



OCR

Détection et résolution d'un Sudoku

Mathieu NEUMAR
Mathis KOPENOU
Jessy COURTEMANCHE
Satya SHETTY
Enzo JEHL

mathieu.neumar
mathis.kpenou
jessy.courtemanche
satya1.shetty
enzo.jehl

Table des matières

1 Présentation du projet	1
1.1 Le projet : résolution d'un sudoku	1
1.2 Notre groupe	1
1.2.1 Mathieu NEUMAR	2
1.2.2 Mathis KPENOU	2
1.2.3 Jessy COURTEMANCHE	2
1.2.4 Satya SHETTY	2
1.2.5 Enzo JEHL	3
1.3 Répartition des tâches	4
2 Prétraitement de l'image	5
2.1 Prétraitement	5
2.1.1 Grayscale	5
2.1.2 Suppression du bruit	7
2.1.3 Threshold adaptatif	8
2.2 Rotation manuelle de l'image	10
3 Segmentation	12
3.1 Détection de la grille	12
3.1.1 Transformée de Hough	12
3.1.2 Traitement de l'accumulateur	13
3.2 Découpage de l'image	13
3.2.1 Traitement des intersections	14
3.2.2 Nouvelle surface	14
4 Réseau de neurones (XOR)	15
4.1 Qu'est ce qu'un réseau de neurones ?	15
4.1.1 Le concept de XOR	16
4.1.2 L'implémentation du Xor	16
5 Résolution du sudoku	20
5.1 Partie 1	20
5.1.1 Récupération puis écriture du sudoku dans un fichier	20
5.1.2 Résolution du sudoku	20
6 Conclusion	21
6.1 Attentes et avancement	21
6.2 Conclusion	21
Annexes	24
Annexe 1	24
1 Sources	24

Chapitre 1

Présentation du projet

1.1 Le projet : résolution d'un sudoku

L'objectif de ce projet est de réaliser un logiciel de type OCR (Optical Character Recognition) qui résout une grille de sudoku. Notre application prendra donc en entrée une image représentant une grille de sudoku et affichera en sortie la grille résolue. Dans sa version définitive, elle 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 sera également sauvegardée. Notre application possédera une capacité d'apprentissage, qui pourra être séparé de la partie principale, et qui permettra d'entraîner notre réseau de neurones, puis de sauvegarder et de recharger le résultat de cet apprentissage. Les différentes étapes sont donc : le chargement d'une image, la suppression des couleurs, le prétraitement, la détection de la grille, la détection des cases de la grille, la récupération des chiffres présents dans les cases, la reconnaissance de caractères, la reconstruction de la grille, la résolution de la grille, l'affichage de la grille résolue et enfin la sauvegarde de la grille résolue.

1.2 Notre groupe

Notre groupe est composé de Satya Shetty, Enzo Jehl, Mathis Kpenou, Mathieu Neumar et Jessy Courtemanche. La première étape était de créer ce groupe. Nous avons donc décidé de former un groupe de 5 personnes. Ayant 5 personnes, la répartition des tâches était compliquée, nous avons Mathieu qui s'occupe du prétraitement de l'image, Mathis qui effectue la rotation manuelle de l'image ainsi que l'enregistrement des poids du réseau de neurones, Jessy qui accomplit la segmentation, Enzo qui résout le sudoku grâce à un algorithme de backtracking, et finalement Satya qui a modélisé un réseau de neurone Xor. Pour mener à bien ce projet, nous avons décidé de faire une réunion par semaine en présentiel afin de voir l'avancement de chacun car le but était de s'avancer le plus possible, pour

pouvoir se concentrer sur les partiels lors de la semaine de révision.

1.2.1 Mathieu NEUMAR

Comme la majorité des élèves à Epita je suis passionné par l'informatique, tout ce qui touche aux nouvelles technologies et à la cybersécurité. J'ai déjà réalisé quelques projets durant mes années de lycée (sous Python), le projet S2 la première année et le même projet OCR l'an passé. J'ai eu la chance de découvrir de nombreux domaines étant jeune ce qui me permet d'être assez flexible dans le développement. Je refais l'OCR cette année, et afin d'apprendre et de me développer dans de nouveaux domaines, j'ai décidé de travailler sur la partie du prétraitement de l'image. Je serais également le chef de projet donc je ferais en sorte de garder mon équipe motivée avec un planning bien construit et réalisable. Étant chef de projet, je me sens responsable d'assister les autres membres dans leurs tâches.

1.2.2 Mathis KPENOU

De nos jours, les nouvelles technologies sont très présentes dans notre entourage. Il était tout naturel pour moi de m'y intéresser. Depuis quelques années maintenant, j'ai pu développer mes connaissances en informatique sur différents langages de programmation à travers plusieurs projets. C'est ainsi que pour mener à bien ce projet, j'ai pu réinvestir certaines de mes connaissances. Comme nous sommes cinq sur ce projet qui contient quatre parties majeures, je suis sur deux parties différentes qui sont le prétraitement et le réseau de neurones. Pour le prétraitement je me suis occupé de la rotation manuelle de l'image et pour le réseau de neurones je me suis occupé de l'enregistrement des poids sur un fichier texte.

1.2.3 Jessy COURTEMANCHE

Au cours de mes études, j'ai développé une attirance pour l'informatique. De plus, ayant des membres de ma famille étant dans le domaine de l'informatique j'ai toujours baigné dans ce domaine. Lors de mon projet S2 qui s'est déroulé l'année passée, j'ai su développer de nombreuses qualités telles que le travail d'équipe ainsi que de nombreuses connaissances sur la programmation. J'ai pu apporter par mes connaissances des solutions aux problèmes rencontrés. Je trouve le sujet de ce projet très intéressant ce qui me motive pour donner le meilleur de moi-même.

1.2.4 Satya SHETTY

Je suis passionné de nouvelles technologies et d'informatique, celle-ci est présente dans notre quotidien. Pour ce projet j'ai décidé de m'attaquer au réseau de neurones. Le groupe nominal réseau de neurone semble assez complexe de prime abord

mais je me suis déjà renseigné. Cela ne semble pas si compliqué. Je compte m'investir dans ce projet au risque de devoir enchaîner de nombreuses nuits blanches. Je compte m'accrocher et je ferai de mon mieux pour mener à terme ce projet.

1.2.5 Enzo JEHL

Etant élève à EPITA je suis passionné d'informatique comme la plupart des élèves. Cependant, je n'ai pas énormément d'expérience dans ce type de projet si ce n'est la réalisation d'un jeu vidéo l'année dernière. Mais ce projet étant quelque chose de très différent il me permettra donc de découvrir de nouvelles choses ce qui me motive énormément. Lors de ce projet je m'occuperais d'abord de la résolution du Sudoku puis ensuite de l'interface finale du solver.

1.3 Répartition des tâches

Voici un tableau (cf. fig. 1.1) récapitulatif de notre analyse de l'existant...

