



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	3
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 Applications de différents filtres	8
2.1.4 Threshold adaptatif	9
2.2 Rotation manuelle de l'image	10
2.3 Rotation automatique de l'image	12
3 Segmentation	14
3.1 Détection de la grille	14
3.1.1 Transformée de Hough	14
3.1.2 Traitement de l'accumulateur	15
3.2 Découpage de l'image	15
3.2.1 Traitement des intersections	16
3.2.2 Nouvelle surface	16
3.2.3 Réduction de la taille de l'image	17
4 Réseau de neurones	18
4.0.1 Dataset	20
4.0.2 Network architecture	20
4.0.3 Training	21
4.0.4 Validation	21
5 Résolution du sudoku	22
5.1 Partie 1	22
5.1.1 Récupération puis écriture du sudoku dans un fichier	22
5.1.2 Résolution du sudoku	22
6 Interface	23
6.1 Section partie interface	23
6.1.1 creation de l'application	23
6.1.2 mise en relation des différent programme	24
7 Website	25
8 Conclusion	26
8.1 Attentes et avancement	26
8.2 Problèmes rencontrés et conclusions personnelle	26

8.2.1	JESSY	26
8.2.2	MATHIS	27
8.2.3	SATYA	27
8.2.4	MATHIEU	27
8.2.5	ENZO	28
8.3	Conclusion	28
	Annexes	30
	Annexe 1	30
1	Sources	30

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 la distribution des tâches

	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 de la répartition de tâches

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.

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

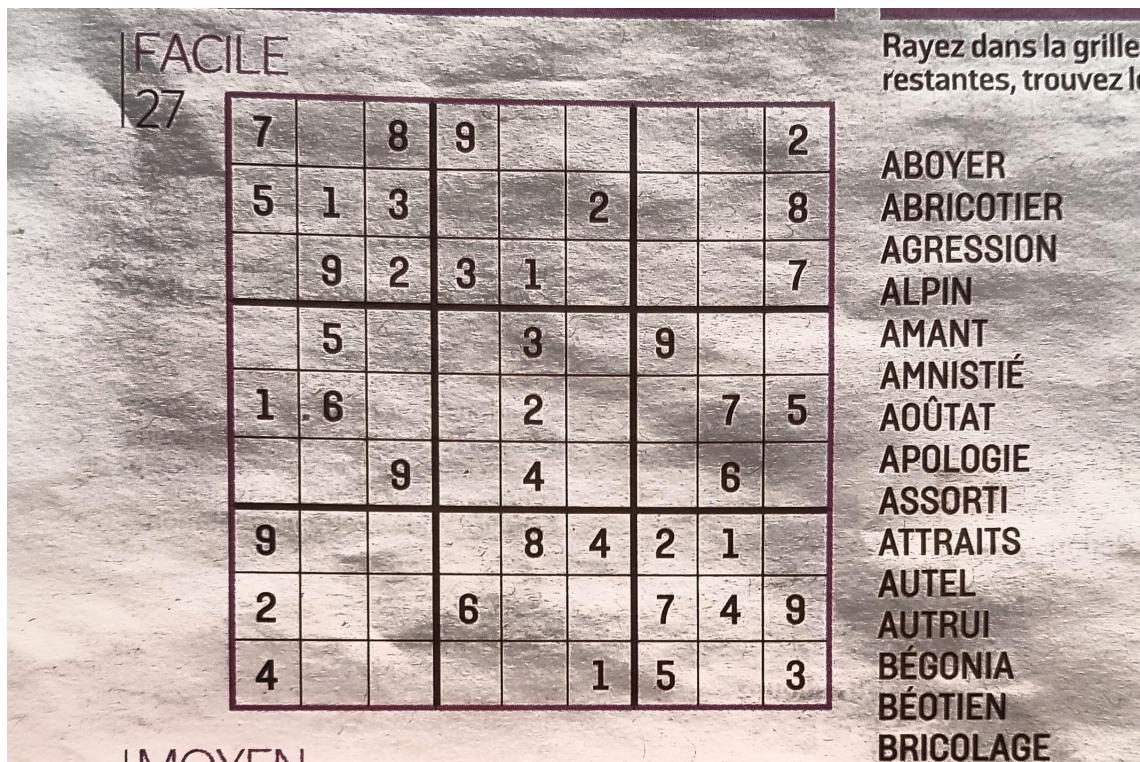


Figure 2.1.1 - Image de base

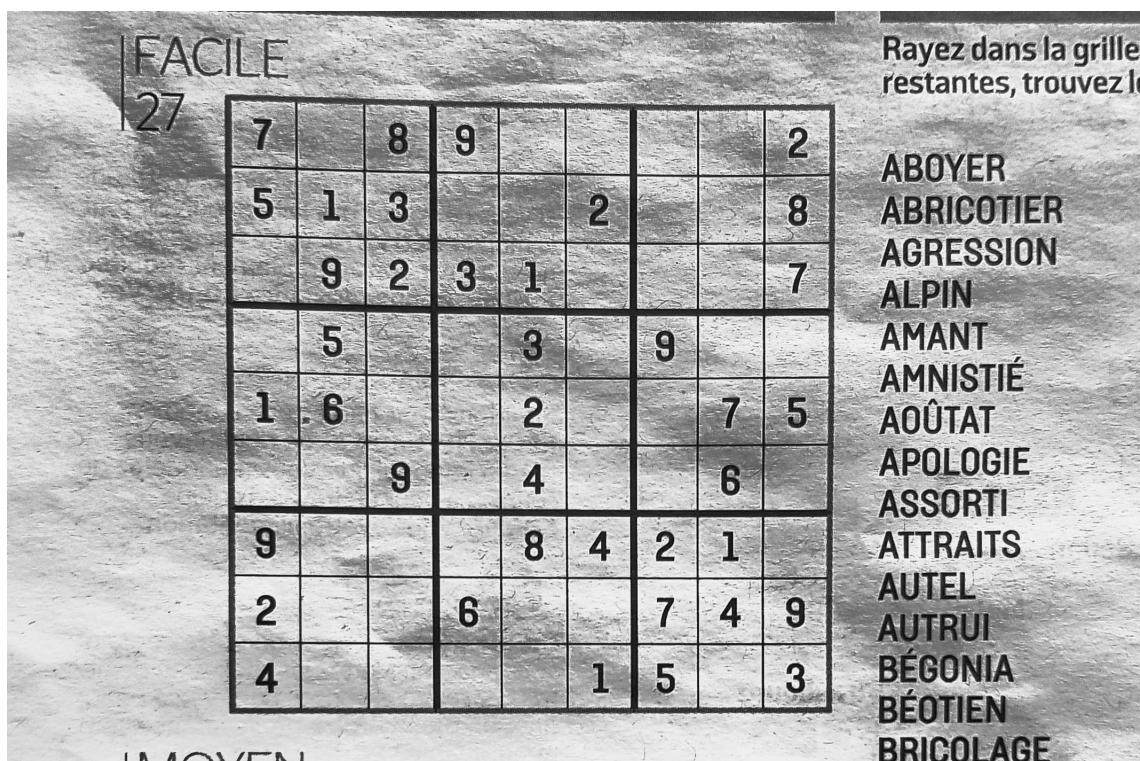


Figure 2.1.2 - Application du grayscale

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 :

