

PROJET OCR

Rapport de première soutenance



HUGO TERNISIEN (CHEF DE PROJET)

NICOLAS REGNIER-VIGOUROUX

CLÉMENT GUIGUET

VINCENT SAUNIER

27 OCTOBRE 2021

Table des matières

Introduction	2
1 Recherches et apprentissage	2
2 Répartition des tâches	2
3 Réseau de neurones	3
3.1 Initialisation du réseau de neurones	4
3.2 Implémentation de feedforward	5
3.3 Implémentation de Backpropagation	6
3.4 Implémentation de la fonction Query	10
4 Traitement d'image	11
4.1 Chargement d'une image	12
4.2 Suppression des couleurs	13
4.3 Binarisation de l'image	14
4.4 Détection de la grille et de la position des cases	15
4.5 Découpage de l'image (sauvegarde de chaque case sous la forme d'une image)	16
4.6 Rotation manuelle de l'image	16
5 Sudoku Solver	17
6 Programme de la prochaine soutenance	20
7 Bilan personnel	20
8 Conclusion	21
9 Bibliographie	22

Introduction

1 Recherches et apprentissage

Cette première période a débuté par plusieurs semaines d'apprentissage et de recherches. Il était indispensable que nous prenions le temps de nous renseigner sur le fonctionnement du réseau de neurones, et sur le traitement d'image.

Ainsi, pour la première soutenance, il a été convenu que Nicolas et Hugo seraient en charge de l'implémentation de la fonction XOR et du réseau de neurones. Pour ce faire, nous avons réalisé de nombreuses recherches afin de comprendre les rouages du fonctionnement d'un réseau de neurones. Dans un premier temps, nous avons pris le temps de lire le livre *"Make your own neural network"* écrit par Tariq Rashid puis avons lu la documentation donnée par Epita.

D'autres parts, Clément et Vincent se sont penchés sur les différentes façons de traiter des images, afin d'extraire des données exploitables par le réseau de neurones.

Une fois ceci fait, nous nous sommes jetés corps et âme dans cette aventure !

2 Répartition des tâches

<i>Tâches</i>	Hugo	Nicolas	Clément	Vincent
Initialisation du réseau de neurones	X			
Implémentation de Feedforward	X			
Implémentation de Backpropagation		X		
Implémentation de Query	X	X		
Suppression des couleurs			O	X
Binarisation			X	
Rotation manuelle de l'image			X	
Détection de la grille			X	
Sudoku solver				X

TABLE 1 – X - Responsable | O - Suppléant

3 Réseau de neurones

Un réseaux de neurones est un ensemble de couches de neurones reliées les unes aux autres. Celui-ci est constitué de plusieurs couches appelées "layers".

Un certain nombre de neurones forment une layer, ces neurones sont reliés à ceux des couches suivantes par des liens qui ne sont ni plus ni moins que les poids (weights). Un neurone est un point du réseau qui reçoit un signal entrant (input) et émet un signal sortant (output) qui sera transmis à la couche suivante. Il y a deux types de couches :

- Les couches observables, qui correspondent à l'entrée et à la sortie du réseau de neurones.
- Les couches cachées, connectées entre elles par des poids, réagissent de la même manière que les neurones observables en interprétant l'intensité du signal reçu puis transmis.

La puissance du réseau de neurones réside en sa capacité à se corriger au fil des époques et apprendre des ses erreurs. Dans le cadre de cette première soutenance, il nous a été demandé de réaliser la fonction OU EXCLUSIF (XOR) mais, pourra après une adaptation, résoudre des problèmes plus complexes.

Ansi, à terme, il sera possible de résoudre une grille de sudoku à partir d'une simple image grâce au traitement d'image et au réseau de neurones.

3.1 Initialisation du réseau de neurones

La première étape de la construction d'un réseau de neurones est son initialisation.

Sachant que nous aurions besoin des matrices pour réaliser celui-ci, nous avons implémenter une structure Matrix qui permettra par la suite une manipulation simple et pratique.

Structures

Pour notre XOR, deux principales structures ont été implémentées :

- La structure Matrix
- La structure neural_network

Concernant la **structure Matrix et les fonctions l'accompagnant**, elles nous permettent de les manipuler comme des array. Aussi, il nous est facile de créer des matrices, d'accéder à un élément à un certain index, de remplir ces matrices de nombres aléatoires (ce qui était nécessaire pour les matrices de poids et de biais) et même d'y changer un élément contenu dans une matrice à un index donné.

Pour ce qui est de la **structure neural_network**, elle contient toutes les variables nécessaires à l'initialisation du réseau.

Fonction d'initialisation

La fonction d'initialisation permet de générer le réseau de neurones. Les matrices de poids et de biais vont se créer et être complétées par des nombres aléatoires. Ainsi, les matrices pourront être ensuite appelées pour les fonctions comme Feedforward et backpropagation.