	Mathieu	Mathis	Jessy	Satya	Enzo
Prétraitement de l'image	X				
Rotation manuelle de l'image		X			
Segmentation			X		
Réseau de Neurones XOR				X	
Enregistrement des poids		X			
Résolution du Sudoku					X

FIGURE 1.1 – Tableau récapitulatif des solutions

Chapitre 2

Prétraitement de l'image

Cette partie est cruciale pour le développement de notre OCR. En effet, le prétraitement de l'image permet de simplifier les prochaines parties. Une photographie floue d'une grille de sudoku pliée et en couleur serait bien trop complexe à étudier pour le réseau de neurones ou même la segmentation. L'objectif de cette partie sera donc de rendre l'image la plus lisible possible, notamment pour la détection de ligne mais également pour que le réseau de neurones détecte bien les différents caractères du sudoku.

Pour cela, nous avons réalisé plusieurs transformations à l'aide de différents filtres et opérations que nous allons détailler. Afin d'avoir une représentation plus claire du suivi des différentes étapes de cette partie, nous utiliserons la même image en tant qu'exemple.

2.1 Prétraitement

2.1.1 Grayscale

En premier lieu, il semblait évident de passer l'image en nuance de gris, afin de simplifier les opérations futures. L'opération ci-dessous permet d'obtenir la nuance de gris de chaque pixel en fonction de leur valeur RGB.

$$\text{Uint32 pixel} = 0.2126 * \text{r} + 0.7152 * \text{g} + 0.07822 * \text{b}$$

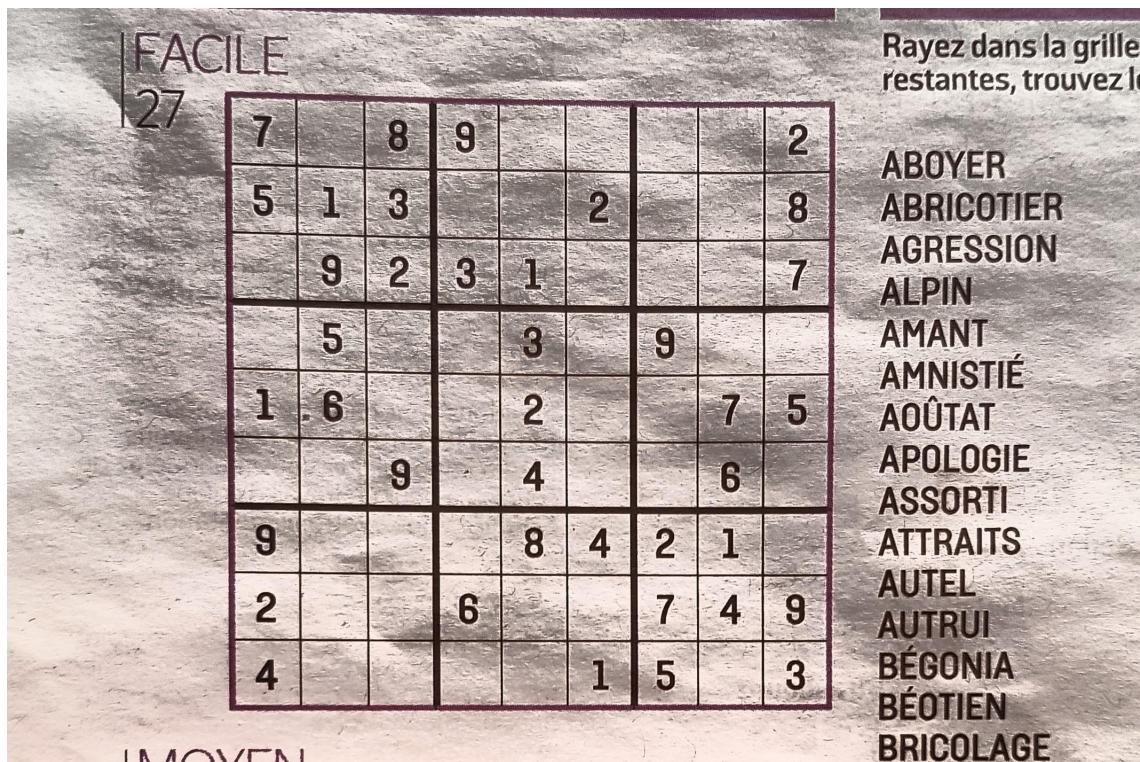


Figure 2.1.1 - Image de base

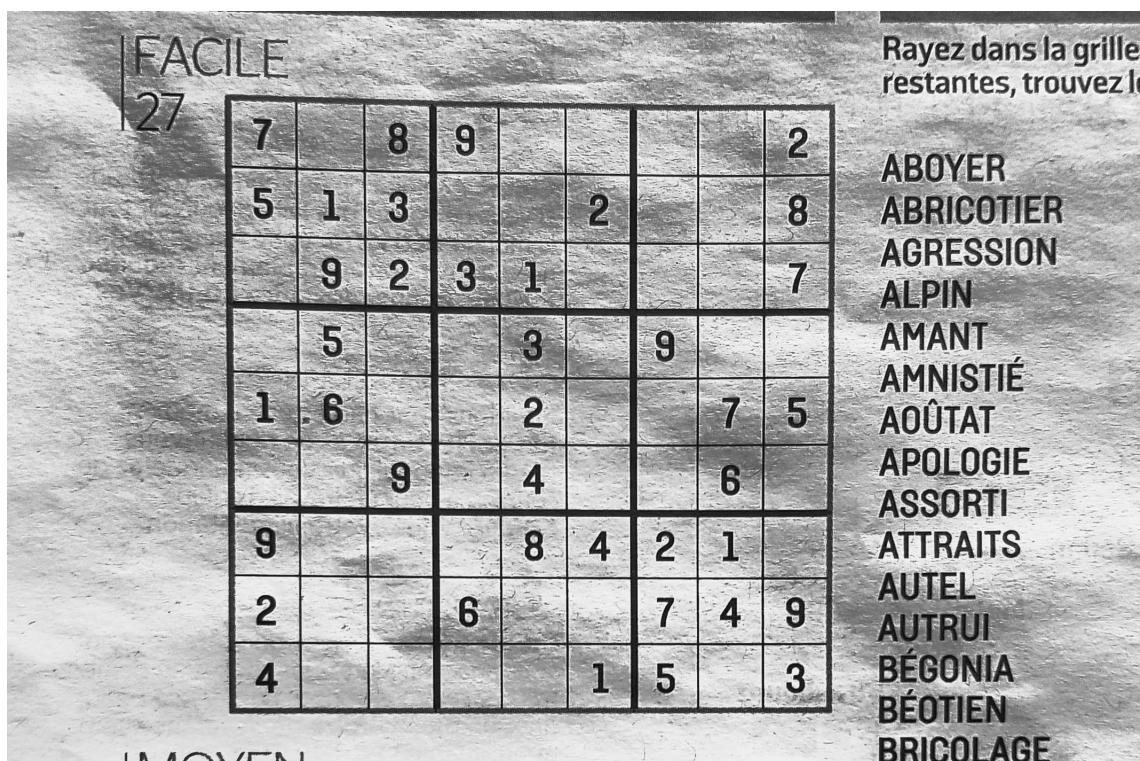


Figure 2.1.2 - Application du grayscale

2.1.2 Suppression du bruit

Afin d'avoir l'image la plus lisible et utilisable possible, nous avons décidé d'appliquer un filtre médian, qui nous permettra de réduire le bruit de l'image. Ce filtre récupère la valeur des pixels voisins, qu'il va trier de manière croissante pour ensuite modifier la valeur du pixel par celle de la valeur médiane de la liste triée.