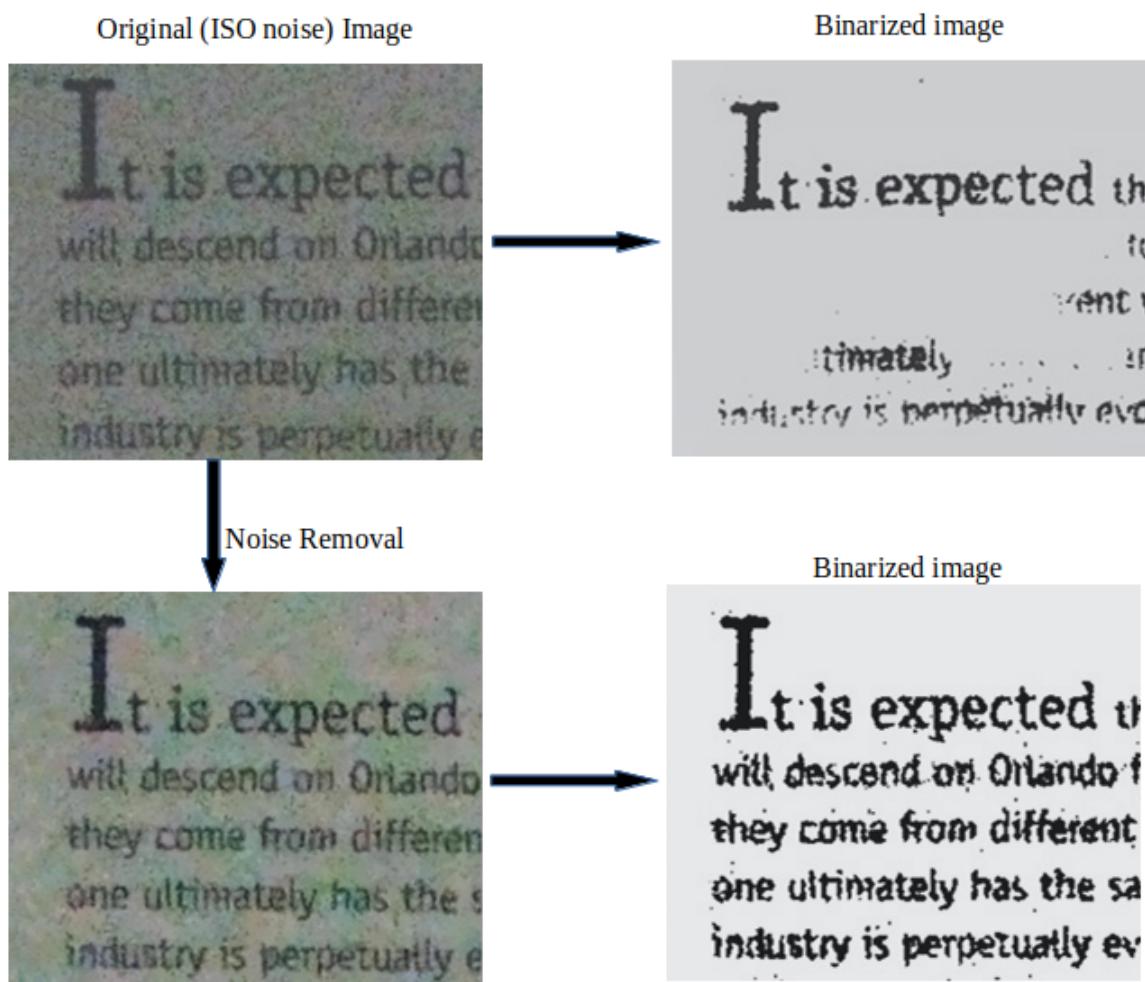


Figure 2.1.3 - Comparaison d'une image binarisée avec et sans suppression de bruit

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

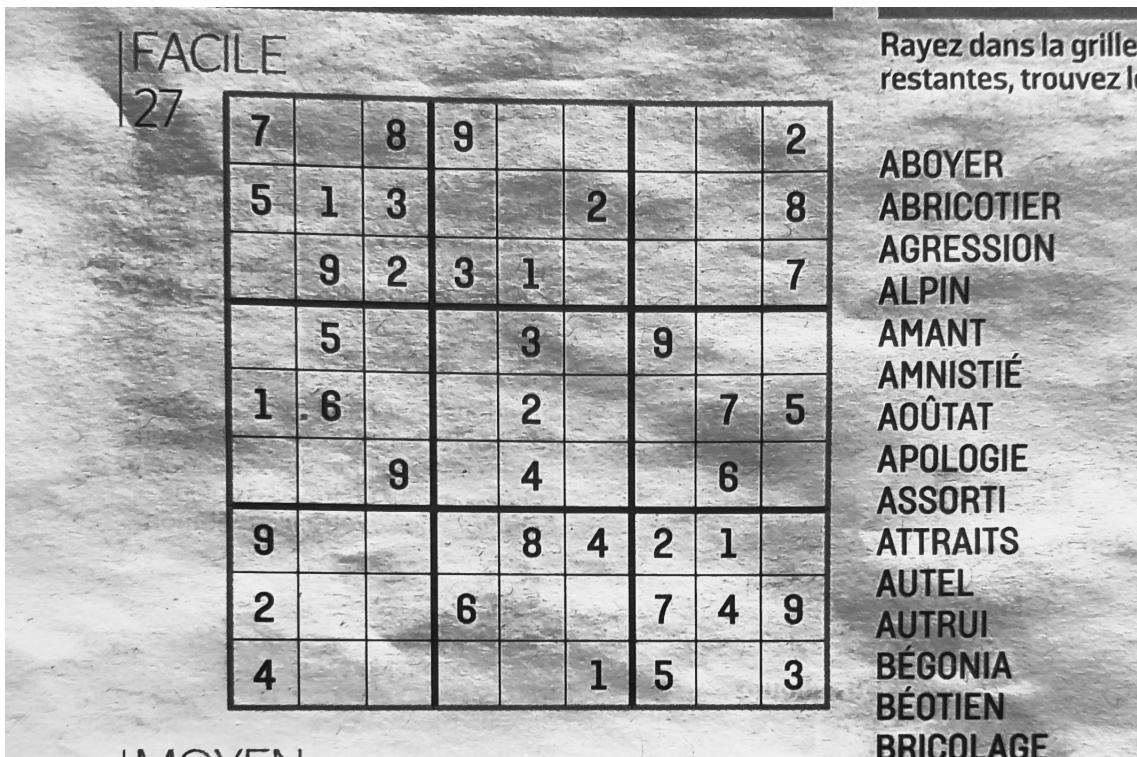


Figure 2.4 - Suppression du bruit

2.1.3 Applications de différents filtres

On a également décidé de développer et d'appliquer d'autres filtres afin d'avoir l'image la plus proposée possible. Ainsi, nous avons reproduit le filtre gaussien, qui a pour objectif de lisser l'image et de réduire le bruit. Ce filtre gaussien fonctionne en appliquant un masque de convolution à l'image, en utilisant une matrice de coefficients générés à partir d'une fonction gaussienne. La fonction gaussienne est une fonction mathématique qui décrit une courbe en forme de cloche, qui s'élargit de manière symétrique autour de sa moyenne. La largeur de la courbe détermine la quantité de flou appliquée à l'image.