3.2 Implémentation de feedforward

Une réseau de neurones ne peut fonctionner sans fonction de propagation et fonction de rétropropagation. La fonction Feedforward permet de transmettre l'information à travers les couches en appliquant différentes modifications sur les entrées.

Effectivement, le principe de la fonction de propagation est de transmettre l'information de chaque neurone de manière pondérée, c'est-à-dire que tous les neurones n'auront pas les mêmes contributions dans la propagation des signaux.

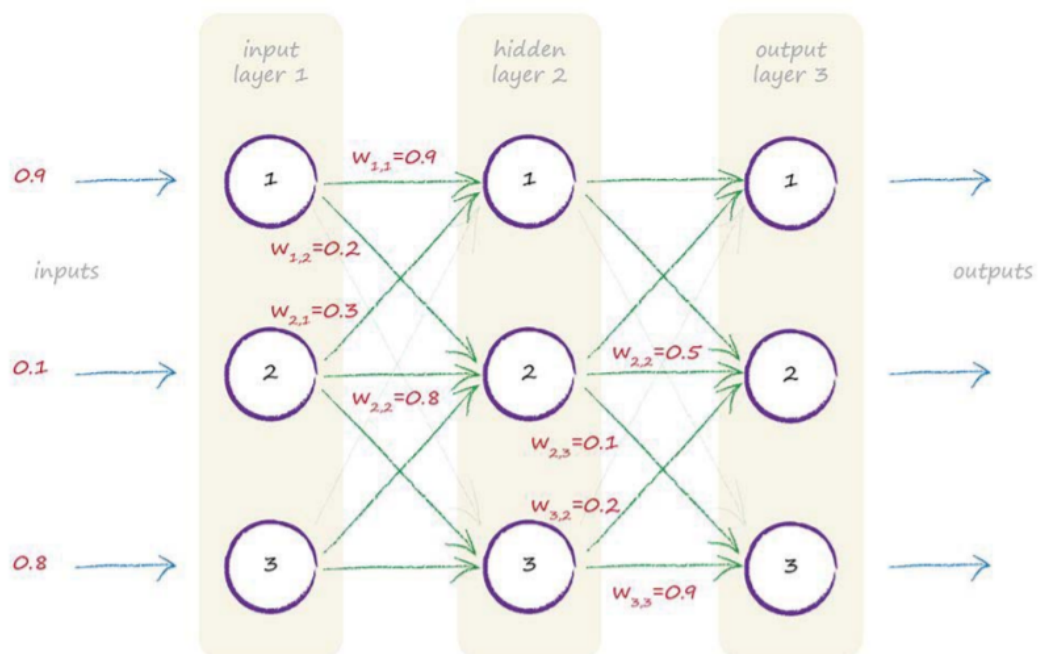


FIGURE 1 – Propagation des signaux

Dans un premier temps, nous traitons la première couche du réseau de neurones à savoir l'input layer. Pour ce faire, il est nécessaire de prendre la matrice de poids (W_{input_hidden}) de la layer puis de la multiplier par les éléments donnés en entrée (I).

Ceci nous donnera une matrice $X_{input_hidden} = W_{input_hidden} \times I$. Prenons enfin la matrice de biais de la layer actuelle, notée B_{input_hidden} . La fonction d'activation sigmoid sera appliquée de la manière suivante pour chaque

élément de la matrice X_{input_hidden} :

$$O_{hidden} = \sigma(X_{input_hidden} + B_{input_hidden}) \quad (1)$$

L'output de la première couche (hidden layer) servira alors d'input pour la deuxième layer, c'est-à-dire la hidden_output layer. Ainsi, nous procéderons au même traitement effectué au préalable sur l'input_hidden layer.

Finalement, nous obtiendrons un résultat (Output) qui sera ensuite utilisé pour calculer la marge d'erreur entre le résultat escompté et le résultat effectif. Ce processus de calcul de l'erreur sera déterminant pour effectuer la rétropropagation de l'erreur (Backpropagating).

3.3 Implémentation de Backpropagation

Pour améliorer le réseau de neurones et se rapprocher du résultat espéré, il faut modifier les poids de chaque couche et leurs biais. Ce sont les deux seuls leviers existant pour modifier le réseau de neurones.

Pour calculer la modification des poids et des biais, il y a plusieurs solutions. Nous avons utilisé la méthode de la descente progressive du gradient. Elle se modélise par des petits sauts pour trouver le minimum d'une fonction.

La tangente du point où l'on se trouve définit la direction et la taille du saut à venir. Si la tangente est positive, on fait des sauts vers l'arrière et donc si la tangente est négative on fait des sauts vers l'avant. Plus le coefficient directeur de la tangente est élevé, plus le saut est grand et inversement jusqu'à ce que l'on trouve le minimum de la fonction. Mais on peut tomber dans des impasses. C'est pour ça qu'on fait plusieurs itérations pour trouver le vrai minimum.