Cela permettra d'éviter des erreurs de prétraitement :

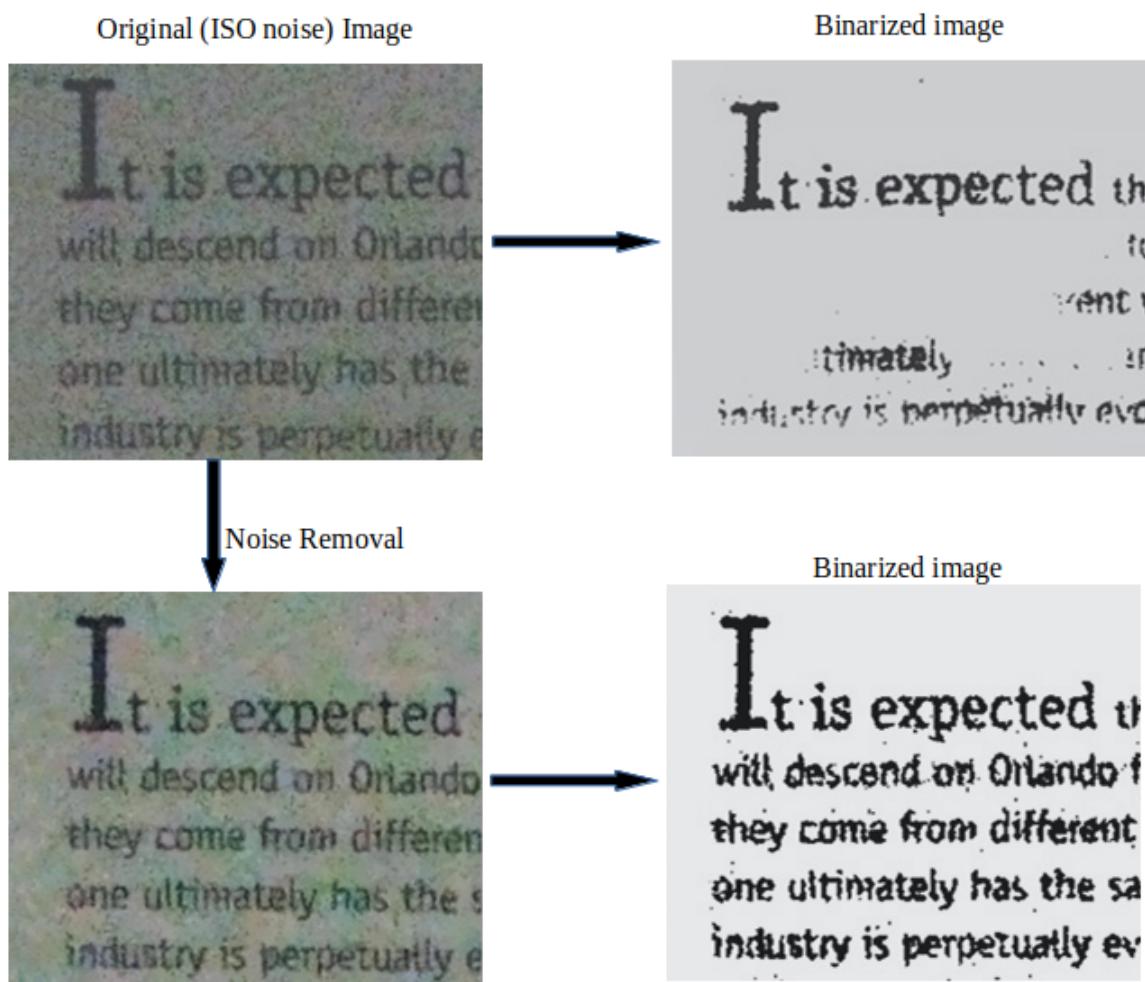


Figure 2.1.3 - Comparaison d'une image binarisée avec et sans suppression de bruit

Application du filtre médian sur le pixel avec la valeur 52 :