Nous avons également développé la détection de bordures de Sobel. Cette algorithme utilise des filtres de convolution pour calculer l'intensité des bordures (verticales et horizontales), puis combine ces informations pour trouver les bordures les plus prononcées dans l'image.

2.1.4 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 éviter d'avoir ce problème, nous avons décidé d'implémenter l'algorithme d'Otsu pour calculer la valeur de seuil. Cet algorithme calcule la variance interclasse des niveaux de gris des pixels de l'image pour différentes valeurs de seuil, et choisit le seuil qui minimise cette variance. La variance interclasse est une mesure de différence de niveaux de gris entre les deux groupes de pixels formés par le seuil. Plus la variance est faible, plus les deux groupes de pixels ont des niveaux de gris similaires, ce qui signifie que le seuil choisi est optimal pour séparer les deux groupes de pixels. Pour expliquer plus simplement, l'algorithme d'Otsu utilise la variance interclasse pour choisir le seuil optimal pour séparer les pixels d'une image en noir et blanc, en minimisant la différence de niveaux de gris entre les deux groupes de pixels formés par le seuil.

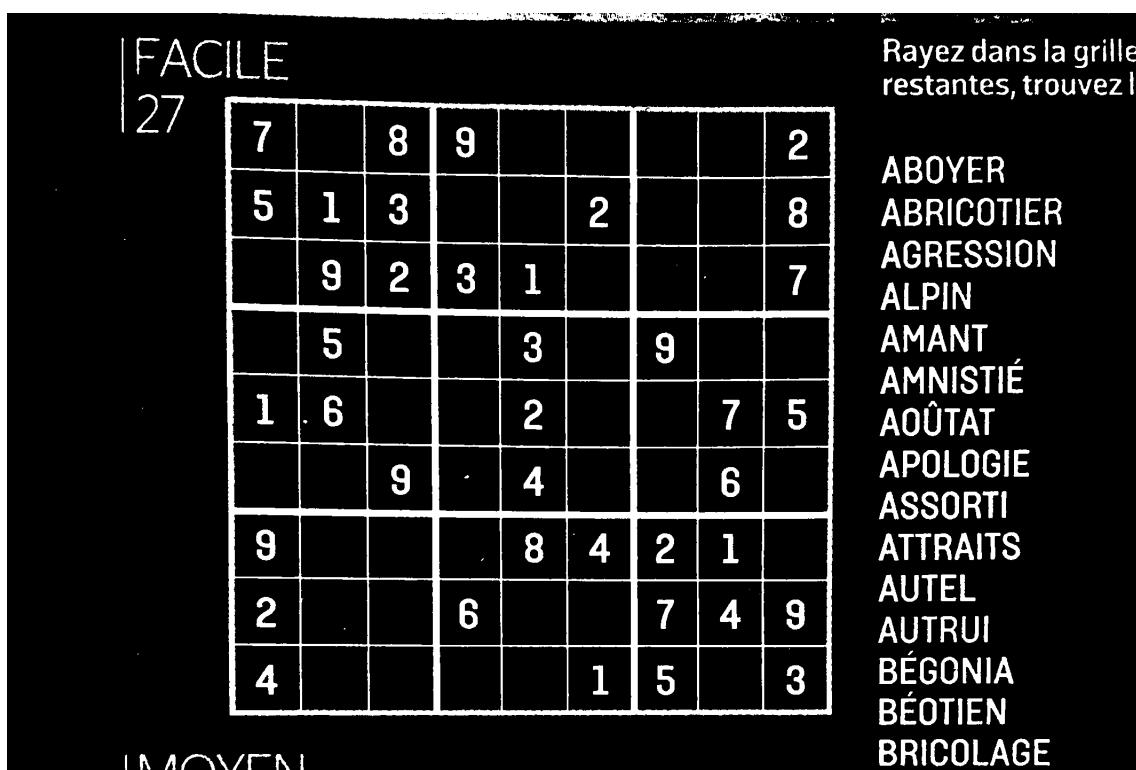


Figure 2.5 - Application du threshold adaptatif

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

FACILE

27

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

INNOVATION

Rayez dans la grille restantes, trouvez le

ABOYER
ABRICOTIER
AGGRESSION
ALPIN
AMANT
AMNISTIÉ
AOÛTAT
APOLOGIE
ASSORTI
ATTRATS
AUTEL
AUTRUI
BÉGONIA
BÉOTIEN
BRICOLAGE

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 utilisé la fonction BitMap de SDL.

