plans differents car on a :

$$y_i = mx_i + c \Leftrightarrow c = -mx_i + y_i$$

On a l'espace de l'image dans lequelle on cherche nos lignes, grace a notre equivalence dans les deux plans, un point sur une des lignes dans le plan de l'image est une droite sur notre deuxieme plan. Chaque point sur une meme lignes du plan image vont cree des droites sur le deuxieme plan qui vont etre secante.



2.2.2 Algorithme Appliquer

Apres avoir compris le concept de hough l'agoritme appliquer etait assez simple à implementer :

Etape 1 : Cree un Accumulateur (Tableau initialiser avec des 0)

Am(m,c)

La difficulter rencontrer lors de la manipulation de l'accumulteur est que si le teableau est trop grand des lignes se fusionnent, si il est trop petit, le bruit sur l'image empeche la detection de certaines lignes sur l'image. On donc pris comme taille du tableau la diagonale de l'image binariser.

Etape 2 : Pour tout points (x_i, y_i) sur les lignes du plan image Si (m,c) est sur la ligne $c = -mx_i + y_i$

$$A(m, c) = A(m, c) + 1$$

Etape 3 : Parcourire l'accumulateur pour trouver les maximum qui represente les lignes sur le plan image.

2.2.3 Meilleur Parametre

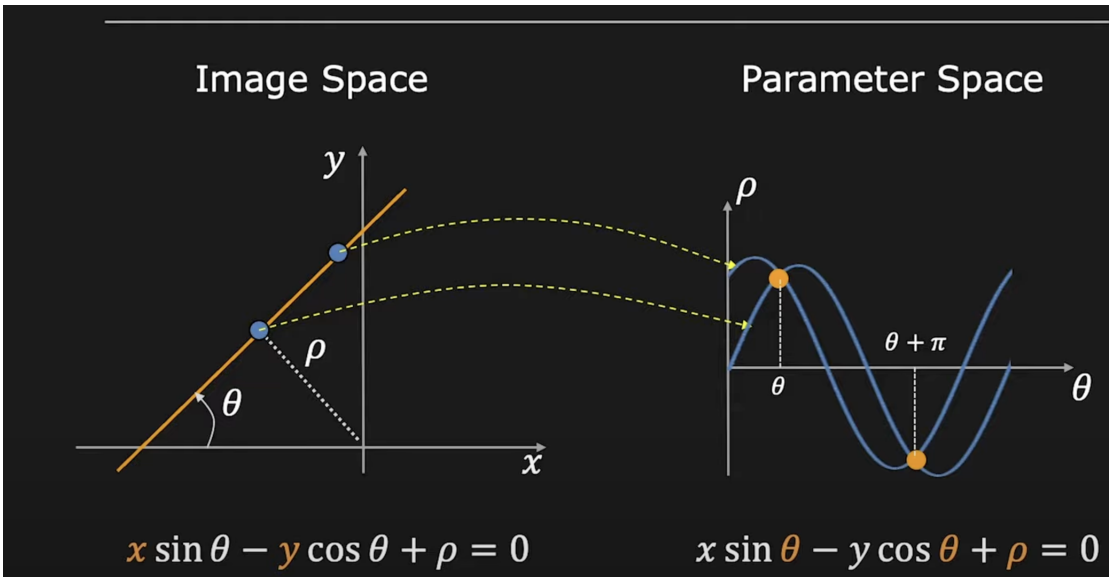
Durant l'implementation de la transformation de hough, on a rencontrer un probleme qui etait le fait que le coefficient directeur :

$$-\infty \leq m \leq \infty$$

Cette contraite nous obligeaient à avoir un accumulateur tres grand, et comme nous l'avons precisé cela ne permet pas une bonne detection de lignes. De plus un grand accumulateur necessite l'utilisation de beaucoup plus de memoire . Comme solution a se probleme, On utilise les coordones polaires (θ, ρ) avec θ l'angle crée par la droite et ρ la distance de la droite à l'origine du plan. En utilisants des equations de passages pour convertir des coordonnées cartésiennes en coordonnées polaires on se retrouve avec :

$$x\sin(\theta) - y\cos(\theta) + \rho = 0 \Leftrightarrow x\sin(\theta) - y\cos(\theta) + \rho = 0$$

Avec ces nouveau parametre l'agorithme reste le meme avec un accumulateur $Am(\theta, \rho)$:



Une fois que l'accumulateur est remplie on va le parcourire pour trouver l'angle necessaire à la rotation.

2.3 Rotation Automatique

La rotation automatique n'a pas ete tres difficile à implemeter car, apres avoir realiser la Transformation de Hough qui nous permis de trouver l'angle necessaire pour mettre l'image à la bonne position, il n'a suffit que d'extraire cette valeur puis d'appliquer la rotation manuel avec l'angle trouver apres la transformation de hough.

2.4 InterFace Graphique

Pour l'interface graphique nous avons utilisé la librairie 'GTK' qui nous facilite la création de l'interface graphique. Cette interface est interessante à utiliser car elle nous facilite l'utilisation des boutons, afficher des images ou encore detecter les signaux emis.

3 Binarisation

Afin d'utiliser l'image pour le programme, il faut binariser l'image ; c'est à dire faire en sorte de convertir l'image avec seulement deux couleurs : le blanc et le noir. Pour se faire, il faut utiliser un seuil, à partir duquel chaque pixel ayant une couleur supérieure à ce seuil sera considérée comme blanche, et à l'inverse, comme noire. Ainsi, il existe plusieurs méthodes pour trouver ce seuil adaptés aux différentes images, tel que l'algorithme d'Otsu, de Sauvola, ou bien de Gauss.
Pour ce projet, nous avons décidé d'utiliser la méthode d'Otsu, qui utilise la variance des pixels dans l'image pour trouver le seuil adaptatif à l'image. C'est une méthode qui est abordable et rapide à comprendre, c'est pour cela que nous l'avons choisi.

3.0.1 Augmentation Des contrastes

Avant de passer notre image dans l'algorithme d'Otsu, il faut tout d'abord augmenter les contrastes de l'image pour faciliter la recherche d'un seuil. Pour cela, il faut faire l'histogramme des pixels de l'image, puis égaliser cet histogramme. Cela permet de répartir les éléments de l'histogramme en son sein. Puis, changer l'image grâce à l'histogramme égalisé.

3.0.2 Algorithme De Otsu

Apes avoir obtenue une image égalisée on applique sur celle ci l'algorithme d'Otsu, pour trouver un seuil à adapté à notre image. C'est un algorithme permettant de calculer la variance des pixels pour chaque couleurs, afin de chercher la variance optimale pour laquelle les couleurs seront le moins im-

pactés. Cette variance va être notre seuil avec lequel nous pourrons ensuite déduire chaque pixel inférieur comme étant noirs, et chaque pixel supérieurs comme étant blancs. Ensuite, binariser l'image en utilisant le seuil trouvé, et enfin, sauvegarder notre image binarisée pour la garder afin de pouvoir l'utiliser dans les programmes suivants.