0	9	10
12	52	7
5	11	13

FIGURE 2.1 – Voisins du pixel 52

0	5	7	9	10	11	12	13	52
---	---	---	---	----	----	----	----	----

FIGURE 2.2 – Tableau triée du pixel et de ses voisins

0	9	10
12	10	7
5	11	13

FIGURE 2.3 – Nouvelle valeur du pixel

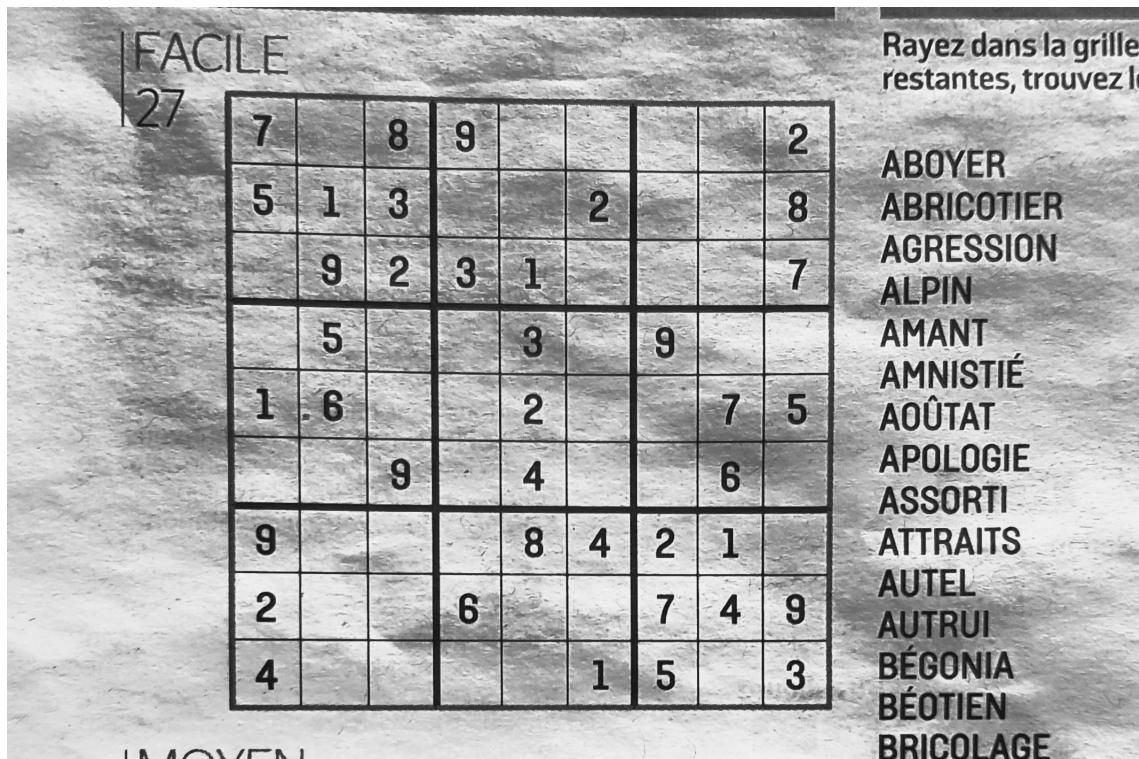


Figure 2.4 - Suppression du bruit

2.1.3 Threshold adaptatif

Le Threshold adaptatif est la partie la plus importante du prétraitement de l'image. Son objectif est de calculer un seuil adapté à chaque image, en effet, si nous prenions le même seuil pour chaque image, cela ne fonctionnerait probablement pas. Si le seuil est trop faible et que l'image est trop foncé, cette dernière finira toute noire, et à contrario si le seuil choisi est trop élevé et que l'image est trop clair, cela rendra une image toute blanche. Pour contrer cela, l'algorithme implémenté permet de calculer un seuil adapté à l'image grâce à l'image intégrale qui nous permettra de calculer une moyenne pour définir le seuil adapté.

$$s(x, y) = i(x, y) + s(x-1, y) + s(x, y-1) - s(x-1, y-1)$$

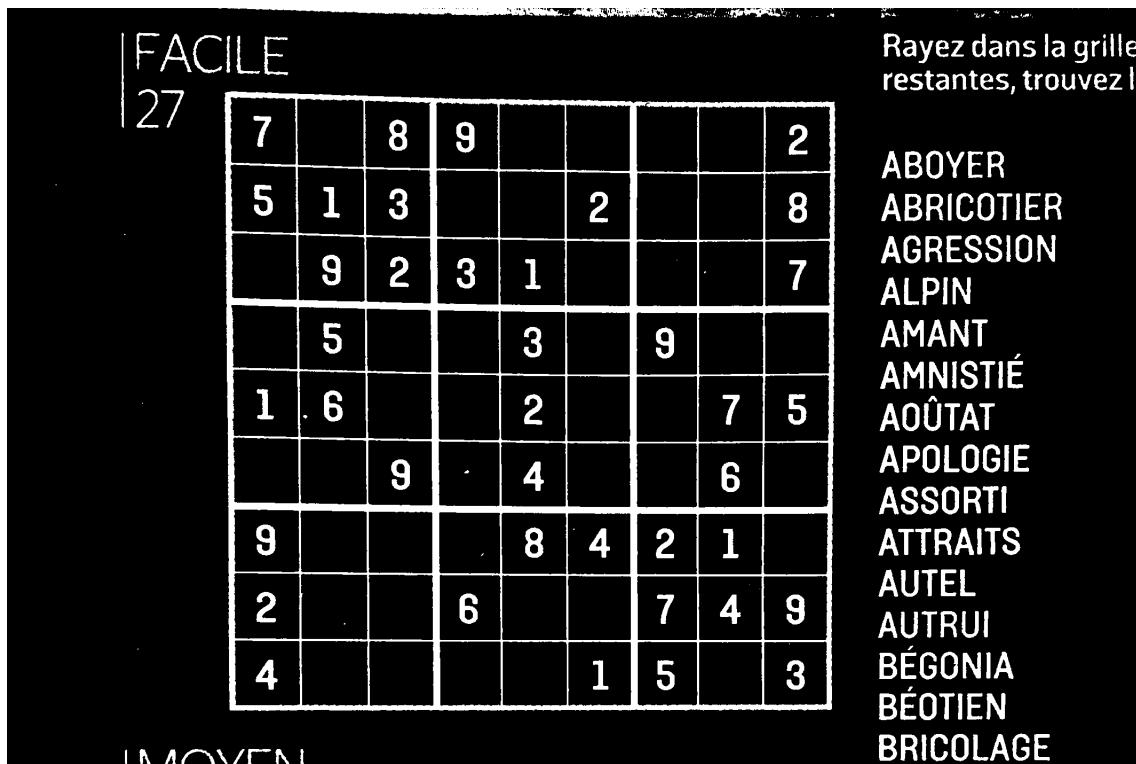


Figure 2.5 - Application du threshold adaptatif

Pour finir, afin de faire en sorte que l'image soit bien traité par la détection de ligne, je réinverse les couleurs.

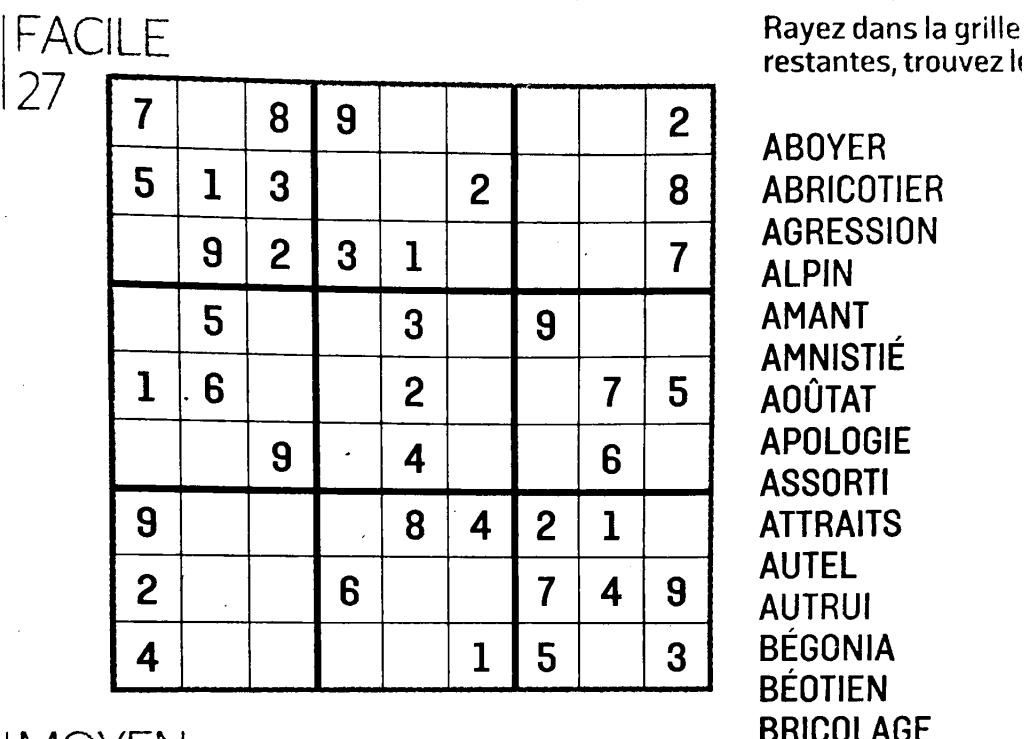


Figure 2.6 - Image finale

2.2 Rotation manuelle de l'image

En ce qui concerne la rotation manuelle de l'image, celle-ci s'est déroulée en plusieurs étapes que nous allons détailler.