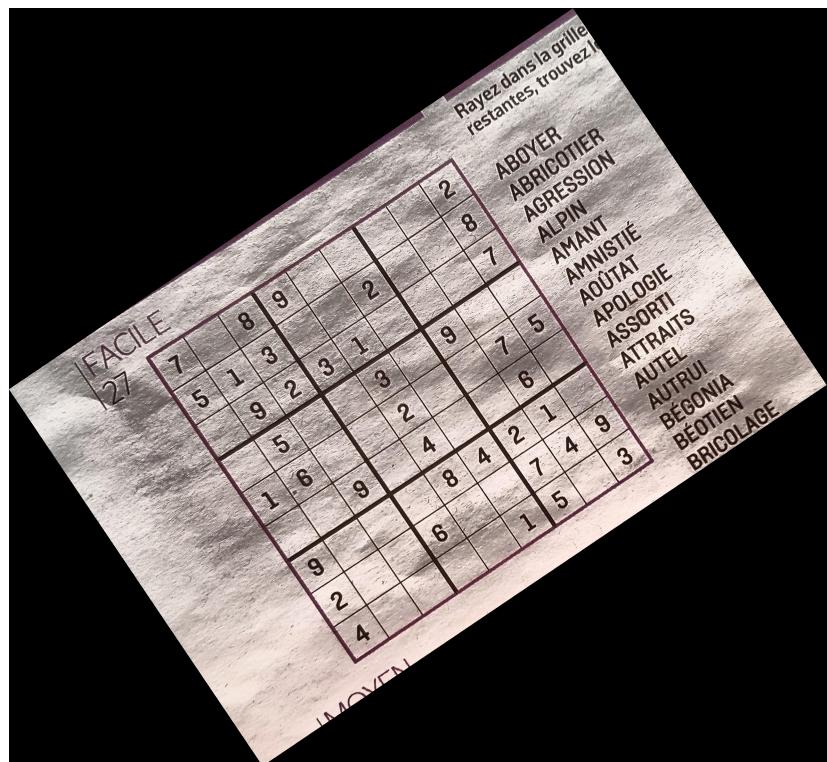


Image avant rotation manuelle

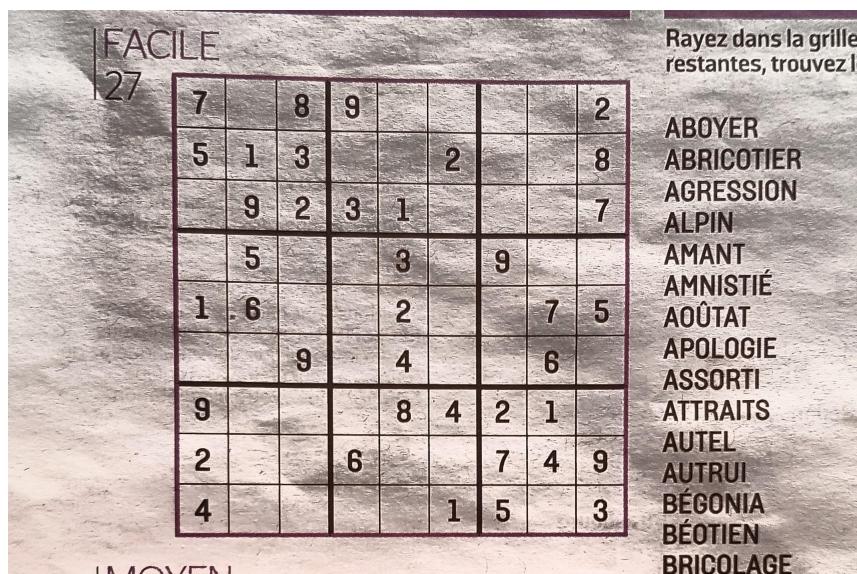


Image après rotation manuelle

2.3 Rotation automatique de l'image

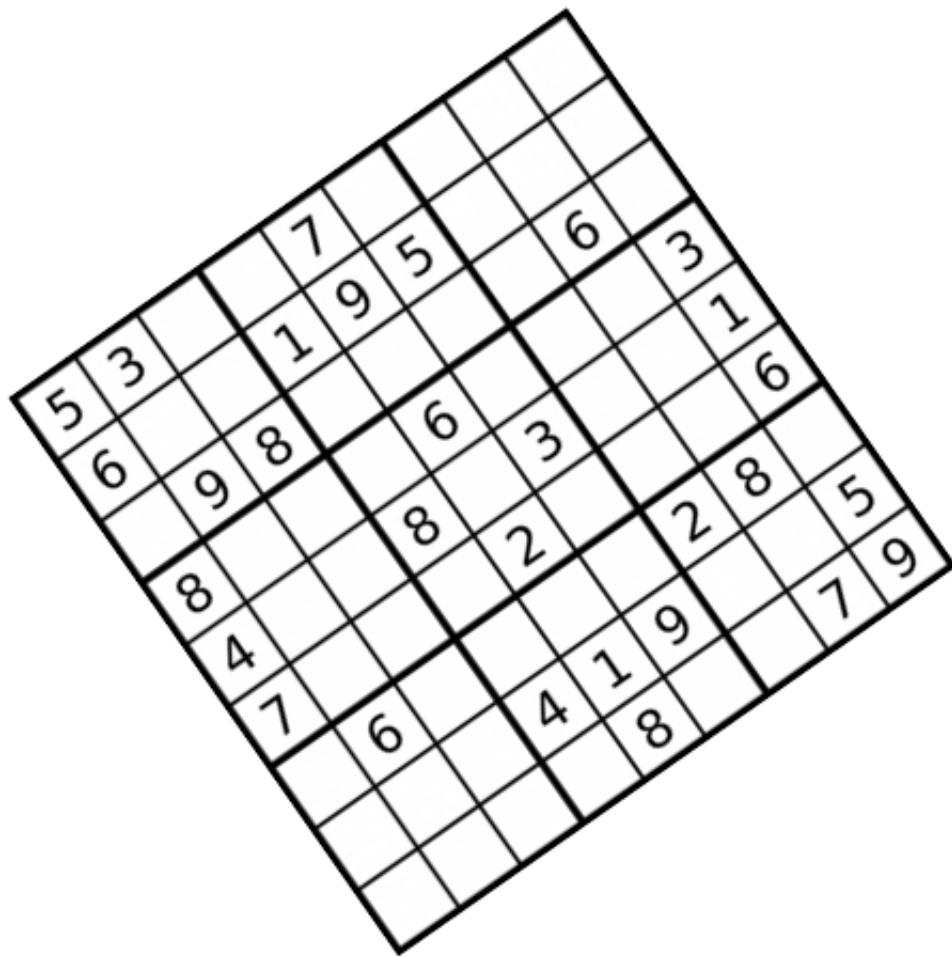


Image avant rotation automatique

Pour la rotation nous avons fait une rotation semi-automatique. En effet, nous utilisons la rotation automatique pour redresser l'image. Ensuite, nous utilisons la rotation manuelle pour mettre l'image dans le bon sens de lecture.

Ainsi, pour faire la rotation automatique nous avons du définir l'angle de rotation. Pour cela, nous commençons par définir les coordonnées de la surface.

De plus, nous avons parcouru l'image pixel par pixel pour déterminer les coordonnées des 4 coins de l'image initial qui est dans la surface. On calcule ensuite l'angle entre les lignes qui relient les différents coins de la surface et ceux de l'image tournée. La moyenne de ces quatres angles nous donne l'angle avec laquelle nous pouvons faire la rotation. On crée une nouvelle surface blanche avec les tailles nécessaires que nous avons défini à partir de l'angle de rotation et de la taille de la

surface initiale. Ensuite nous changeons pour chaque pixel de l'image ses coordonnées avec l'angle de rotation. Puis on copie l'image avec les pixels réordonnés dans la nouvelle surface.

Problème sur rotation automatique : Nous pouvons faire la rotation que pour image avec une surface (en fond) qui est blanche et des contours de grille noir.

Chapitre 3

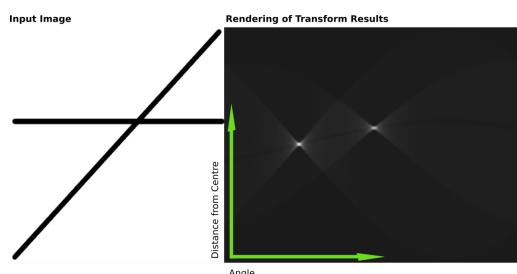
Segmentation

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 et 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 valeurs qui nous dira combien de pixels sont parcourus par la droite.



Espace de Hough

3.1.2 Traitement de l'accumulateur

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

Afin d'obtenir les lignes, on parcourt l'accumulateur et on recueille seulement l'angle et le rayon pour lesquels la valeur associée dans l'accumulateur donc le nombre de pixel noir que traverse la ligne est supérieur à la valeur de seuil. Il fallait ensuite prendre une valeur de seuil qui sera adapté à l'image et pour cela nous utilisons la hauteur de l'image fournie. Une fois cette valeur de seuil donné nous avions plein de lignes différentes pour une seule et même ligne.