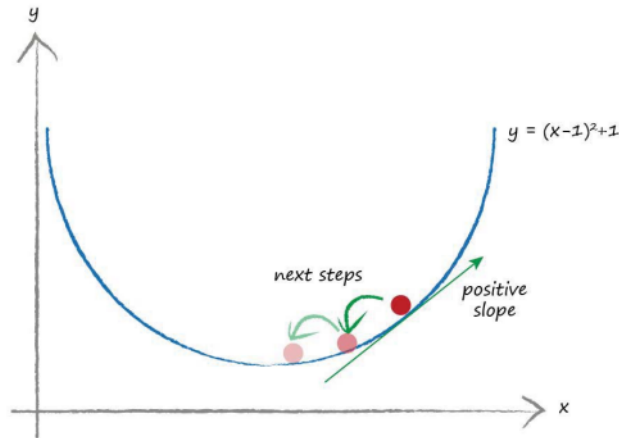


FIGURE 2 – Descente progressive sur la fonction $y=(x-1)^2 + 1$

Donc maintenant que nous avons posé les bases de la théorie, il nous faut passer à la pratique. On va commencer par rechercher la fonction que l'on va étudier. Et cette fonction s'appelle le coût. Il y en a plusieurs possibles mais elles partent sur la même base, à savoir la somme des valeurs obtenues moins les valeurs visées. Nous utiliserons la fonction qui est la somme des valeurs obtenues moins les valeurs visées au carré. Elle est décrite ci-dessous avec \mathbf{j} étant l'index du noeud, \mathbf{a} la sortie du noeud et \mathbf{y} la sortie espérée du noeud.

$$C_0 = \sum_{j=0}^{n_L-1} (a_j^{(L)} - y_j)^2$$

La fonction de coût utilisé dans notre réseau de neurones (2)

Pour modifier la fonction de coût qui est représentée par l'erreur, il faut modifier la sortie du noeud qui n'est ajustable que par les poids et les biais. Cette relation est décrite dans l'arbre ci-dessous. On peut y voir que pour chaque petit mouvement de la fonction de coût, les modifications sur les autres termes augmentent selon la profondeur du terme sur l'arbre. On représente mathématiquement ces modifications par des dérivées.

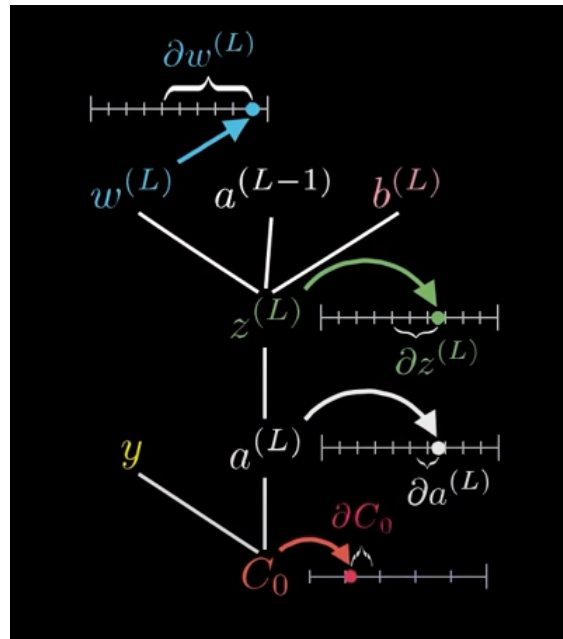


FIGURE 3 – Arbre de rétropropagation

La modification d'un poids consiste à soustraire à l'ancien poids la dérivée du coût, le tout divisé par la dérivée du poids multipliée par un taux d'apprentissage permettant au réseau de neurone de ne pas être en sur ou en sous-apprentissage. La modification d'un biais se fait de la même façon à l'exception que l'on soustrait la dérivée du coût par la dérivée du biais.

Le plus difficile arrive : comment on calcule la dérivée du coût par la dérivée du poids ?

On trouve cette formule ci-dessous (3) avec \mathbf{z} qui correspond à l'entrée de chaque noeud, \mathbf{a} correspondant à la sortie de chaque noeud, \mathbf{w} qui correspond aux poids et \mathbf{L} qui représente la couche où se trouve le terme.

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

(3)

La dérivée de l'entrée sur la dérivée du poids est égale à la sortie du noeud précédent. La dérivée de la sortie sur la dérivée de l'entrée est égale à la dérivée de la fonction d'activation qui est pour nous la fonction sigmoid. Ce qui donne $\text{sigmoid}(\mathbf{z})(1-\text{sigmoid}(\mathbf{z}))$. Et enfin la dérivée du coût sur la dérivée

de la sortie est égale à l'erreur de chaque noeud. En finalité, la dérivée du coût par la dérivée du poids est égale à la sortie du noeud précédent * l'erreur* la dérivée de la sigmoid.