Premièrement, nous avons du charger l'image sur laquelle nous voulions faire la rotation. Pour cela et pour le reste de la rotation nous avons utilisé la bibliothèque SDL. Grâce à notre angle de rotation choisi au préalable et l'image que nous venions de charger nous pouvions effectuer la rotation.

Nous avons commencé par prendre les mesures de l'image initiale afin de déterminer avec l'angle donné les dimensions de la nouvelle image et allouer la mémoire suffisante pour la surface de l'image qui reçoit la rotation. Ensuite, comme nous avons procédé par rotation centrale, il nous a fallu déterminer la position du centre de l'image. Nous avons alors parcouru l'image de départ, pixel par pixel. Pour chaque pixel, nous lui appliquons une formule qui prend comme paramètre le centre de l'image ainsi que l'angle et les coordonnées en x et en y du pixel. Cette formule détermine la nouvelle position du pixel que l'on place dans la nouvelle image. Pour finir, nous n'avions plus qu'à sauvegarder la nouvelle image. Pour cela nous avons utiliser la fonction BitMap de SDL.

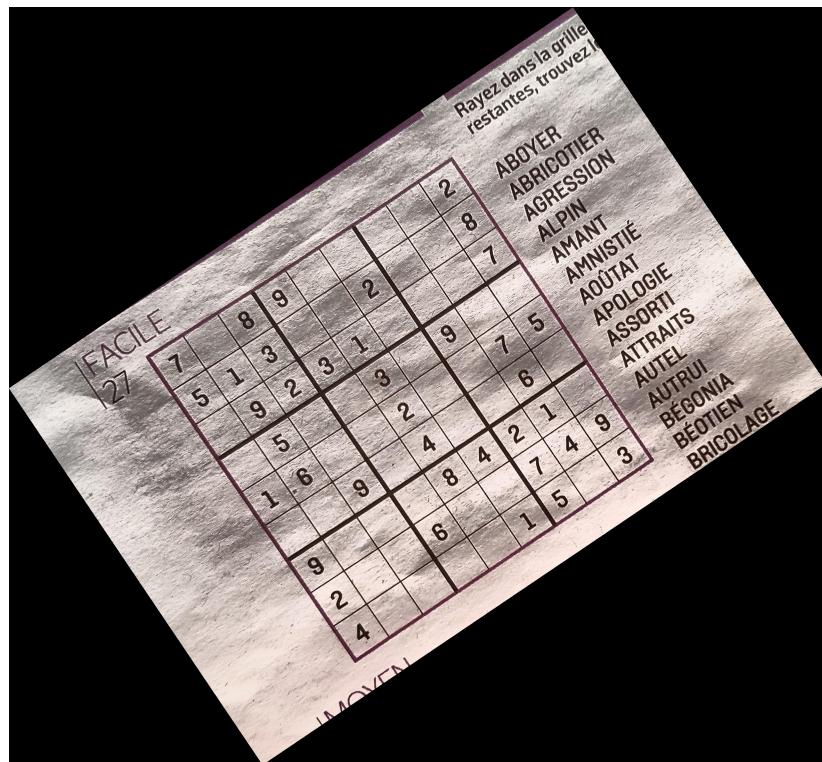


Image avant rotation

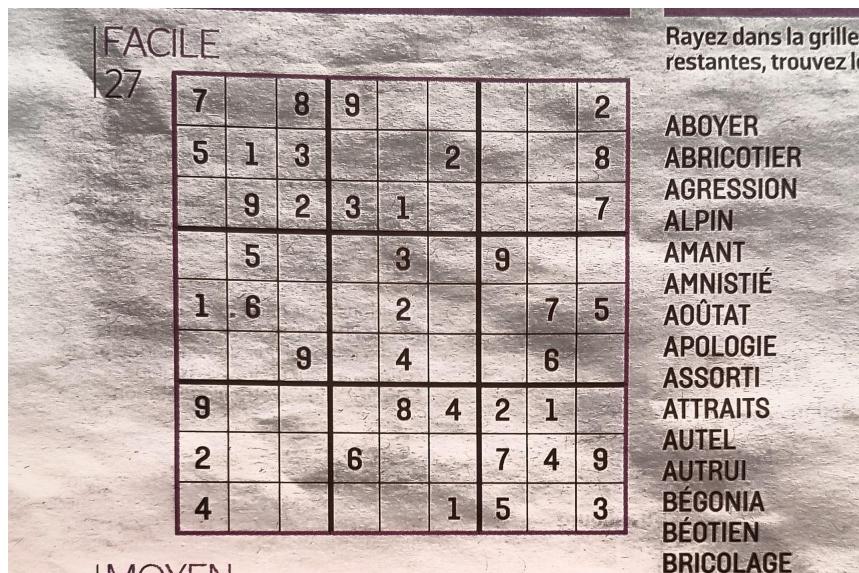


Image après rotation

Chapitre 3

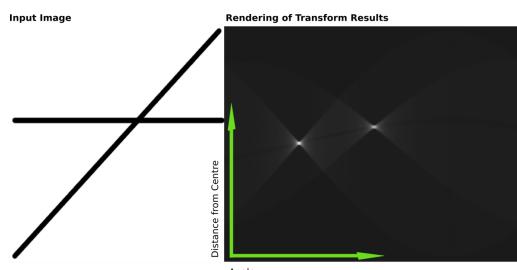
Segmentation

3.1 Détection de la grille

Pour cette partie nous allons devoir utiliser l'algorithme de Hough que l'on détaillera dans la première sous partie. Il nous permettra de détecter les lignes ainsi que de détecter les intersections pour pouvoir découper les cases que nous mettrons par la suite dans un dossier nommé au nom du fichier donné en argument.

3.1.1 Transformée de Hough

La première partie de la transformée de Hough était de trouver les pixels noirs grâce au module SDL, une fois les pixels détectés, nous ajouterons leur présence dans l'accumulateur qui est un tableau accessible grâce à un angle et un rayon qui seront définis par le x et l'y du pixel. Cet angle et ce rayon seront les paramètres d'une droite passant par le pixel. Une fois tous les x et les y de l'image parcourue nous obtiendrons notre accumulateur rempli de valeur qui nous dira combien de pixels sont parcouru par la droite.



Espace de Hough

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

Affichage de la détection de lignes

3.1.2 Traitement de l'accumulateur

Il fallait ensuite prendre une valeur de seuil qui pourra être remplacée dans le futur par un threshold qui permettra de changer cette valeur de seuil en fonction de l'image. En effet, comme il peut y avoir plus ou moins de pixels noirs suivant les images, il serait efficace pour la détection de ligne d'avoir un seuil adaptatif, comme dans le cas où l'on aurait des écritures sur le côté de la grille . Une fois cette valeur de seuil donné nous avions plein de lignes différentes pour une seule et même ligne.