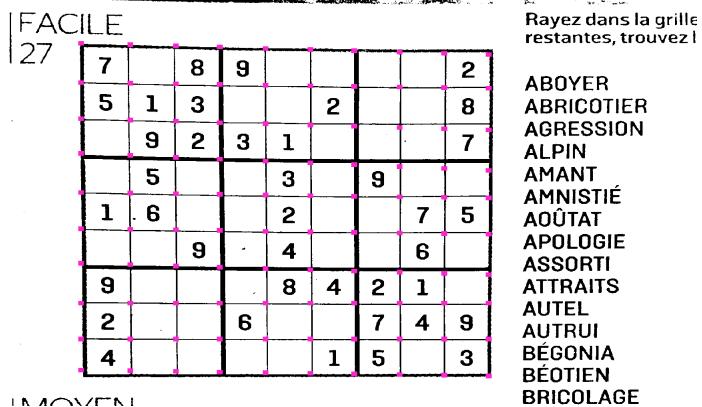
3.2 Découpage de l'image

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 10 lignes et 10 colonnes

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

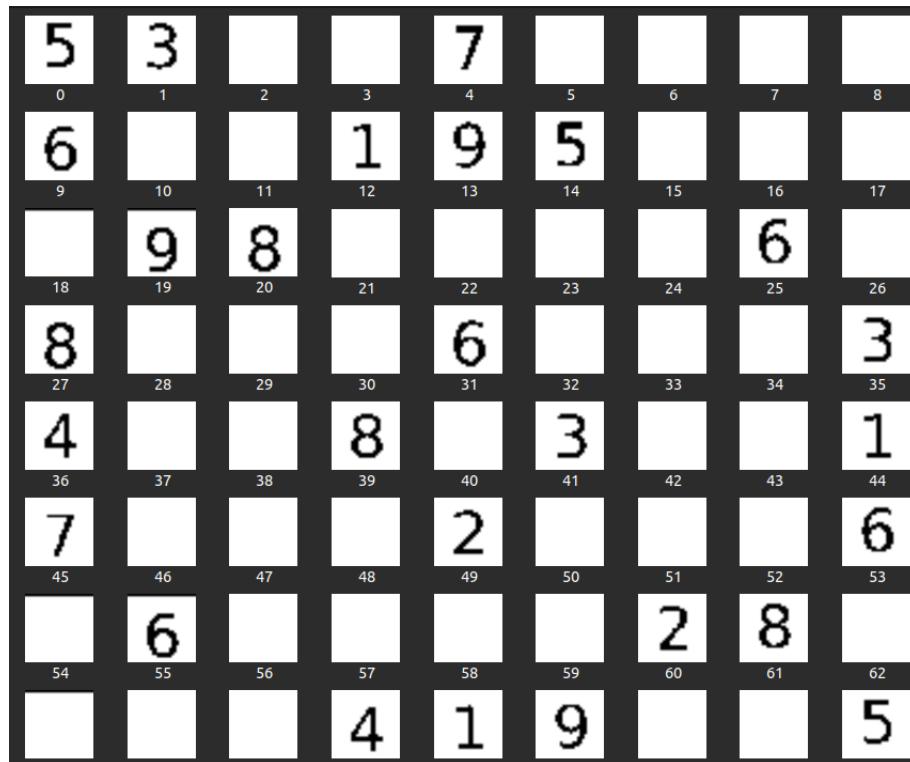


Affichage des interctions

Pour traiter les intersections, nous commençâmes par détecter 10 colonnes et 10 lignes distinctes. Pour cela, a 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 as 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 intersections grace à une formule mathématique et nous avons gardé les coordonées des intersections dans un tableau a deux dimensions pour le x et le y.

3.2.2 Nouvelle surface

Pour chaque intersections nous avons pris le coin opposé du carré pour créer une nouvelle surface en copiant les pixels de l'ancienne surface à cette endroit. Contrairement à l'avancement lors de la première soutenance maintenant aucune ligne n'apparrait dans la case enregistrée, il n'y a que le chiffre. Ensuite nous les avons enregistré dans un nouveau dossier nommé `case_list_nomdufichier` et avec comme nom le numéros de la case. Le dossier `case` est quand a lui créer a chaque compilation du fichier détection grace a la commande `mkdir` et le flag `-p` qui permet de créer le dossier s'il n'existe pas mais dans le cas ou il existe il ne fait rien sans ce flag la commande `mkdir` enverrait une erreur. Grace à cette archicecture de dossier nous avons pu inclure dans le Makefile la suppression du dossier mère "`case`" qui contient les dossiers contenant les cases des différents images.



Résultat pour l'image vue précédemment

3.2.3 Réduction de la taille de l'image

Les images sorties lors de la première soutenance faisaient la taille de la case et celle-ci pouvait renvoyer autre chose que des carrés et varier suivant la case. Il a donc fallu toujours couper les cases de manière à avoir des carrées puis les réduire en 28 fois 28 pour les réseaux de neurones. Plusieurs possibilités s'offraient alors à nous mais nous avons utilisé l'algorithme appelé "Nearest Neighbors" qui fonctionne en utilisant l'information de couleur des pixels pour déterminer les pixels à supprimer ou à conserver dans l'image. L'algorithme parcourt l'image pixel par pixel et compare la couleur du pixel courant avec celle de ses voisins les plus proches. Si la couleur du pixel courant est suffisamment proche de celle de ses voisins, alors il est supprimé, ce qui permet de réduire la taille de l'image tout en conservant la qualité de l'image originale.