Pour calculer les nouveaux biais, la dérivée du coût sur la dérivée du biais est équivalente à la formule vue plus haut sauf qu'on remplace la dérivée des poids par la dérivée du biais.(2) Et donc la dérivée de l'entrée par les poids donne 1. La dérivée du coût sur la dérivée du biais est finalement égale à l'erreur*la dérivée de la sigmoid.

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

(4)

Pour trouver les erreurs pour toutes les couches, on va chercher l'erreur pour la dernière couche puis la propager à travers le réseau. On ne calcule l'erreur que sur la dernière couche car on ne connaît que le résultat espéré pour les noeuds finaux.

On a vu que l'erreur est égale à la dérivée du coût sur la dérivée de la sortie. La fonction de coût pour chaque noeud étant égale à la soustraction de la sortie obtenue par la sortie souhaitée au carré ce qui donne 2 fois la soustraction du résultat obtenu par celui espéré.

Ensuite pour propager l'erreur aux autres noeuds et couches, on va propager proportionnellement l'erreur d'un noeud par rapport aux poids qui viennent vers lui. C'est à dire qu'un noeud recevant comme poids $W1=2.0$ et $W2=1.0$, le noeud dont vient le poids $w1$ aura donc 2 fois plus de l'erreur du noeud dont arrive $W1$ que celui de $W2$. Puis on additionne l'erreur venant de chaque poids pour obtenir l'erreur du noeud

Et on fait ça pour chaque noeud de chaque couche sauf pour la première couche, la couche d'entrée car les erreurs obtenues ne serviront à aucun calcul.

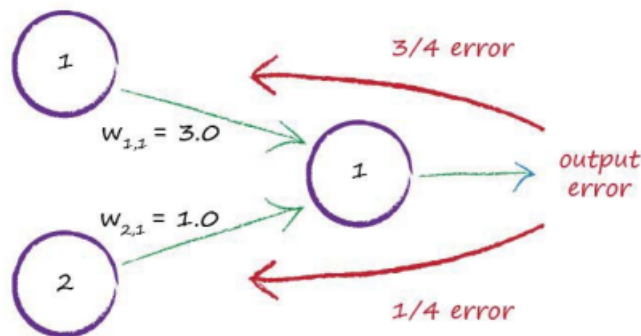


FIGURE 4 – Rétropropagation de l'erreur

3.4 Implémentation de la fonction Query

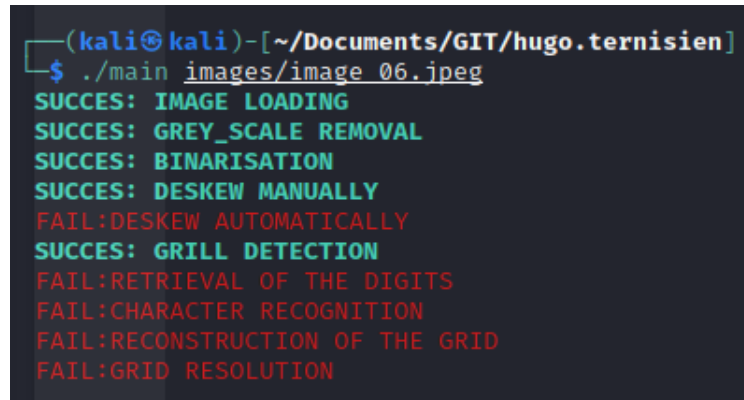
Une fois entraîné, le réseau de neurone nous retournera un résultat. Ici, dans le cadre de la première soutenance de ce projet, nous devons implémenter la fonction OU-EXCLUSIF. Ainsi, nous donnerons deux entrées au réseau de neurones et il nous renverra la sortie correspondante.

Le résultat ne pouvant jamais être égal à 1 ou 0, nous considérerons que le résultat est fiable car la valeur renvoyée aura une précision de 99,4%.

Effectivement, nous aurons 0.9949 lorsque le résultat attendu sera de 1 et 0,0041 pour 0.