3.0.3 Difficultés recontrées

Pour ce qui est de la binarisation, ce ne fut pas simple de trouver des informations sur un algorithme permettant un résultat constant. Il y avait des informations sur l'algorithme de Sauvola, celui d'Otsu, celui de Minimum Error Threshold, et j'en passe, tous ayant un point fort, et un point faible sur beaucoup de points. Cela renda les recherches assez fastidieuses, rendant beaucoup de nos recherches inutilisables ou bien obsolètes. De plus, les explications de ces algorithmes n'était quelques fois pas assez claires pour que l'on puisse les utiliser au maximum ou bien même que l'on réussisse à les implémenter. Cependant nous avons tout de même réussi à trouver un algorithme sûr et fiable afin de faire notre binarisation et nous rendre une image pas trop dénaturée par l'image initiale, malgré le noir et blanc.

4 Résoudre la grille

Après avoir analysé la grille grâce au réseeau de neurone, nous pouvons utiliser toutes les 81 cases afin de créer un document texte avec une représentation de la grille en photo, avec des '.' pour les cases à remplir, et les chiffres autrement.

4.0.1 Resolution en BackTracking

Puis, avec un algortihme de résolution, grâce à un système de backtrack (récursion successive sur un chemin donné), la grille est résolue et est renvoyée dans un nouveau document texte sans aucun '.'. Ainsi, la résolution de la grille n'est pas le challenge de notre projet, étant donné que c'est assez simple, tant pour le programme que pour la création de fichiers.

5 Rendre une image finale

Pour ce qui est du rendu de l'image finale, nous avons rencontré quelques problèmes. Tout d'abord, la création de toute pièce d'une grille de sudoku, qui s'est avéré plus compliqué que prévu, étant donné que nous ne connaissions pas grand chose à la librairie SDL2. Ensuite, ce fut d'inséré le résultat de la grille.result dans la grille.png, étant donné que rien ne fonctionnait au départ. Ainsi, nous avons essayé d'utiliser la sous librairie de SDL2, TTF. Cependant, il y eu un soucis aussi avec cette librairie, l'importation de différentes polices d'écriture ne se faisant pas et rendant le programme donc utilisable. Après maintes et maintes recherches sur le sujet, nous avons trouvé une nouvelle façon de faire, étant de copier une surface sur une autre. En effet, il est possible grâce à la libraire SDL2 de copier une surface sur une autre, afin de les fusionner. Ainsi, nous avons créé des images de chaque chiffre de 1 à 9 dans un dossier "digits/", puis avec notre programme, on associait l'endroit où l'on copiait l'image sur la grille à son emplacement dans la grille résultante de la résolution. Ainsi, nous obtenons une grille nette et précise résultant à notre grille de départ, permettant de retourner notre tout dernier résultat sur ce projet, la grille résolue.

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

6 Filtres

6.1 Grayscale

Avant de pouvoir traiter l'image du sudoku en tant que tel, il faut supprimer toutes couleurs afin de pouvoir mieux détecter la grille, les cases et les chiffres. Ainsi nous pouvon appliquer un filtre de gris qui nous permet de transformer l'image entièrement en noir et blanc.