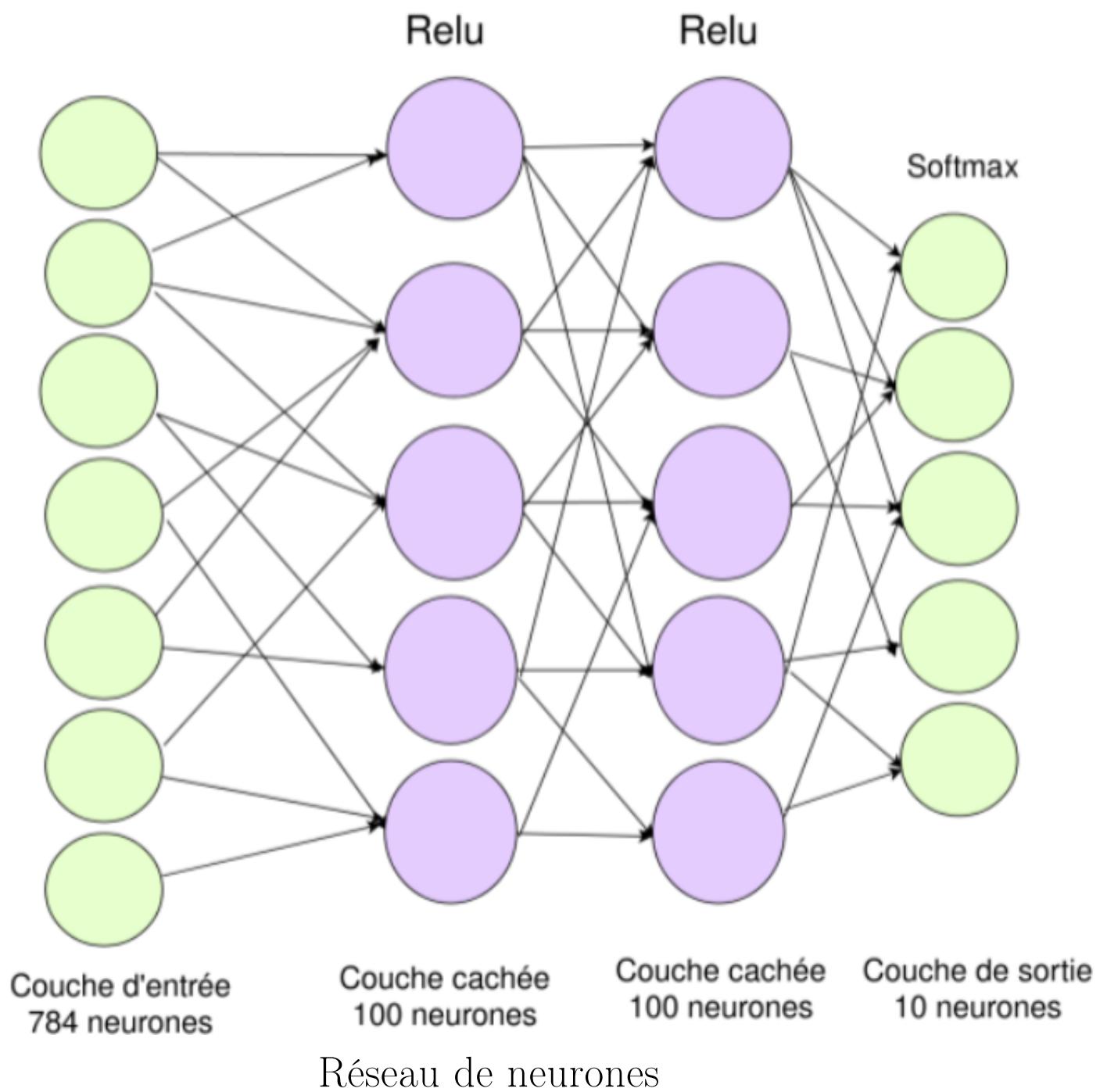
Chapitre 4

Réseau de neurones

Au moment de la première soutenance, ma partie était relativement primitive. D'abord, ma base de données contenait des images en nuances de gris ainsi que des zéros, ce qui est inutile dans le cas présent puisque nous traitons des images de sudoku en noir et blanc.

Ensuite, l'architecture de mon réseau de neurones était extrêmement basique, puisqu'il n'y avait pas de couche de neurones cachés. La couche d'entrée était directement reliée à la couche de sortie, ce qui limitait bien évidemment les capacités d'apprentissage de mon intelligence artificielle. Pour ne pas arranger les choses, il se trouve que l'entraînement de mon réseau de neurones n'était pas correct, l'entraînement n'était pas assez long et je ne passais pas en revue toutes les images qui composaient ma base de données. Certaines formules utiles à la backpropagation n'étaient également pas justes, ou du moins pas exactes.

C'est donc tout naturellement que j'ai modifié la base de données avec laquelle je travaillais, amélioré grandement l'architecture de mon réseau de neurones et revue intégralement ma méthode d'entraînement. Bien évidemment, toutes ces spécificités techniques seront précisées par la suite.



4.0.1 Dataset

Pour l'entraînement de mon réseau de neurones et la validation de ses capacités, j'utilise la célèbre base de données MNIST qui regroupe des chiffres écrits à la main et numérisés. Cette base de données contient 60 000 images d'entraînement et 10 000 images de validation. Bien évidemment, chaque image est représentée sous forme de matrice et associée ‘a son label respectif, à savoir le chiffre auquel l'image est supposée correspondre. Lorsque je charge ces images, je prends soin de les convertir en noir et blanc et de retirer tous les zéros, puisque ceux-ci n'apparaîtront jamais dans un sudoku. Chaque image étant de dimension 28x28 pixels, nous en déduisons le point d'entrée de l'architecture de notre réseau de neurones.

4.0.2 Network architecture

Comme mentionné ci-dessus, les images de notre base de données sont de taille 28x28 pixels. La couche d'entrée du réseau de neurones est donc composée de 784 noeuds représentant chacun un pixel de l'image ($28 \times 28 = 784$). Le réseau de neurones est ensuite composé de deux couches cachées comportant chacune 100 noeuds. C'est une taille conséquente qui permet d'atteindre une très bonne précision quant ‘à la reconnaissance des caractères, bien qu'elle provoque quelques ralentissements lors de la phase d'entraînement. Le choix de mettre deux couches cachées plutôt qu'une seule ou 100 noeuds plutôt qu'autre chose aurait pu être discuté, mais je préférerais être sûr que mon intelligence artificielle soit performante, sans quoi notre logiciel ne pouvait pas fonctionner correctement. Enfin, le réseau de neurones contient une inévitable couche de sortie. Celle-ci est composée de 10 noeuds : le noeud 0 représentants une image vide et les noeuds 1 ‘a 9 représentants les chiffres respectifs. Permettre à mon réseau de neurones de distinguer une image vide d'une autre image a permis de simplifier grandement la transition entre le découpage de la grille et l'identification des chiffres qui composent chaque case. Mon réseau de neurones adopte donc la forme 784-100-100-10. Les deux couches cachées sont activées avec la fonction RELU (Rectified Linear Unit) et la couche de sortie est activée avec la fonction Soft-max, qui est très adaptée aux problèmes de classification puisqu'elle détermine des probabilités d'activation pour chaque neurone de sortie. Ce réseau de neurones est donc solide, bien bâti, mais pour l'instant très stupide. Nous allons alors entamer l'étape la plus importante : la phase d'apprentissage.

4.0.3 Training

Pour l'entraînement de mon réseau de neurones, j'utilise la méthode de la descente de gradient (SGD) combinée à un learning rate de 0,01. D'abord, je commence par initialiser les poids et les biais du réseau de manière aléatoire et selon la méthode de He. Cette méthode permet d'échelonner les valeurs aléatoires en fonction de la taille du réseau, dans le but d'accélérer l'apprentissage en faisant converger les gradients plus rapidement vers le minimum. Ensuite, je mélange de manière aléatoire ma base de données de 60 000 images afin que mon réseau de neurones s'entraîne dans un ordre différent à chaque epoch d'apprentissage. Enfin, après de nombreux équilibrages, je parcours finalement ma base de données de 60 000 images grâce à des batchs de seulement 10 images. Je n'applique donc pas de descente de gradient après chaque image, mais toutes les 10 images et selon une moyenne relative à celles-ci. Au total, je répète cet entraînement sur 10 epochs et réduis ma fonction de coût (entropie croisée) à une valeur qui varie entre 0,01 et 0,05. Cette valeur n'étant pas très représentative, je soumets mon intelligence artificielle à un test dans le but d'évaluer ses capacités.