4 Traitement d'image

Une fonction main a été implémentée pour centraliser le processus du traitement d'image. Elle charge une image donnée en paramètre, lui applique les filtres désirés, puis l'enregistre. Cela nous permet de pouvoir tester plus rapidement et plus facilement notre code, tout en récapitulant l'avancé du projet. Voici donc son application :



```
(kali㉿kali)-[~/Documents/GIT/hugo.ternisien]
$ ./main images/image 06.jpeg
SUCCES: IMAGE LOADING
SUCCES: GREY_SCALE REMOVAL
SUCCES: BINARISATION
SUCCES: DESKEW MANUALLY
FAIL:DESKEW AUTOMATICALLY
SUCCES: GRILL DETECTION
FAIL:RETRIEVAL OF THE DIGITS
FAIL:CHARACTER RECOGNITION
FAIL:RECONSTRUCTION OF THE GRID
FAIL:GRID RESOLUTION
```

FIGURE 5 – Résultat actuel de la fonction main

4.1 Chargement d'une image

Afin de gérer les images, nous avons fait le projet à l'aide de la librairie SDL2. Charger et sauvegarder une image était donc simple, puisqu'il existe des fonctions automatisant ce processus. Voici ci-dessous un exemple de chargement d'une image, et de la sauvegarde.

```
image = IMG_Load(argv[1]);
```

FIGURE 6 – Fonction de chargement d'une image

```
IMG_SavePNG(image, "greyscale.png");
```

FIGURE 7 – Fonction de sauvegarde d'une image

Voici l'image sur laquelle vont être appliqués les filtres.

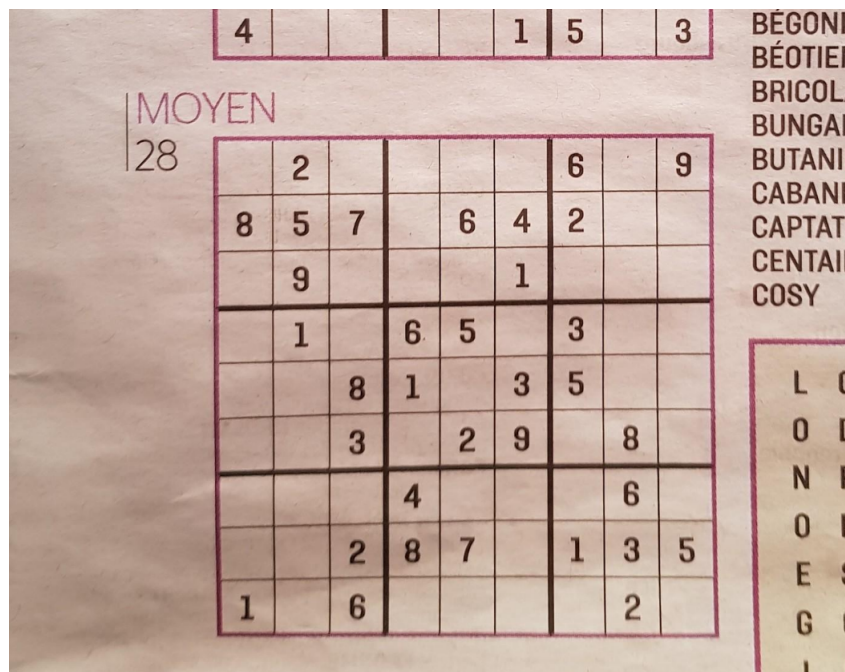


FIGURE 8 – Image de base

4.2 Suppression des couleurs

La fonction grayscale a été faite sous SDL2, avec l'aide du tp n°3. Le principe ayant été vu en TP, la difficulté de cette fonction n'était pas de comprendre son principe qui est plutôt simple, mais plutôt d'écrire les fonctions auxiliaires sous SDL2.

Les fonctions qui gèrent les pixels, dans le fichier pixeloperations, sont les mêmes que celle du tp.

Mais les fonctions pour afficher l'image et la charger, nécessaires pour pouvoir tester la fonction étaient différentes du tp pour la version SDL2.

Cette fonction est testée dans le main en chargeant une image et appelant la fonction avec cette image.