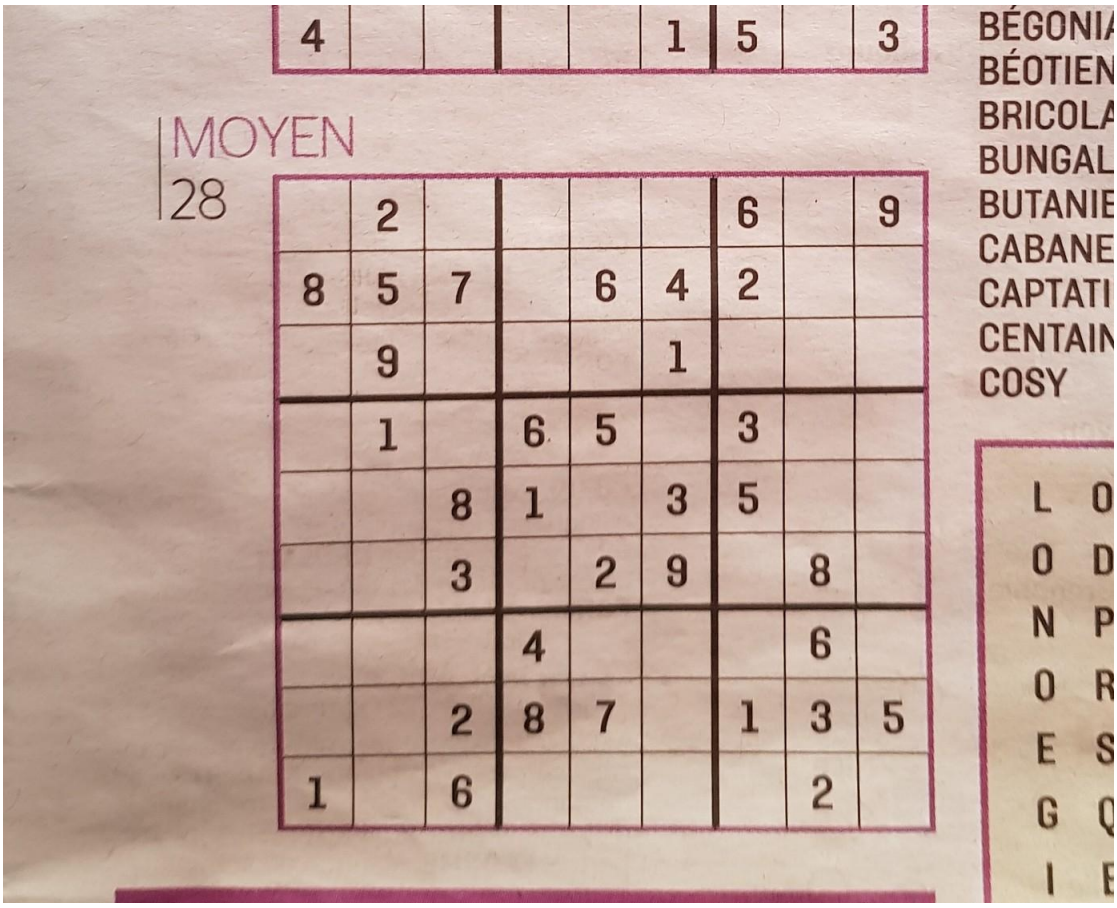


Image non traitée

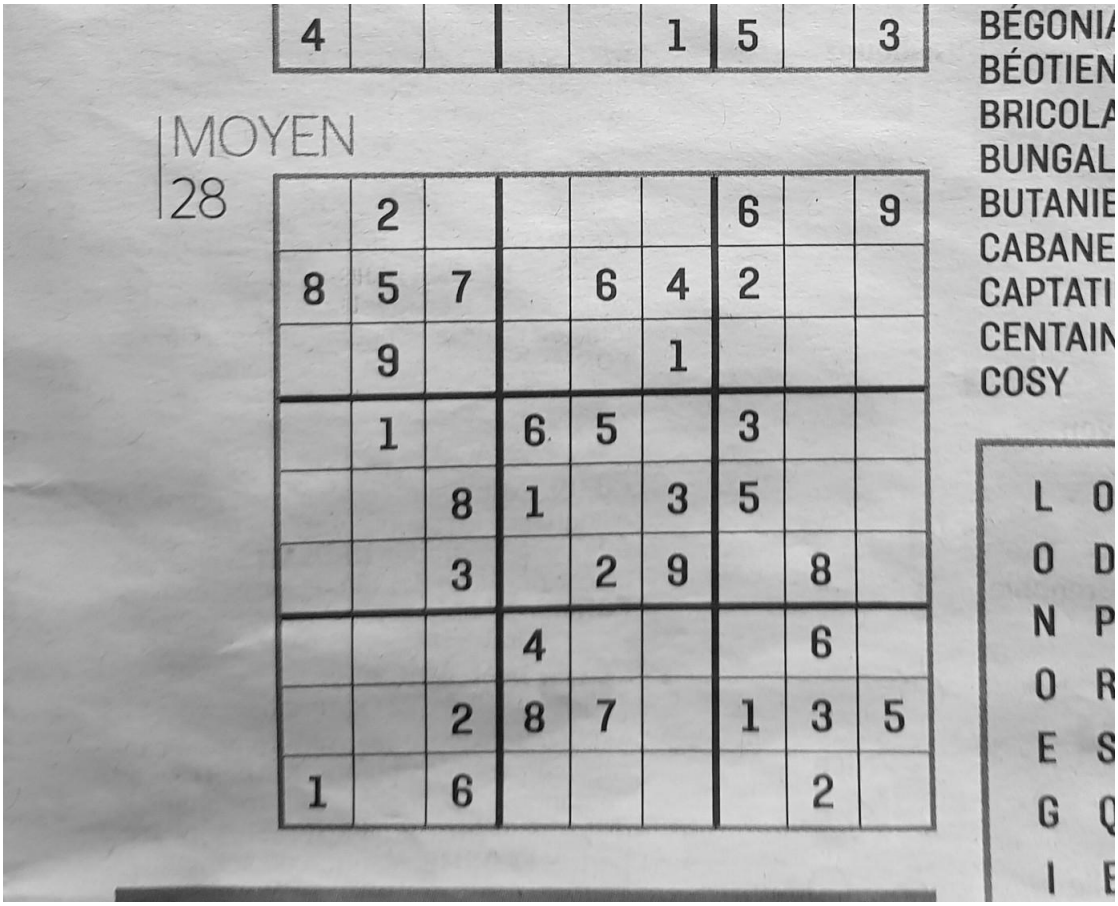


Image après un filtre de gris

Pour avoir ce résultat, on aplique sur chaque pixel une moyenne des couleurs (RGB) et on le réécrit avec cette nouvelle valeurs.

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

6.2 Erosion

Après avoir effectué le flou, la binarisation et la rotation automatique de l'image, nous avons implétmenté l'algorithme de l'érosion de l'image, permettant à toutes les textures noires de l'image de réduire de taille, réduisant ainsi le bruit possiblement présent sur l'image. Cela combiné à l'algorithme suivant permet une réduction du bruit de l'image sans pour autant rendre illisible les textures de l'image.

6.3 Dilation

L'algorithme de dilation n'est rien de plus que l'inverse de l'algorithme précédent. Il permet d'agrandir toutes les textures de l'image. Ainsi, en combinant ces deux algoritmes, permettant de filtrer le bruit de l'image, nous atténuons, voire supprimons, le bruit de l'image avec le premier algorithme, puis nous remettons les textures à leur taille initiales, permettant de supprimer tout gros bruit de l'image.

6.4 Filtre De Gauss

Après avoir appliqué notre filtre de gris, on doit appliqué un filtre de gauss. Ce filtre permet de réduire le bruit de l'image en la flouttant un tout petit peu. Ce flou permet de rendre la binarisation plus tard meilleur en enlevant certaines tâches visible autrement.