4.0.4 Validation

Le principe de cette étape est simple, j'évalue la capacité qu'à mon réseau de neurones à reconnaître les chiffres entre 1 et 9, ou à dire qu'une image est vide si celle-ci ne contient aucun chiffre. En travaillant sur les 10 000 images de validation de notre base de données, le réseau de neurones obtient une précision de 97,35, ce qui est tout de même un très bon score. Sachant que la base de données MNIST contient de nombreuses images qui portent à confusion, comme deux chiffres imbriqués l'un dans l'autre par exemple, nous pouvons être relativement sereins quant au bon fonctionnement de la reconnaissance des caractères présent dans une grille de sudoku.

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 créé 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

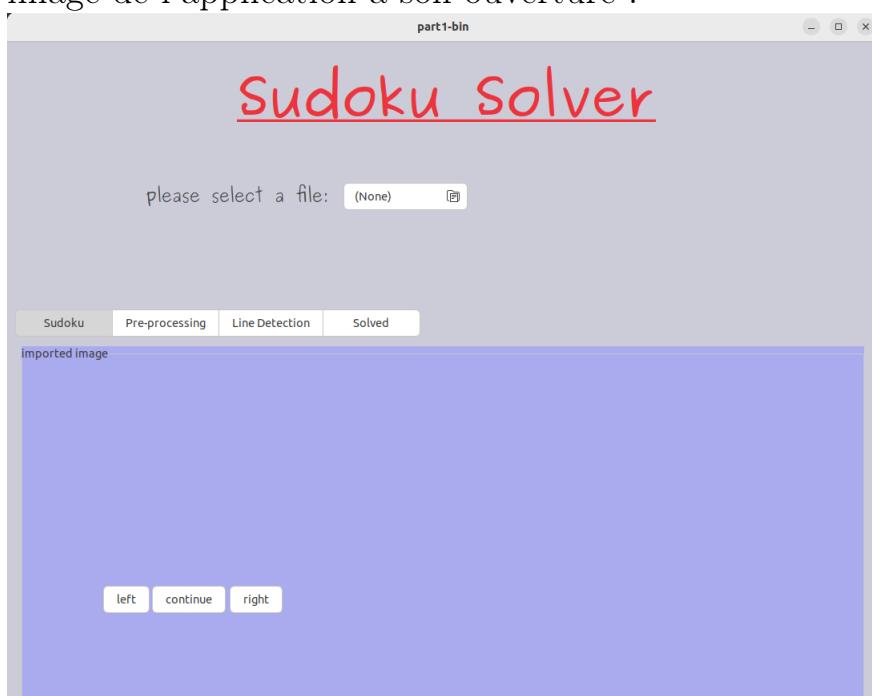
Interface

6.1 Section partie interface

6.1.1 creation de l'application

Tout d'abord il a fallu crée une interface graphique composé des éléments nécessaires à son futur fonctionnement. J'ai donc ajouté un bouton permettant de récupérer un fichier pour permettre de récupérer l'image du sudoku. Puis j'ai créé 4 onglet qui permettront d'afficher les différentes étapes de la résolution du sudoku. Ensuite dans les 3 premiers onglet j'ai ajouté un bouton continue pour permettre à l'utilisateur de lancer la partie suivante du programme. De plus, dans le premier onglet il y a un bouton droit et gauche qui permettront une rotation centrale vers la gauche ou la droite. Enfin j'ai mis un bouton permettant de choisir un fichier dans le dernier onglet pour que l'utilisateur choisisse où il désire enregistrer les résultats. J'ai aussi mis des textes et des fond de couleur pour une application plus belle visuellement.

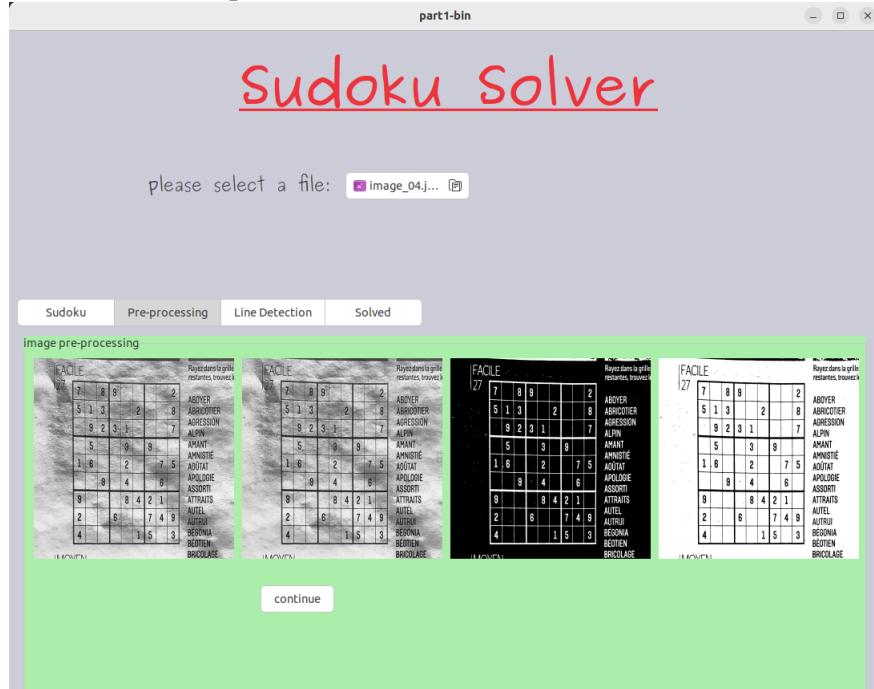
image de l'application à son ouverture :



6.1.2 mise en relation des différent programme

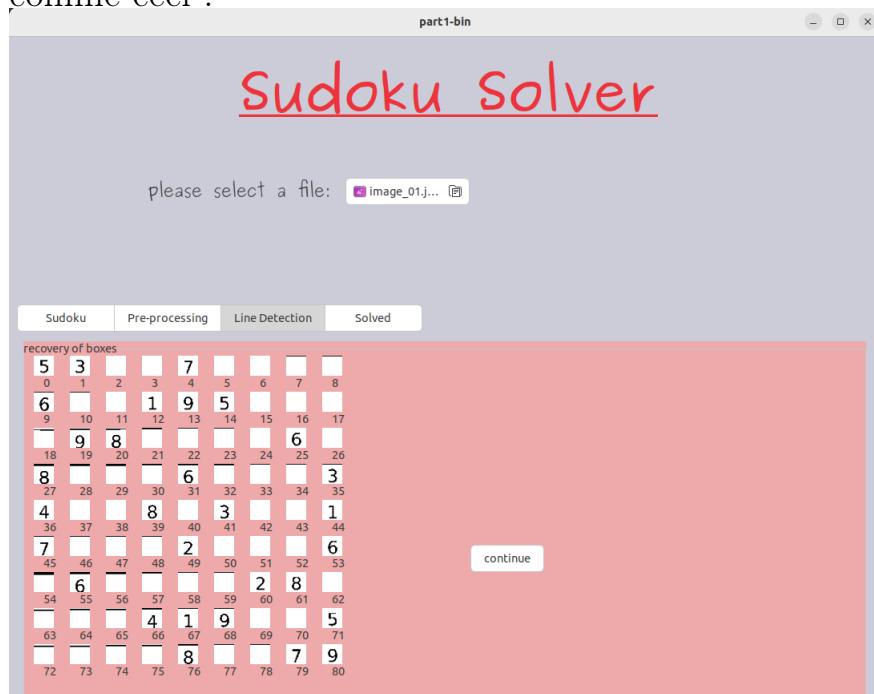
Une fois l'image choisie l'utilisateur peut la faire tourner vers la droite ou la gauche puis ensuite il doit appuyer sur continue pour lancer la partie suivante du programme qui est le pre-processing de l'image. Cela a aussi pour effet d'afficher l'image durant toutes les étapes du pre-processing.