3.2 Découpage de l'image

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

Affichage des intersections

Nous avions donc plein de lignes pour une seule et même ligne, il fallait donc traiter le nombre de ligne pour avoir 10 lignes et 10 colonnes à la fin et ensuite trouver les intersections entre ces lignes pour pouvoir découper les images entre 4 intersections.

3.2.1 Traitement des intersections

Pour traiter les intersections, nous commençâmes par détecter 10 colonnes et 10 lignes distinctes. Pour cela, à chaque fois que l'on tombe sur une ligne nous avons augmenté le x ou le y suivant si l'on traitait les colonnes ou les lignes, d'un certain nombre de pixels, ce qui nous a permis d'avoir les 10 colonnes et les 10 lignes. On peut les observer en vert et en rouge sur la photo. Ensuite pour toutes les lignes nous avons trouvé chaque intersection grâce à une formule mathématique et nous avons gardé les coordonnées des intersections dans un tableau à deux dimensions pour le x et le y.

3.2.2 Nouvelle surface

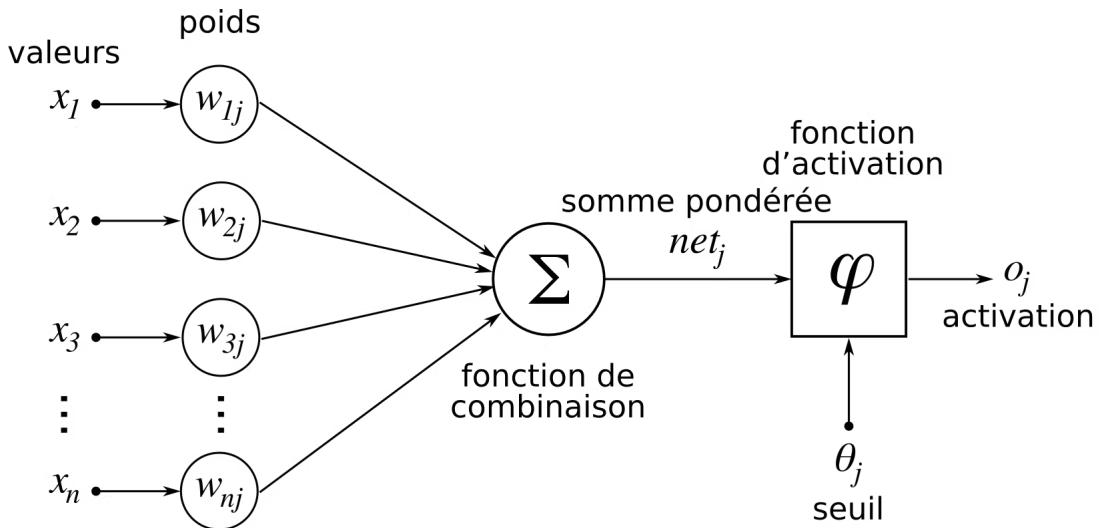
Pour chaque intersection nous avons pris le coin opposé du carré pour créer une nouvelle surface en copiant les pixels de l'ancienne surface à cette endroit. Ensuite nous les avons enregistré dans un nouveau dossier nommé list_nomdufichier avec les x et les y de l'intersection comme nom de chaque case, malheureusement le découpage n'est pas encore opérationnel sur toutes les images dû au différentes valeurs de seuil qui n'est pas fonctionnel sur toutes les images et qui le sera en vue de la prochaine soutenance.

Chapitre 4

Réseau de neurones (XOR)

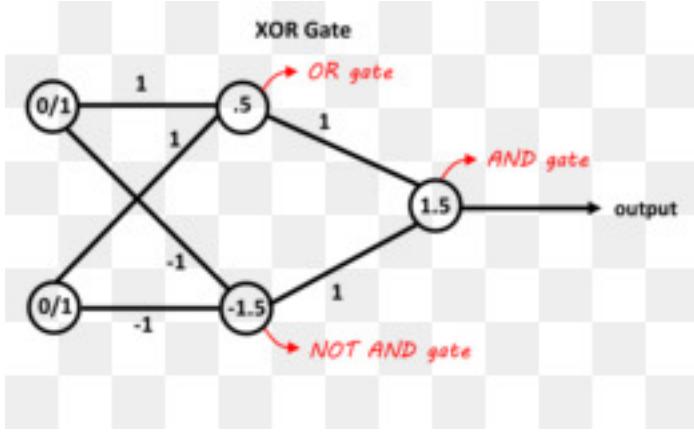
4.1 Qu'est ce qu'un réseau de neurones ?

Dans cette partie, je vais vous expliquer les différentes étapes pour créer un réseau de neurones. La première étape (la moins évidente) est de comprendre le concept assez complexe de réseau de neurones. Je vais donc définir le concept de réseau de neurones. Un réseau de neurones est un concept informatique permettant de résoudre divers problèmes (dans la plupart des cas des problèmes de reconnaissances de formes). Celui-ci possède une entrée, une sortie et plusieurs couches de neurones intermédiaires. Ces neurones intermédiaires sont reliés par une relation mathématique. La valeur du prochain neurone vaut la somme des précédents neurones fois le poids plus le biais. La sortie est modifiée grâce à une fonction mathématique, celle-ci est ramenée entre 1 et 0. La fonction qu'on utilise est la fonction Sigmoid (voir annexe). La partie la plus importante d'un réseau de neurones est la capacité d'apprentissage, celle-ci se fait en comparant la sortie avec une valeur de prédiction. En fonction de cet écart, on modifie les biais et les poids pour pouvoir nous rapprocher de la valeur de prédiction.(On utilise un algorithme de backpropagation qui sera expliqué par la suite.) Voici un schéma récapitulant le concept de réseau de neurones



4.1.1 Le concept de XOR

Il existe un cas assez simple du réseau de neurones, le problème XOR. Le but de cette implémentation est de mettre en entrée 2 valeurs par exemple 1 et 0 et d'avoir un résultat se rapprochant le plus possible de la bonne valeur. C'est donc un cas particulier d'un réseau de neurones, la seule différence est qu'on a que 2 entrées et une seule sortie.