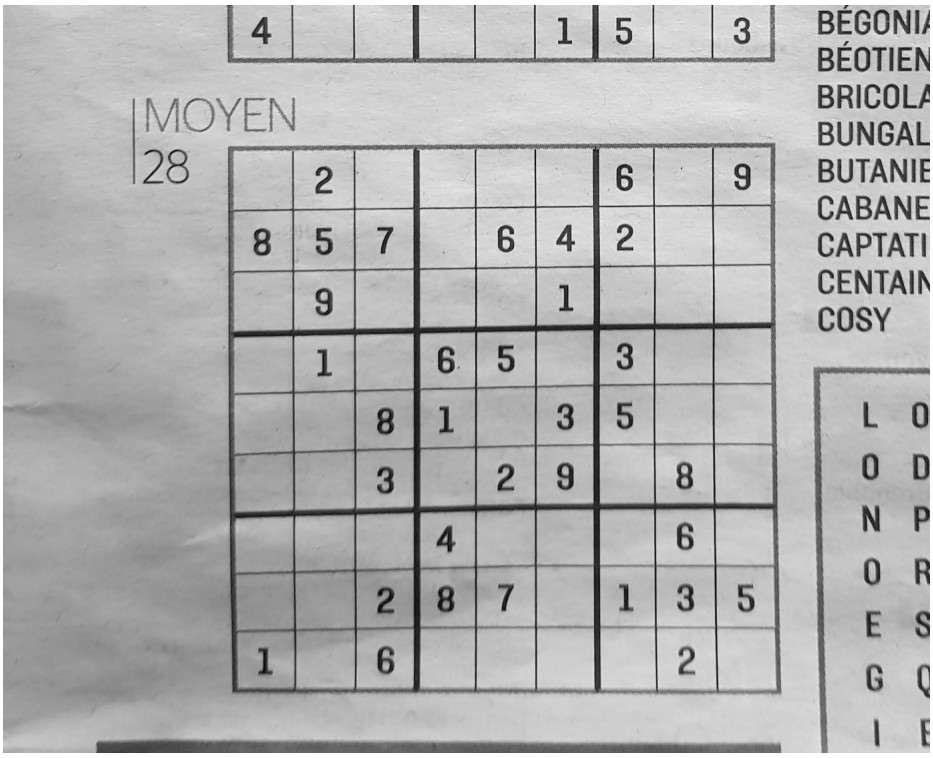


Image après un filtre de gris

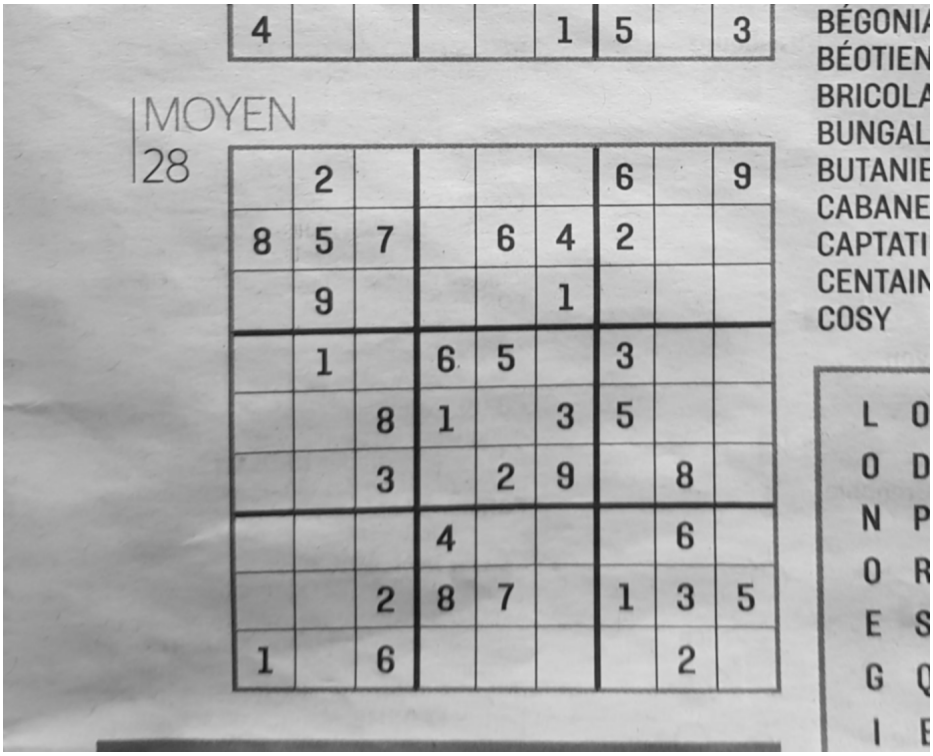


Image après un filtre de gris et filtre de Gauss

La méthode de Gauss consiste à regarder chaque pixel et à faire une moyenne de ses pixels voisins (3x3, 5x5, 7x7, etc...) et appliquer un poids à chacun de ces pixels. Cette matrice qui stocke les poids des pixels s'appelle un kernel. Nous utiliserons un kernel de 5x5 pour notre projet avec les valeurs ci-dessous.

$$kernel = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

7 Détection Des Bords

7.1 Algorithme De Sobel

Similairement au filtre de Gauss, l'algorithme de Sobel va utiliser des kernels. Cette fois deux de 3x3. Un en x :

$$xKernel = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

et un en y :

$$yKernel = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Une fois appliqué, ces kernels nous permettent d'avoir des bords très nets.

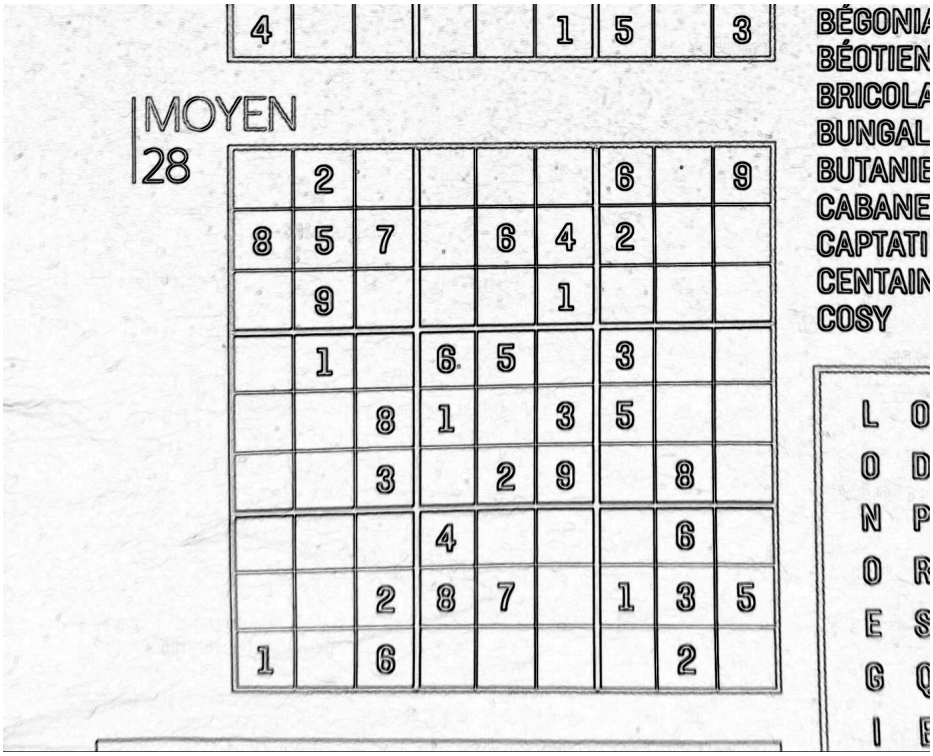


Image après un filtre de gris, un filtre de Gauss et l'algorithme de Sobel

Cet algorithme est également très pratique car il nous permet de savoir dans quel sens est le bord. Ainsi nous pouvons appliquer l'algorithme de Canny qui nous permet de garder que les bords les plus forts en fonction de leur angle.

7.2 Algorithme De Canny

Après l'algorithme de Sobel, nous avons pu récupérer l'orientation de chaque ligne dans le sudoku. Ainsi l'algorithme de Canny n'as plus qu'a vérifier si un bord est assez fort pour être gardé. Pour ce faire, l'algorithme de Canny peut se décomposer en plusieurs étapes. La première est d'affiner tous les bords pour qu'ils ne fassent plus qu'un pixel de largeur. Ensuite, nous devons regarder pour chaque pixel appartenant à un bord si c'est un maximum local, c'est-à-dire si dans l'orientation, donné par l'algorithme de Sobel, ses voisins sont plus intense ou pas. Si ils le sont alors on enlève ce pixel du bord et on garde les autres. Pour finir, nous pouvons faire une petite correction pour être sur que les bords restent cohérents avec un double threshold. Ainsi, les bords dans l'images apparaissent très nettement et après

binarisation la grille ressort correctement.