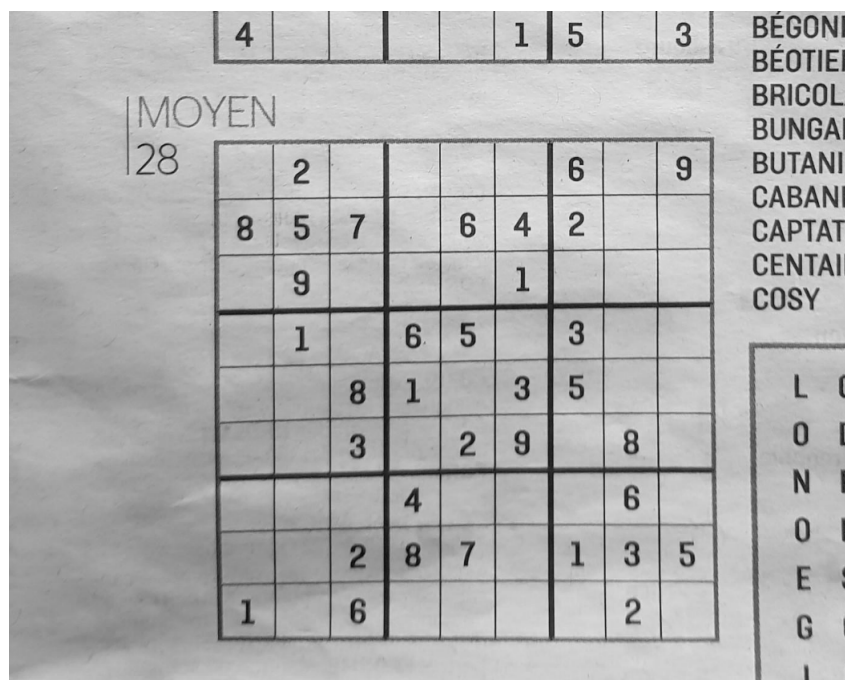


FIGURE 9 – Image de base en greyscale

Ici, on voit que la fonction, qui prend l'image du sudoku en couleur en entrée, renvoie bien une image de ce sudoku en ton de gris.

4.3 Binarisation de l'image

La binarisation accentue la différence entre le fond et le texte. Elle consiste à différencier les pixels que l'on va mettre en noir, et en blanc. Le principe de la binarisation d'une image est donc de prendre la moyenne de tous les pixels, pour la comparer avec la valeur de chaque pixel. Nous distinguerons donc les pixels plus grands que cette moyenne, qui seront en noir, et ceux plus petits, en blanc. L'inconvénient de cette méthode est que si une zone de l'image est plus sombre que le reste, alors le résultat sera faussé. Pour palier ce problème, nous avons divisé l'image en un certain nombre de carrés, et nous traitons la moyenne à l'intérieur de ces carrés. En effet, les zones d'ombres seront donc traitées à part, pour avoir un résultat cohérent. Voici l'image renvoyée par notre algorithme de binarisation.

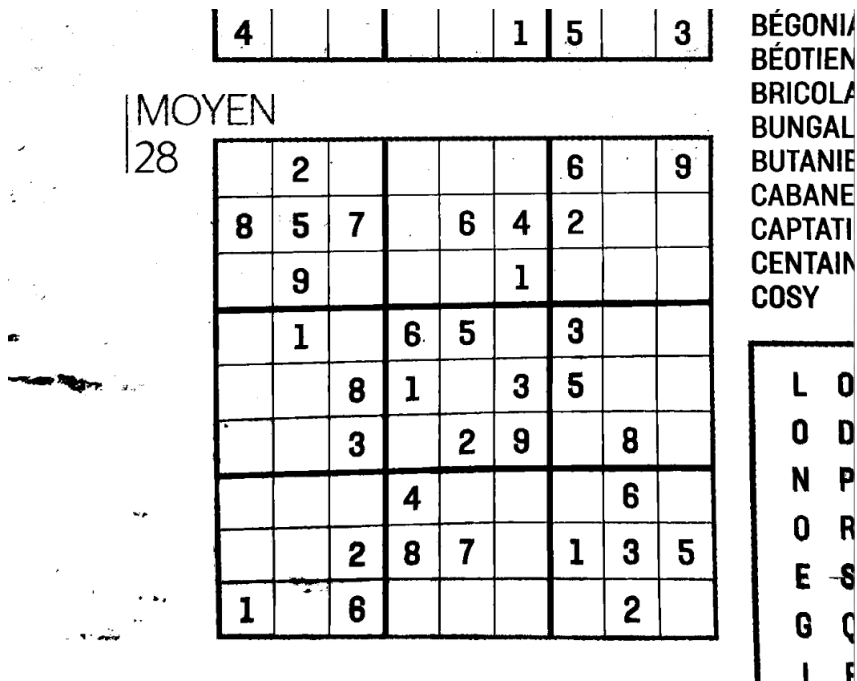


FIGURE 10 – Image de base binarisée

4.4 Détection de la grille et de la position des cases

Afin de détecter la grille, nous allons parcourir toute les lignes et colonnes de pixels de l'image. Les pixels étant soit noir, soit blanc suite à la binarisation de l'image, il suffit de détecter sur l'image des suites de pixels, formant ainsi des lignes, et des colonnes. Nous obtenons suite à l'application de ce filtre ce résultat ci-dessous, où nous voyons clairement la grille, détectée.