4.1.2 L'implémentation du Xor

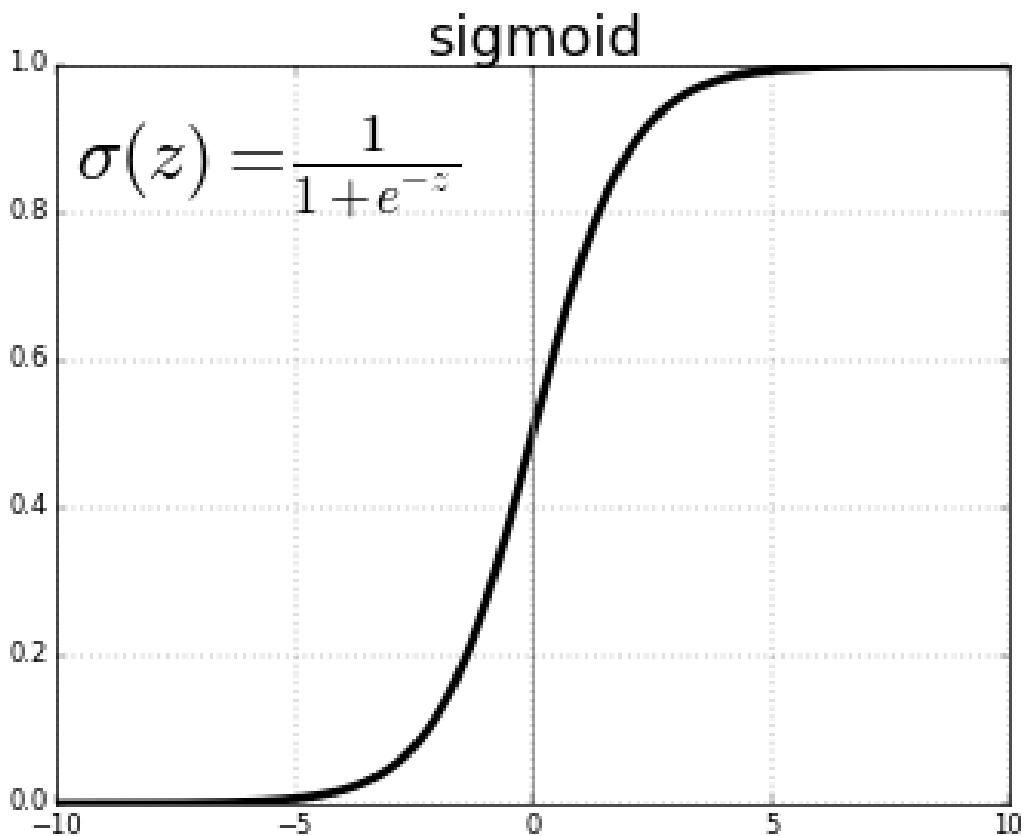
Dans cette partie, je vais vous décrire mon cheminement de pensée pour la création du XOR. Nous avons un réseau de neurones XOR dans lequel on met un input 01,11,10 ou 00 et on'a un processus d'apprentissage qui se lance (avec le résultat et la prédition qui sont print) et à la fin de ce processus la valeur la plus précise possible qui correspond à la valeur d'input s'affiche.

1) La mise en place du réseau de neurones

La première étape était de créer la couche cachée et la couche de sortie. Nous avons représenté ces couches par des tableaux à une dimension. On crée aussi deux variables qui vont avoir une importance, deux matrices, la première avec toutes les entrées possibles, la deuxième qui correspond aux sorties possibles. Il y a aussi une variable qui est indispensable (la variable lr). Qui correspond à la vitesse de calcul du réseau de neurones. La deuxième étape est de créer aléatoirement les biais et les poids des différentes couches, ces valeurs seront modifiés par l'apprentissage. Nous avons aussi créé une fonction dont le but est d'avoir un tableau avec les 4 valeurs (Chaque valeur correspond à une combinaison (01,11,10,00) et lorsque l'on lance notre apprentissage on mélange le tableau pour avoir un input aléatoire pour que notre apprentissage prenne en compte toutes les possibilités. La dernière étape est de décider du nombre d'itération pour que notre réseau de neurones soit le plus précis, nous avons décidé de prendre le nombre 100000.

2) le calcul de la prochaine valeur

Nous commençons donc par mélanger le tableau pour avoir notre entrée, puis nous calculons la valeur du prochain neurone. Si on est sur la couche d'entrée, on calcule celle de la cache cachée et si on est dans la couche cachée, on calcule celle de la couche de sortie. Pour cela, on dit que notre neurone est égale au biais auquel on ajoute la somme des poids fois les neurones de la couche précédente. Mais cette valeur, on doit la comparer, on fait donc Sigmoid de cette valeur pour avoir un chiffre entre 0 et 1. En résumé on dit donc que la couche de sortie et cachée correspond à sigmoid(opération mathématique exprimée plus haut).

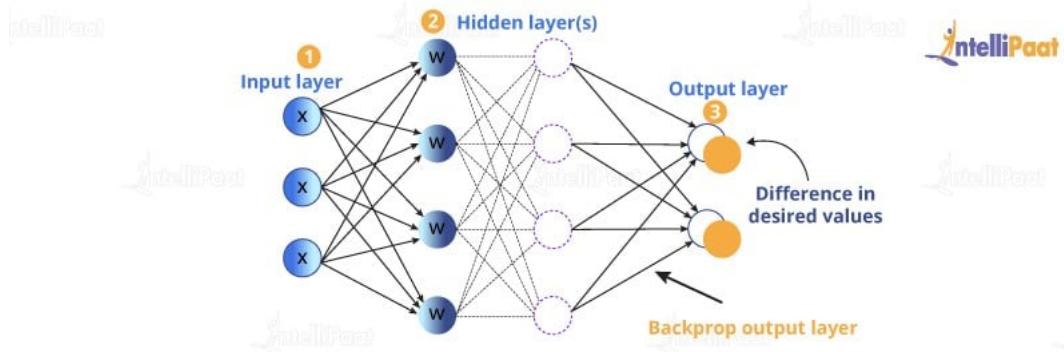


Ici on voit la fonction sigmoid, on observe bien que la fonction ramène la valeur entre 1 et 0.

3) L'algorithme de backpropagation

Cette partie était la partie la plus complexe, car elle était purement mathématique. Maintenant, qu'on a notre sortie, il faut comparer cette sortie avec notre prédiction et modifier les poids et biais en conséquence. Tout d'abord, il fallait

utiliser la fonction dsigmoid qui est la dérivée de la fonction sigmoid. La première étape était de dire que notre taux d'erreur correspondait à la différence entre notre prédiction et notre sortie. Il fallait ensuite multiplier notre taux par la fonction sigmoid pour avoir un nombre plus exploitable (deltasortie). Et il fallait remonter cet algorithme vers le début, c'est-à-dire que pour la couche cachée, on dit que le deltagache correspond au deltasortie calculé précédemment * les poids de la couche de sortie. La dernière étape était de calculer des nouveaux poids et biais pour pouvoir rendre le réseau de neurones plus précis. Pour cela, il fallait multiplier les différents deltas par la constante lr (qui correspond à la vitesse d'apprentissage) par la couche qui correspond. (La couche cachée pour les poids de la couche de sortie et les différentes entrées pour les poids de la couche cachés



4) Appliquer le réseau de neurones à un input

De plus, nous avons affiché les différentes valeurs à chaque itération d'apprentissage sous la forme. (Entrée : " , Sortie : " " , Prédiction :) Mais nous avons un problème, le premier problème est que lorsque nous effectuons l'apprentissage celui-ci ne dépend pas des valeurs d'inputs et recommence à 0 à chaque fois. J'ai donc modifié le script pour qu'on mette des valeurs d'input 01 , 10 , 11 ou 00 . Le réseau de neurones fait l'apprentissage puis print la valeur qui correspond à l'input donné .