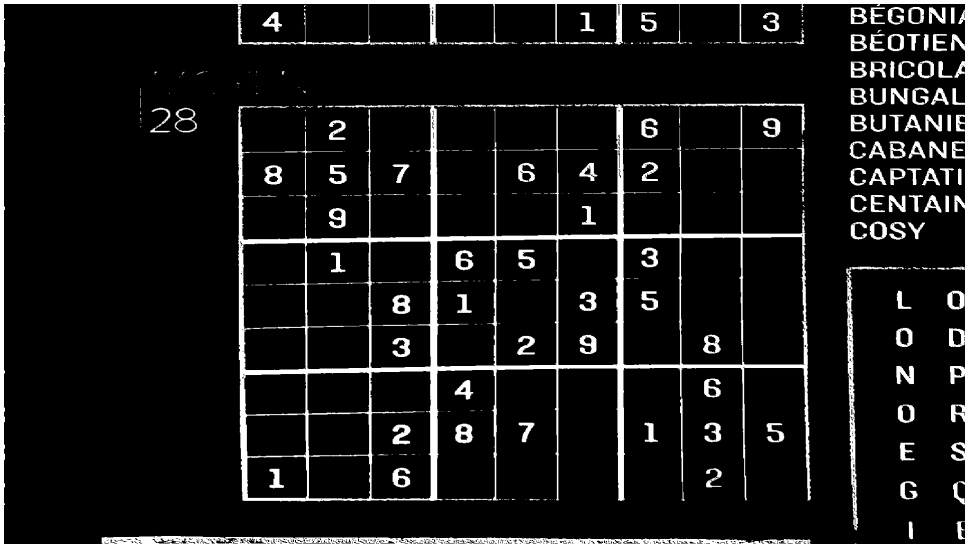


Image après un filtre de gris, un filtre de Gauss, l'algorithme de Sobel, l'algorithme de Canny et la binarisation

8 Détection

Après avoir récuperé les contours de l'image, nous pouvons passer à la détection des lignes dans l'image pour extraire la grille du sudoku.

8.1 Hough Lines

Pour détecter les lignes de notre image, on utilise la transformation de hough. Grâce à l'accumulateur récupéré au préalable lors de la rotation automatique, on peut analyser les coordonnées que l'on a en polaire pour les repasser en cartésiens ce qui nous permet d'afficher chaque ligne de l'image.

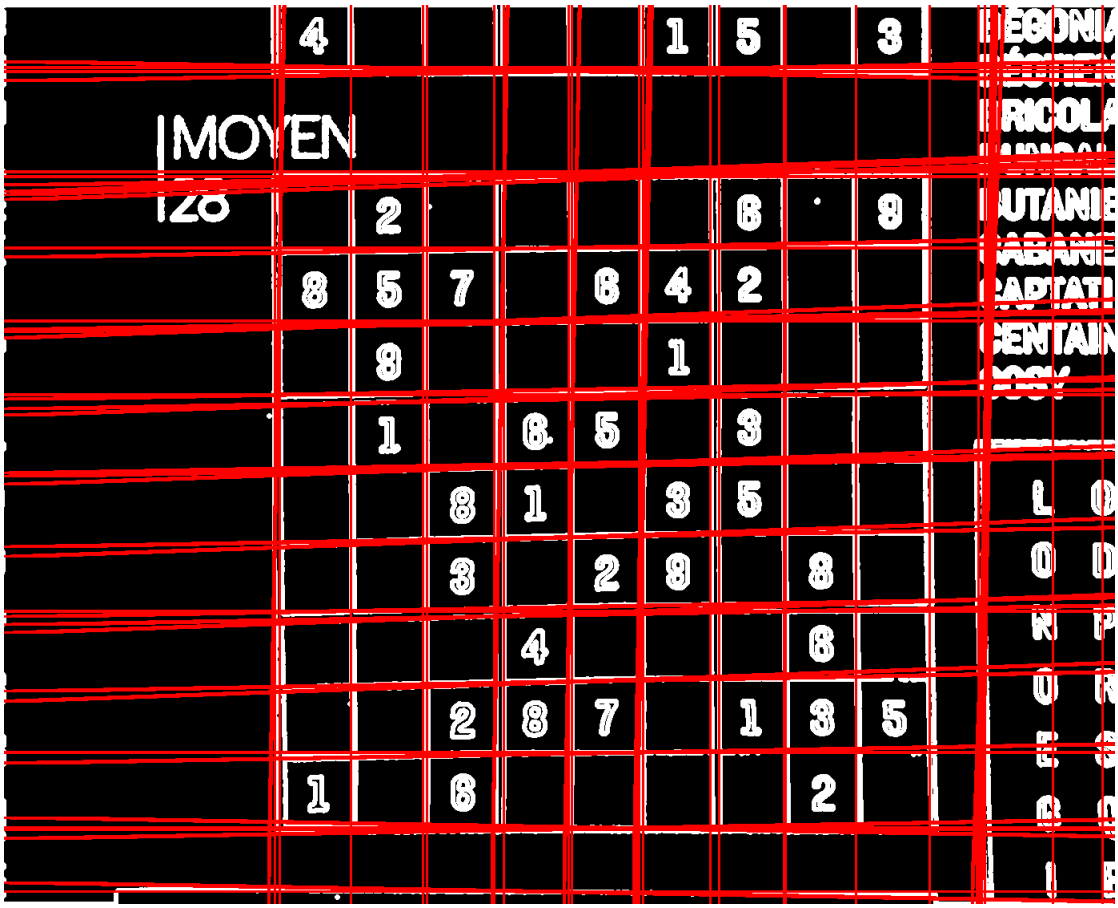


Image avec les lignes détectées

8.2 Intersections

Après cette détection des lignes, nous devons détecter toutes les intersections entre les lignes. Une fois ces intersections détectées, nous pouvons prendre les 4 intersections les plus éloignées de l'image qui seront nos coins du sudoku. Une fois les coins détectés, nous pouvons adapté l'image pour qu'elle prennent l'entièreté de l'image originelle pour qu'on est plus que la grille a traité.