voici un exemple :



Ensuite une fois que l'utilisateur appuis sur continue cela lance la détection de ligne et va afficher toutes les cases détectées dans l'onglet Line Detection.

comme ceci :



Enfin quand l'utilisateur appuis sur continue cela lance la résolution du sudoku qui va ensuite s'afficher sous forme d'image

Chapitre 7

Website

Afin d'appuyer et de mettre en avant notre projet nous avons décidé de réaliser un site Web. // <https://mathieu-21.github.io/OCRWebsite/>

Chapitre 8

Conclusion

8.1 Attentes et avancement

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

	Avancement soutenance finale	Objectif rendu finale
Prétraitement de l'image	100%	100%
Rotation de l'image	100%	100%
Détection de ligne	100%	100%
Découpage des cases	100%	100%
Réseau de Neurones Xor	100%	%
Réseau de Neurones final	100%	100%
Enregistrement des poids	100%	100%
Résolution du Sudoku	100%	100%
Interface	100%	100%

FIGURE 8.1 – Tableau récapitulatif des solutions

8.2 Problèmes rencontrés et conclusions personnelle

8.2.1 JESSY

Personnellement je me suis occupé de la détection de ligne et j'ai aidé à la rotation automatique, le sujet était plutôt complexe sur différents niveaux, mais le plus compliqué était sûrement le fait que nos algorithmes doivent compiler avec une multitude d'images différentes. Malgré cela le projet m'a vraiment plu et j'ai pu acquérir de nouvelle compétence en C et plus particulièrement pour la bibliothèque "SDL2" dont je suis servi tous le long du projet que ça soit pour la segmentation ou pour la rotation automatique.

8.2.2 MATHIS

Pour ma part, les impressions que j'ai sur ce projet que nous avons réalisé durant ce semestre divergent. En effet, celui-ci était intéressant et m'a permis de m'enrichir sur plusieurs sujets tels que la rotation d'une image. Cependant, j'ai eu beaucoup de mal à comprendre ce qui était attendu au départ et à faire les tâches que j'avais dû faire. Malgré ça le fait d'acquérir de nouvelles connaissances m'a permis de rester motivé à poursuivre les tâches qui m'étaient attribuer.

8.2.3 SATYA

Pour moi, le projet était une expérience enrichissante, mais vraiment ardue. Pour ce projet, je me suis occupé du réseau de neurones. Pour la première soutenance, c'était une tâche compliquée, mais assez accessible, car de nombreuses ressources étaient là pour m'aider. Pour la deuxième soutenance, la tâche était vraiment plus compliquée. Le premier problème rencontré était que les entraînements faisaient crash mon ordinateur et lorsqu'il ne faisait pas crash ça prenait énormément de temps. Ainsi, c'était assez compliqué de debugger. Le second problème était que toutes les documentations sur les réseaux de neurones étaient en python. Car le langage python possède des librairies fournissant déjà de nombreuses fonctions utiles. Le travail de recherches qui était nécessaire au développement de notre logiciel fut donc probablement la partie la plus difficile, les ressources étant pratiquement inexistantes. Le troisième problème et le plus important que je n'ai pas pu régler par manque de temps étaient l'implémentation du réseau de neurones à une image. J'ai décidé pour mon réseau de neurones finals d'utiliser la bibliothèque MNIST pour train mon réseau, mais je me suis rendu compte tardivement que la base MNIST était composé que de chiffres écrits à la main alors que les chiffres que je teste sont censés être des chiffres d'ordi. Je m'en suis rendu compte trop tardivement.

8.2.4 MATHIEU

Lors de ce projet je me suis occupé de toute la partie de prétraitement de l'image, ainsi que de la gestion des cases avant l'envoie dans le réseau de neurones. Je me suis également occupé du site Web pour la présentation du projet. J'ai personnellement apprécié ce projet que je réalisais pour la seconde fois, et ce pour la simple raison que les parties que j'ai effectué étaient plus intéressante que ce que j'avais traité l'année passée. J'ai néanmoins rencontré de nombreux problèmes lors du traitement de l'image. J'ai notamment passé beaucoup de temps à trouver un algorithme de calcul de seuil efficace qui fonctionnerait sur toutes les images, ayant finalement opté pour la technique de Otsu. Afin de rendre une image plus claire et utilisable, j'ai également essayé d'implémenter d'autres filtres et opérations sur

l'image comme la détection des bordures de Sobel, le filtre gaussien ou encore la saturation. Cela m'a néanmoins permis de me motiver et de trouver de nouvelles méthodes de prétraitement.

8.2.5 ENZO

Je me suis occupé de la mise en place de l'interface. J'ai trouvé le plus compliqué le fait de devoir faire en sorte que toutes les parties du projet fonctionne ensemble.

Mais cela m'a permis de devoir plus ou moins comprendre les différentes parties ce que j'ai trouvée très intéressant. De plus, ce projet m'a aussi permis de découvrir plus en détail la librairie gtk qui est très complète et utile. Tout ceci a rendue pour moi ce projet très intéressant.

8.3 Conclusion

Pour clôturer ce rapport, nous pouvons affirmer que nous avons pris plaisir à construire ce projet de bout en bout.

Tout au long de celui-ci, nous avons rencontré de nombreuses difficultés comme nous avons pu le développer auparavant. Cependant, cela ne nous a pas enlevé l'ambition de finir ce projet.

Malgré les lacunes sur certaines tâches, nous sommes tout de même satisfait compte tenu du temps de travail que nous avons mis à notre disposition pour le bien de ce projet. Mêlé à ça un sentiment de frustration car nous aurions aimé que tout marche à la perfection.

Ce projet nous a apporté à tous de manière générale des connaissances chacun sur des points différents.

De plus ce type de projet aide aussi à renforcer une organisation de travail, en groupe qui plus est. En effet, les hauts et les bas que nous rencontrons lors de cette période de projet sont plus ou moins difficile. Chacun d'entre nous à fait au mieux pour aider, soutenir ses coéquipiers et à rester motivés pour terminer ce projet dans les délais impartis.

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 05 Novembre 2022, <https://ieeexplore.ieee.org/docum>

-IPOL, (2015), Retrouvé 11 Décembre 2022, <https://www.ipol.im/pub/art/2016/158/ar>