5) Enregistrement des poids

Afin de conserver une trace du travail de notre réseau de neurones, nous avons enregistrer tous les poids que le réseau de neurones calcule pendant l'apprentissage. Pour réaliser cela, nous les avons écrit dans fichier texte dans lequel. Nous pourrons, en lisant le fichier texte nous en servir pour la soutenance finale.

```
1 Sauvegarde:'
2 ''
3 '0000 1.000000  Sortie : 0.736111  Prediction : 1.000000'
4 ''
5 'Entrée : 0.000000 0.000000  Sortie : 0.706245  Prediction : 0.000000'
6 ''
7 'Entrée : 1.000000 1.000000  Sortie : 0.745149  Prediction : 0.000000'
8 ''
9 'Entrée : 1.000000 0.000000  Sortie : 0.713517  Prediction : 1.000000'
10 ''
11 'Entrée : 0.000000 0.000000  Sortie : 0.696548  Prediction : 0.000000'
12 ''
13 'Entrée : 1.000000 1.000000  Sortie : 0.734712  Prediction : 0.000000'
14 ''
15 'Entrée : 1.000000 0.000000  Sortie : 0.703007  Prediction : 1.000000'
16 ''
17 'Entrée : 0.000000 1.000000  Sortie : 0.718049  Prediction : 1.000000'
18 ''
19 'Entrée : 0.000000 1.000000  Sortie : 0.720725  Prediction : 1.000000'
20 ''
21 'Entrée : 1.000000 0.000000  Sortie : 0.711155  Prediction : 1.000000'
22 ''
23 'Entrée : 0.000000 0.000000  Sortie : 0.694308  Prediction : 0.000000'
24 ''
25 'Entrée : 1.000000 1.000000  Sortie : 0.732331  Prediction : 0.000000'
26 ''
27 'Entrée : 0.000000 1.000000  Sortie : 0.712812  Prediction : 1.000000'
28 ''
29 'Entrée : 0.000000 0.000000  Sortie : 0.684336  Prediction : 0.000000'
30 ''
31 'Entrée : 1.000000 0.000000  Sortie : 0.697084  Prediction : 1.000000'
32 ''
33 'Entrée : 1.000000 1.000000  Sortie : 0.724566  Prediction : 0.000000'
34 ''
35 'Entrée : 0.000000 1.000000  Sortie : 0.705041  Prediction : 1.000000'
36 ''
37 'Entrée : 0.000000 0.000000  Sortie : 0.677092  Prediction : 0.000000'
38 ''
39 'Entrée : 1.000000 1.000000  Sortie : 0.713637  Prediction : 0.000000'
40 ''
41 'Entrée : 1.000000 0.000000  Sortie : 0.681908  Prediction : 1.000000'
42 ''
43 'Entrée : 0.000000 1.000000  Sortie : 0.697446  Prediction : 1.000000'
44 ''
45 'Entrée : 1.000000 1.000000  Sortie : 0.712508  Prediction : 0.000000'
46 ''
47 'Entrée : 0.000000 0.000000  Sortie : 0.663051  Prediction : 0.000000'
48 ''
49 'Entrée : 1.000000 0.000000  Sortie : 0.674291  Prediction : 1.000000'
50 ''
51 'Entrée : 0.000000 1.000000  Sortie : 0.689983  Prediction : 1.000000'
52 ''
53 'Entrée : 1.000000 0.000000  Sortie : 0.681016  Prediction : 1.000000'
54 ''
```

Fichiers enregistrement des poids

Chapitre 5

Résolution du sudoku

5.1 Partie 1

5.1.1 Récupération puis écriture du sudoku dans un fichier

Dans le but de résoudre le sudoku il est d'abord nécessaire de le récupérer depuis le fichier où il est enregistré, puis ensuite une fois résolut il faut l'écrire dans un nouveau fichier en lui ajoutant l'extension ".result".

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

Fichier à récupéré

```
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
```

Ficher crée en sortie

5.1.2 Résolution du sudoku

Pour résoudre un sudoku il faut tester de nombreuses possibilités différentes, car pour remplir une case vide il faut essayer un chiffre puis ensuite si ce n'était pas le bon, revenir en arrière. Pour cela il faut faire un programme en backtracking. Ce programme permettra d'explorer toutes les possibilités et si le sudoku est solvable il le renverra. Pour faire cela le programme va chercher une case vide dans le sudoku la remplir et continuer jusqu'à ce que le programme ne puisse plus rien trouver à mettre dans une case. C'est-à-dire que la façon donc il a rempli la grille de sudoku est fausse. Il va donc retourner en arrière pour chercher à remplir la grille autrement, c'est le principe du backtracking. Quand il n'y a plus de case vide c'est que le sudoku est résolut.

Chapitre 6

Conclusion

6.1 Attentes et avancement

Voici un tableau (cf. fig. 6.1) récapitulatif de notre analyse de l'existant...

	Avancement 1er rendu	Objectif 1er rendu	Objectif rendu finale
Prétraitement de l'image	90%	90%	100%
Rotation de l'image	40%	40%	100%
Détection de ligne	100%	100%	100%
Découpage des cases	60%	60%	100%
Réseau de Neurones XOR	100%	100%	100%
Réseau de Neurones final	0%	0%	100%
Enregistrement des poids	100%	100%	100%
Résolution du Sudoku	100%	100%	100%
Interface	0%	0%	100%

FIGURE 6.1 – Tableau récapitulatif des solutions

6.2 Conclusion

Au cours de cette première période de projet, nous avons compris l'objectif de celui-ci, nous l'avons divisé en des sous-parties majeures que nous nous sommes répartis avec plus ou moins de difficultés. Malgré les accrocs que nous avons pu rencontrer sur le projet nous avons tous réussis à apporter nos connaissances et notre réflexion sur ses différentes parties et à aider les autres si besoin afin de mener à bien ce projet.

Nous sommes conscient qu'il nous reste du travail afin de finaliser ce projet mais nous restons positif quant à son bon déroulement.

Nous sommes impatients de voir le résultat final ce qui nous motive pour

travailler encore plus dur sur ce projet maintenant que nous sommes à plongés à coeur perdu dans celui-ci.

Annexes

Annexe

1 Sources

Voici les différents liens de documentation utilisés :

<http://neuralnetworksanddeeplearning.com>

<https://fr.wikipedia.org/wiki/R>

<https://greaby.co/les-reseaux-de-neurones-apprentissage-supervise-dans-godot-engine/>

<https://www.ibm.com/fr-fr/cloud/learn/neural-networks>

<https://medium.com/analytics-vidhya/coding-a-neural-network-for-xor-logic-classifier-from-scratch-b90543648e8a>

<https://towardsdatascience.com/implementing-the-xor-gate-using-backpropagation-in-neural-networks-c1f255b4f20d>

-Pyimagesearch, (2021), Adaptive Thresholding, Retrouvé le 25 Octobre 2022,

<https://pyimagesearch.com/2021/05/12/adaptive-thresholding-with-opencv-cv2-adaptivethreshold/>

-Docs.Opencv, (n.d), Py Thresholding, Retrouvé le 25 Octobre 2022, <https://docs.opencv.org>

-IEEEExplore, (2015), Retrouvé le 05 Novembre 2022, <https://ieeexplore.ieee.org/docum>