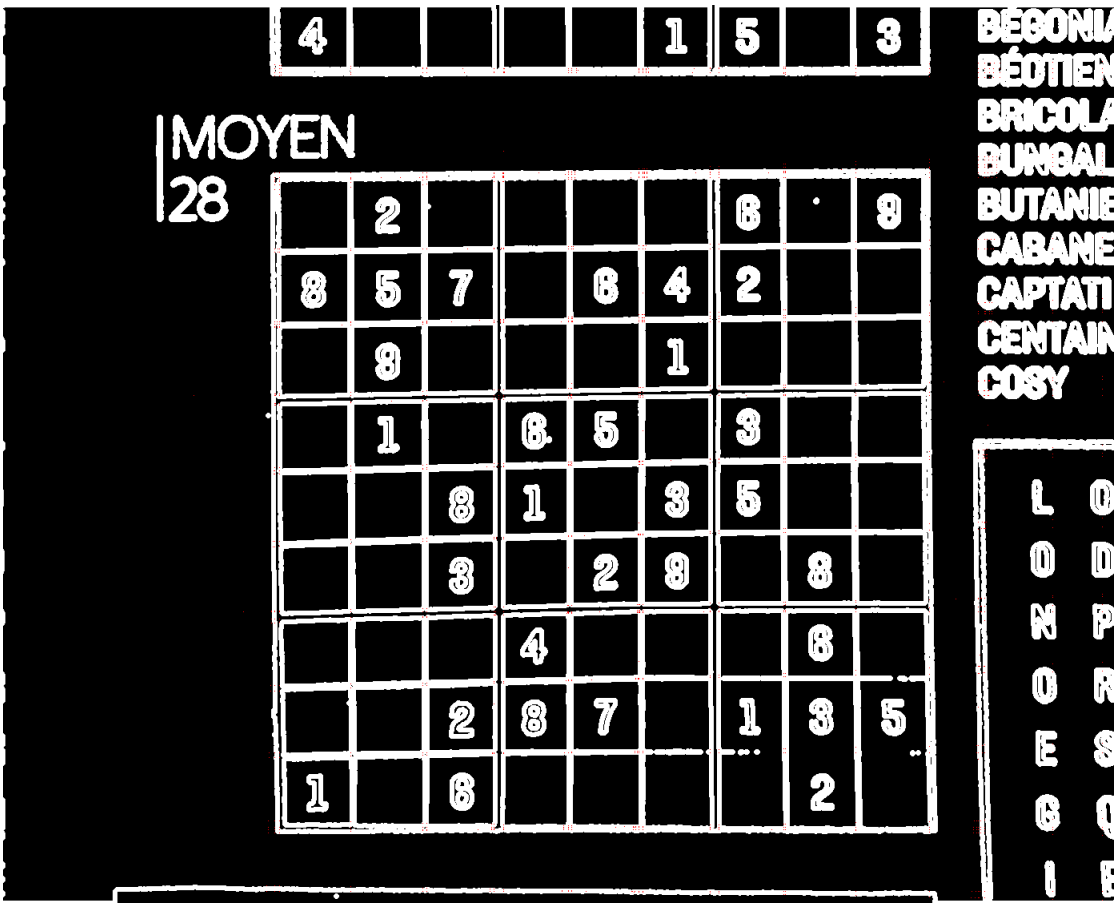


Image avec les intersections des lignes détectées

9 Découpage

Une fois la grille remise bien en place, nous pouvons découper les 81 cases du sudoku en découpant l'image en 9x9 et en sauvegardant chaque image dans un sous-dossier cases pour être classées dans l'ordre pour être traité par le réseau de neurones. Pour découper correctement l'image, on prend la plus grande distance entre les deux coins du haut et les deux coins du bas. On peut alors parcourir l'image en partant des coins et à partir de la distance récupérée pour découper l'image en 81 cases.

10 Réseau de neurones

Dans ce projet, le réseau de neurones est une composante capitale dans le bon fonctionnement du programme dans sa globalité puisque c'est lui qui va permettre de reconnaitre les numéros dans chaque case. Sans cette étape de reconnaissance de caractère, il est impossible de compléter la grille de sudoku et le produit devient inutilisable. La réalisation d'un tel réseau de neurones s'est révélé difficile mais très intéressante et enrichissante.

10.1 Documentation

Dans un premier temps, nous avons décidé de se documenter et de comprendre la théorie derrière un réseau de neurone. Pour ce faire, de nombreuses sources de qualité sont à disposition sur internet comme la série de vidéo sur les réseaux de neurones faite par **3Blue1Brown** ou bien le livre de Mnielsen. Pour cette première soutenance, nous nous sommes donc concentré sur comprendre la théorie des réseaux de neurones et la mettre en pratique sur un exemple plus rudimentaire que la reconnaissance d'images. Nous avons donc réaliser un réseau de neurone qui apprend la fonction OU EXCLUSIF.

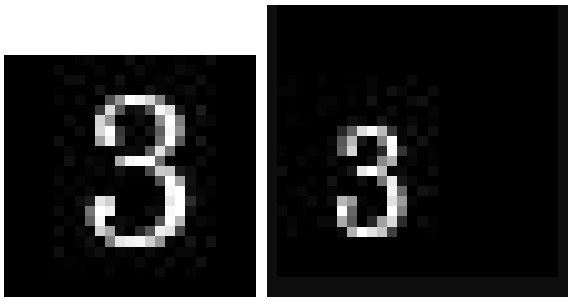
10.2 La reprise du modèle initial

Pour cette deuxième étape du projet, il a fallu implémenter toutes les fonctionnalités permettant de reconnaitre des caractères. Ainsi, nous avons repris la structure du premier réseau de neurones qui apprend le ou exclusif et l'avons adapté, remanié et complété dans bien des aspects afin que celui-ci soit capable de reconnaitre les chiffres que nous lui donnons avec le plus de précision possible.

10.3 Les challenges

Le premier challenge a été de trouver une base de donnée intéressante pour faire apprendre notre réseau. Dans un premier temps, nous avons utilisé la base de donnée MNIST qui constitue un ensemble de plus de 70000 images de chiffres manuscrits. Bien que les résultats étaient concluants sur les chiffres manuscrits, le réseau manquait de précision pour les chiffres numériques. Nous avons donc complété notre base de donnée d'apprentissage avec 10000 images de chiffres de 0 à 9 dans des polices diverses et variées. Ainsi, le réseau était fonctionnel et très efficace. Pourtant, lorsque nous avons testé sur des images issues du découpage de la grille, nous n'avions pas les résultats escomptés. En effet, dans toutes les images de notre base de données, les

chiffres étaient centrés et les images issues du découpage ne l'étaient pas tout le temps. Ainsi, à l'aide de la librairie PIL, nous nous sommes attelés à développer un script python qui génère des images de chiffres à des positions aléatoires dans la case avec des polices variées, au format que notre réseau prend en entrée.
Le second challenge fut d'implémenter le chargement de la base de donnée dans un format exploitable pour les calculs du réseau. Ainsi, une image était représenté par un tableau de valeurs entre 0 et 1 indiquant la valeur en grayscale du pixel. Ainsi, notre réseau avait 784 entrées, pour des images de 28x28 pixels.