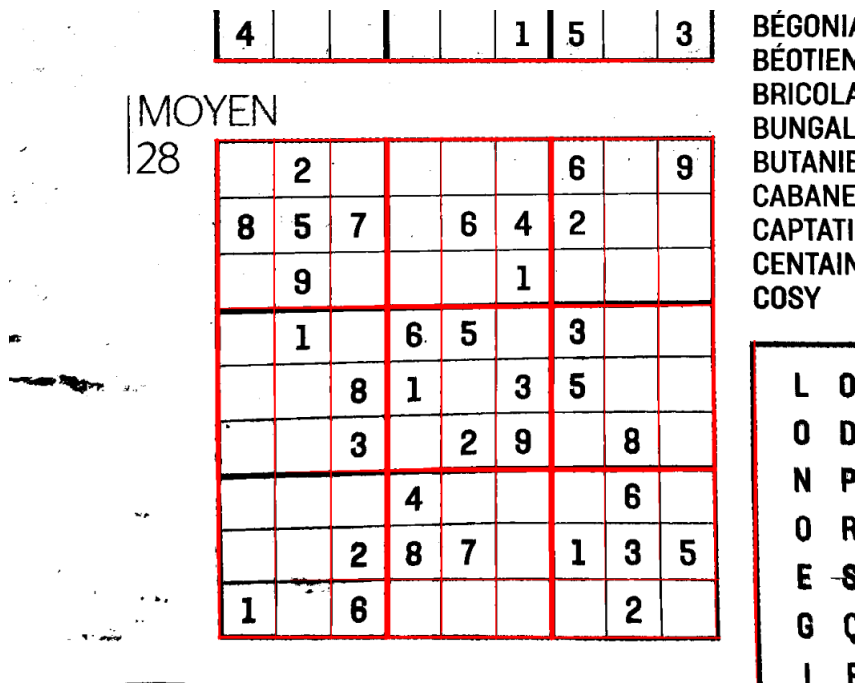


FIGURE 11 – Image de base avec la grille détectée

4.5 Découpage de l'image (sauvegarde de chaque case sous la forme d'une image)

Nous avons manqué de temps pour faire la programmation de cette partie. Néanmoins, nous savons comment procéder pour cette partie. Vu que nous avons détecté la grille, nous allons facilement trouver les coordonnées de la grille, puis celles des différentes cases. Ainsi, nous aurons les coordonnées, et les dimensions des 81 cases, et nous allons pouvoir enregistrer chaque case individuellement.

4.6 Rotation manuelle de l'image

Avant d'appliquer certains filtres sur l'image, il faut qu'elle soit bien droite. Or, l'image peut être une photo prise de travers. Nous avons donc dû implémenter une fonction rotate, qui tourne l'image d'un angle donné. Il nous suffit de calculer la taille optimale de la nouvelle image, puis d'utiliser la matrice de rotation ci-dessous pour appliquer la rotation.

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}. \quad (5)$$

Notre fonction prend comme paramètre l'image, et un angle. Celui-ci est en degrés, qui sera converti en radian lors de l'application de cette fonction.

5 Sudoku Solver

Le Sudoku Solver a été réalisé en trois parties :

- D'abord la transformation du fichier txt en array de int
- Ensuite la resolution du sudoku en prenant ce tableau de int
- L'affichage du tableau de int dans un fichier qui prend l'extension .result

D'abord la transformation du fichier txt en array de int :

Le fichier de base sur lequel tester la fonction est un fichier txt qui représente un sudoku vide. Pour pouvoir le manipuler on doit le passer en tableau de int.

```
| ... ..4 58.  
| ... 721 ..3  
| 4.3 ... ..  
  
| 21. .67 ..4  
| .7. ... 2..  
| 63. .49 ..1  
  
| 3.6 ... ..  
| ... 158 ..6  
| ... ..6 95.
```

FIGURE 12 – Sodoku non résolu

Grâce à une fonction, on peut faire passer le fichier ci-dessus en array de int à deux dimensions, en mettant les chiffres dans les bonnes case du tableau et en remplaçant les points par des 0 dans le tableau.

Pour le principe de la fonction, on lit le fichier jusqu'à sa fin en lisant caractères par caractères jusqu'à atteindre la fin du fichier.

On vérifie que chaque caractère n'est pas un espace ou un retour à la ligne, auquel cas on ne fait rien.

Si le caractère est un point on ajoute un 0 au bon endroit dans le tableau.

Si le caractère est bel et bien un chiffre, on le met au bon endroit du tableau en le convertissant en int.

Ensuite la résolution du sudoku en prenant ce tableau de int :

On utilise une fonction pour afficher le tableau, et une fonction pour vérifier que la case du tableau sur laquelle on se trouve respecte les règles du sudoku.

C'est à dire pas 2 fois le même chiffre dans la même colonne, dans la même ligne, ou dans le même cellule de 3*3 cases.

Enfin, la fonction pour résoudre le sudoku vérifie si la fin des lignes ou des colonnes est atteinte.

Pour chaque cellule du tableau elle vérifie si elle respecte les règles, et met alors un chiffre sur la case.

La fonction est ensuite appelée récursivement pour constituer toutes les cases du tableau.

L'affichage du tableau de int dans un fichier qui prend l'extension .result :

Lorsque le sudoku est résolu, une fonction va prendre le tableau en entrée et va afficher le sudoku résolu en sortie.