10.4 Implémentation

Pour ne pas se perdre dans les calculs, nous avons décidé de faire un réseau avec une unique couche cachée et nous nous sommes attelées à tester un grand nombre de configurations différentes au niveau des hyperparamètres du programme tel que le nombre de neurone dans la couche cachée, le nombre de fois que nous nous entrainons sur la base de donnée et le taux d'apprentissage, afin d'obtenir un réseau de neurone le plus efficace possible.

10.5 Les difficultés rencontrées

Evidemment, le développement de ce réseau de neurone s'est accompagné d'un grand nombres de difficultés. Des problèmes encore inconnu pour nous et cela a été par conséquent très instructif et stimulant. Dans un premier temps, gérer l'allocation de mémoire dynamique de manière propre en

évitant quelconque fuite de mémoire fut un réel défi. En effet, lors de la phase d'apprentissage du réseau, l'allocation dynamique des structures de donnée contenant la base de donnée est nécessaire pour pouvoir avancer. S'ajoute à cela les nouvelles notions que vous voyions chaque semaine en cours magistral de programmation telles que les struct ou bien les pointeurs de fonctions, nous devions nous approprier de nouveaux outils rapidement afin de pouvoir exploiter leurs avantages et les mettre à profit dans notre projet. D'autre part, la difficulté théorique posée par le réseau était assez grande. En effet, les mathématiques derrière l'algorithme de backpropagation peuvent paraitre assez indigeste au premier abord, mais nous avons du nous accrocher pour le comprendre un minimum afin de pouvoir l'utiliser dans la phase d'apprentissage. D'autre part, nous avons tenté de reprendre le modèle du réseau que nous avions et de le remodeler avec une implémentation complètement matricielle mais comme nous avions des difficultés et que le temps nous était compté, nous nous sommes contentés de reprendre et améliorer au maximum le modèle déjà existant. Pour autant, le réseau est loin d'être inéfficace et il est largement suffisant pour le rôle qu'il doit tenir dans l'OCR. C'est pour cela que nous sommes très satisfaits de ce réseau malgré les difficultés rencontrés pendant notre parcours.

10.6 Sauvegarde des données après entraînement

Pour que le réseau de neurones prenne toute son utilité, nous ne voulons pas que celui-ci apprenne de nouveau à chaque fois que nous voulons l'utiliser. Par conséquent, nous avons dû implémenter une méthode de sérialiser le réseau de neurone et de le désérialiser pour pouvoir stocker les données associées à un réseau entrainé, et si besoin, pouvoir charger ses données pour éviter toute la phase d'apprentissage. Ce processus de persistance se fait via un stockage des poids, biais et valeurs d'activation des neurones du réseau dans un fichier texte. La fonction de chargement de ces valeurs s'occupe de lire ces valeurs et de les charger au bon endroit.

10.7 outils développé pour la réalisation du réseau

Pour améliorer la productivité du groupe, nous avons dévloppé quelques programmes simples pour éviter de faire des tâches répétitives à la main. Par exemple, le réseau de neurones prends en entrées des images en blanc sur noir au format 28x28. Pour ce faire, il a fallu coder un programme permettant d'inverser la couleur des pixels d'une image car la majorité des images sont en noir sur blanc et il a fallu aussi écrire un programme permettant de mettre les images à la bonne échelle pour à toutes les images récupérées sur internet d'être exploitable par le réseau de neurones.

10.8 fonctionnalités

Pour conclure, notre réseau de neurones est capable de reconnaitre des chiffres manuscrits, des chiffres numériques et le tout placer à des endroits aléatoires dans une image carrée. Ainsi, notre applicaiton devrait être en mesure de résoudre une grande majorité de sudoku que l'on peut trouver sur le web et ceux que l'on peut rencontrer de nos jours.

11 Interface Utilisateur

Après avoir longuement travaillé sur le projet au complet, il était temps de passer à la création de l'interface pour l'utilisation de notre programme. Ainsi, nous avons créé une interface simple d'utilisation, fiable et performante, afin de laisser une place importante à notre visualisation plus qu'à notre utilisation. Pour ce faire, notre interface a entièrement été créée grâce à la librairie GTK permettant l'affichage d'une interface.

11.1 Glade

Pour ce qui est de l'emplacement des boutons, des images, du logos, de la taille globale, etc... nous avons utilisé ce qui est appelé un glade. C'est un fichier écrit en HTML permettant de coordonné en données des informations visuelles sur notre interface. Ce fichier détermine tout ce qui fait notre interface, de A à Z. Nous avons opté pour un visuel sobre et efficace, ne perdant pas notre utilisateur dans de maintes et nombreuses options inutiles, et nous concentrant sur l'objectif de montrer notre avancement et notre résultat.

11.2 Boutons

Pour ce qui est des boutons implémentants notre interface visuelle nous avons optés pour peu de boutons permettant une meilleure compréhension sur l'utilisation. Un bouton de plein écran, un de fenêtré, un permettant de récupérer une image depuis un dossier, un pour afficher la grille résolue à partir de la grille donnée en paramètres, et un dernier permettant de faire petit à petit l'avancement de la grille au fur et à mesure que le préprocessing de l'image s'efffectue.

11.3 Fonctionnement

Voici comment s'utilise notre interface.
Tout d'abord il nous faut chercher une image dans un dossier. Pour ce faire, cliquer sur le bouton prévu à cet effet, puis parcourir les dossiers afin d'insérer une grille de sudoku non résolue.
Ensuite, il y a deux options. La première étant de cliquer sur le bouton solve, permettant de résoudre la grille et affichant la grille résolue. Cette méthode est rapide et permet une utilisation simple de l'interface, ne laissant pas de place au doute dans son efficacité. La seconde option est de cliquer sur le bouton "Binarize and Rotate", permettant d'afficher la première étape du préprocessing, étant la binarisation et la rotation. Cela s'effectue en arrière plan, et permet de montrer étape par étape l'utilisation que l'on fait de l'image. Ensuite un autre bouton apparait, lui permettant de réduire le bruit présent sur l'image.
Ainsi, une nouvelle image est affichée, permettant de montrer que nos filtres sont utiles et rapide. Ensuite, il ne reste plus qu'à résoudre la grille rendu, et une nouvelle image, créée de toute pièce apparait à l'écran dans l'interface, montrant la grille résolue et permettant de montrer que la détection de la grille et le passage dans le réseau de neurones fonctionne.
Ainsi, voici comment s'utilise l'interface visuelle.

11.4 Difficultés rencontrées

Pour ce qui est de l'interface visuelle, nous avons rencontré nombres de problèmes. Premièrement la librairie GTK qui ne s'installait pas correctemment, ce qui nous a pris longtemps à régler, ou bien même la création du glade qui fut bien plus fastidieuse que prévue. En effet, le glade n'est pas simple à réaliser si nous ne savons pas EXACTEMENT ce que nous souhaitons, ce fut donc difficile de créer ce glade.
Ensuite nous avons eu de gros soucis sur l'implémentation globale de l'interface visuelle, en commençant pas les boutons qui ne s'affichaient pas, qui ne fonctionnaient pas, rendant impossible son utilisation.
Ensuite l'importation de l'image nous a aussi posé grand soucis puisque nous n'arrivions pas à parcourir les dossiers pour trouver l'image, puisque le bouton ne fonctionnait point, rendu impossible l'utilisation de l'interface visuelle.
Cependant tout fut réglé dans le temps imparti et nous sommes fiers de pouvoir dire que notre interface visuelle fonctionne.

12 Ressentis

12.1 Grégoire GENET

Pour ce qui est de mon ressenti sur le projet, je l'ai trouvé très intéressant dans les points de la recherche et de la réussite face à l'adversité, malgré de grosses difficultés globales sur le point de la fusion du projet. De plus, cela m'a permis d'apprendre beaucoup de choses sur le traitement de l'image, sur les réseaux de neurones et leur implementations/utilisation dans la vie quotidienne. En effet, les réseaux de neurones sont les principes fondamentaux de l'intelligence artificielle, permettant à un être inerte de "prendre vie" et de "réfléchi" selon différents patterns et un système de pensée primitive mais fonctionnel. Cependant cela fait découler nombreuses applications dans l'intellignece artificielle, rendant cela très intéressant.
Pour ce qui est du projet en lui même, il fut difficile, étant donné que les données sur les ocr en c ne sont pas abondantes, ne nous permettant pas à nos recherches d'aboutir simplement. Cependant, nous avons tout de même réussi à rendre quelque chose quelque peu fonctionnel, permettant de résoudre plusieurs grilles de sudoku, ainsi, je suis tout de même fier du travail fourni.

12.2 Cédric DAMAIS

Mon ressenti sur le projet est que ce fut compliqué, pour la rotation automatique et la création de l'interface visuelle. J'ai eu beaoucp de mal à faire tout ce que je souhaitais, mais j'ai tout de même réussi à rendre quelque chose de nous négligeable sur ce projet.
J'ai pour autant bien aimé travailler sur ce projet mais suis tout de même un peu déçu du résultat final, montrant que nous n'avons pas assez travaillé pour finir.
Cependant je suis tout de même heureux d'en avoir fini avec ce projet.

12.3 Gabriel CALVENTE

J'ai trouvé super intéressant la création du réseau de neurone de notre projet. Malgré nombres de galères sur l'entrainement du réseau, la recherche de la database, l'implémentation de tout le projet autour de ce réseau, et j'en passe, ce fut tout de même bien intéressant de travailler là dessus.
Je suis heureux d'avoir terminé enfin ce projet, étant donné que c'était grandement chronophage, et qu'il ne laissait le temps de ne rien faire à côté. Ce fut assez cool de passer du temps à travailler en groupe et de voir ce qu'un groupe pouvait faire avec ce temps pour un OCR, quelque chose que l'on avait jamais vu ni fait. Donc c'était intéressant, mais je suis quand même content d'avoir terminé.

12.4 Clément FABIEN

Personnellement, j'ai trouvé ce projet très enrichissant. Il m'as permis d'apprendre beaucoup sur les différentes manières de chercher sur internet des algorithmes pour ne pas tomber tout le temps sur les mêmes sites. J'ai également pu beaucoup apprendre sur le traitement de l'image, notamment les filtres, la détection de ligne et la détection des bords. De plus j'ai pu apprendre comment fonctionnait un réseau de neurones et comment il pouvait nous aider dans la reconnaissance de caractère ainsi que de chiffre (dans notre cas).
Mettre le projet en commun fut une épreuve mais au final nous avons plutôt réussi et je suis content de toutes les ressources que j'ai pu exploité pour venir à bout de ce projet.
Ce projet fut très dur en lui-même étant donné qu'il a demandé énormément de temps et qu'il était assez difficile surtout lorsque la majorité de la documentation réorienté vers OpenCV. Cependant je suis heureux d'en avoir finis avec ce projet.

13 Mise en commun des programme/Fusion du travail

Pour que tout notre projet compile, nous avons dû nous atteler à la dernière tâche suivante : Fusion des programmes, afin que tout fonctionne à la suite, chaque petit programme que nous avons effectué jouant son propre rôle d'importance dans notre programme final. Ce fut une tâche drôlement fastidieuse étant donné que nous ne nous étions pas originellement coordonnés pour connaitre nos entrées et sorties de fonctions. Cependant, après de longues heures à redéfinir nos fonctions, tout connecter ensemble, nous avons réussi à avoir un programme fonctionnel, fusionnant une binarisation, une rotation automatique, une détection de grille puis un découpage des cases de la grille, un redimensionnement de ces cases afin qu'elles soient dans un format 28x28 pixels, puis un calcul du pourcentage de couleur noire afin de savoir si la case possède un chiffre ou non, puis passage de chacune des cases dans le réseau de neurones afin de détecter leur chiffre (si elle en possède un), puis la création d'une grille de sudoku à passer dans le programme de résolution, avec un '.' si la case se trouvait vide, puis la résolution de la grille, et enfin la création de l'image finale, de la grille résolue par tout ce programme. Tout cela supporté par la création d'une interface pour l'utilisateur, permettant une meilleure utilisation du matériel que nous procurons. Après tout cela, nous avons un programme fonctionnel prenant en paramètre une image de grille de sudoku, et nous rend une grille résolution de cette image de sudoku.

14 Conclusion

Ainsi, nous avons réussi tant bien que mal à fournir un travail conséquent sur ce projet d'ampleur semestrielle. Ce fut une expérience très intéressante à vivre, tant sur le point de nos compétences individuelles et nos capacités de recherches sur un problème inconnu, que sur le point de notre travail de groupe, nous ayant demandé une importante cohésion de groupe afin que tout fonctionne de la façon dont on le souhaitait. Ce projet fut donc très fastidieux, mais tout de même très enrichissant. Cela nous permettra par ailleurs de montrer notre capacité à réaliser des projets en groupe ou autonomie, fonctionnel, pouvant nous aider dans notre recherche d'emploi pour les années suivantes à l'obtention de notre diplôme. Cela étant dit, nous sommes fiers du travail rendu, et avons tous appris maintes choses en rapport avec tout ce qui fut cité au-dessus.