Pour cela on crée un nouveau fichier dans lequel on met le sudoku non résolu, on prend l'ancien fichier qui contient toujours le sudoku non résolu et on lui ajoute l'extension .result comme demandé. On résoud alors le sudoku dans ce fichier.

On l'affiche, 3 colonnes par 3 et on affiche des espaces entre les colonnes par 3.

Lorsqu'on atteint la ligne 2 ou 5 on saute une ligne pour respecter l'affichage de retour demandé.

Si le sudoku n'est pas résoluble, on affiche alors "no solutions exists".

Le sudoku est alors affiché comme ceci :

```
$ ./solver grid_00
$ ls
grid_00 grid_00.result 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
```

FIGURE 13 – Sodoku résolu par le Solver

Comme on peut observer sur l'image ci-dessus, la compilation du sudoku solver crée un fichier result qui affiche alors le sudoku résolu.

6 Programme de la prochaine soutenance

D'ici la prochaine soutenance, nous comptons implémenter les fonctionnalités suivantes, afin de finaliser le projet :

- la détection des cellules
- La reconnaissance des caractères
- La rotation automatique de l'image
- L'interface graphique

Dès à présent il est très probable qu'une majorité des fonctionnalités que nous avons implémentées aura besoin d'être remodelée afin de répondre parfaitement aux objectifs finaux.

7 Bilan personnel

Ce projet est passionnant et m'a permis de me plonger dans tous les algorithmes de rétropropagation qui permettent la création du réseau de neurones. Ce fut complexe de comprendre le fonctionnement d'un réseau de neurones mais ces connaissances obtenues lors de ce projet me seront sûrement utiles pour mes projets futurs. De plus, avoir créé une intelligence Artificielle complexe et fonctionnelle est tellement gratifiant.

Nicolas Regnier-Vigouroux

Cela fait plus d'un mois que nous avons commencé ce travail de groupe : l'OCR. Ce projet m'a beaucoup appris par rapport au traitement d'image en C. Le démarrage était plutôt compliqué, le C étant pour nous un nouveau langage de programmation. Devant le délai plus court que les précédents projets, j'ai appris que la cohérence du groupe, et le raisonnement à plusieurs permettent d'avancer plus rapidement, et plus efficacement que si l'on est seul. Devant la progression et l'évolution du projet en seulement un mois, je suis très motivé pour porter à bien ce projet.

Clément Guiguet

Ce projet m'a permis d'apprendre et de découvrir beaucoup de nouvelles choses. Déjà, pour la partie du traitement d'image, la manipulation des fonctions qui servent à afficher une image, effacer une fenêtre, et manipuler des pixels a été très intéressante à découvrir et me permettra d'aborder plus facilement la suite du projet. Ensuite, les fonctions du sudoku solver, et surtout les conversions de fichier en tableaux et de tableaux en image, sont des fonctions que j'ai trouvé intéressantes à coder et qui me seront sûrement utiles à l'avenir.

Vincent Saunier

Ce projet a été pour moi l'opportunité de découvrir de certaines choses et d'en comprendre d'autres. J'ai trouvé tout à fait passionnant le livre que j'ai lu concernant le fonctionnement des réseaux de neurones. Certes, au premier abord, cela peut paraître un peu obscur et les notions abordées étaient parfois difficiles à cerner mais je pense avoir compris les rouages des Neural Network. Je suis motivé pour la suite de ce projet, que je compte mener à bien !

Hugo Ternisien

8 Conclusion

Nous concluons ce rapport avec un récapitulatif de toutes les avancées que nous avons faites depuis le début de ce projet. Tout d'abord, ce projet a été très constructif pour tous les quatre. Nous avons amélioré notre capacité à travailler en groupe. Nous avons du retard sur le traitement d'images, mais le rattraper ne sera pas compliqué, notamment grâce à notre travail de groupe. Aujourd'hui, nous connaissons la théorie du fonctionnement d'un réseau de neurones, ainsi que le traitement d'image. Ces outils nous seront utiles dans notre futur, puisqu'ils sont utilisés dans de plus en plus de domaines informatiques.

9 Bibliographie

Dans cette section, nous dressons la liste des sources que nous avons utilisées afin de réaliser les premières fonctionnalités de cet OCR.

- *"Make your own neural network"* écrit par Tariq Rashid [livre broché].
- Neural networks and Deep Learning [En ligne] <http://neuralnetworksanddeeplearning.com/>
- Réseau de neurones artificiels [En ligne] https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels
- Calcul de la rétropropagation | Apprentissage profond, chapitre 4 [En ligne] <https://www.youtube.com/watch?v=tIeHLnjs5U8&t=294s>
- Librairie SDL2 <https://wiki.libsdl.org/>
- Lecture d'un fichier en c [En ligne] <https://youtu.be/Czbmehljpgo>