SOUTENANCE 2

2021-2022

Projet OCR

Les MédiOCRes PrOCRastinateurs

Alexandre Devaux-Riviere

Jules Girod

Esteban Arroyo

Romain Ludet

Table des matières

1 Introduction - Présentation du sujet

L'objectif de ce projet est de réaliser un programme capable de résoudre un Sudoku à partir d'une image de la grille et uniquement à partir de cela.
L'image peut être de bonne qualité tout comme elle peut ne pas l'être, elle peut également avoir été tournée ou non.

Pour contourner ces problèmes, nous devons réaliser un traitement d'image afin d'obtenir une image la plus épurée possible et nette pour pouvoir reconnaître les chiffres à l'aide d'une intelligence artificielle. Cette étape de reconnaissance est appelée Optical Character Recognition d'où le nom du projet OCR.
Une fois les chiffres reconnus, nous devons résoudre le Sudoku et afficher le résultat à l'utilisateur via une interface.

2 Présentation

2.1 Présentation du groupe

Nous sommes "Les Médiocres Procrastinateurs" mais il ne faut pas s'y méprendre, nous travaillons beaucoup car nous sommes médiocres en procrastination.

2.2 Présentation des membres

Alexandre Devaux-Riviere :

Passionné d'informatique depuis mon stage d'observation de 3e, être considéré comme le technicien familial ne me suffisait pas et j'ai donc rejoint EPITA où j'ai acquéri la plupart de mes connaissances en programmation.
Aujourd'hui, en tant que chef de projet, mon rôle est de rester à l'écoute, faire tout mon possible pour épauler mes camarades ainsi que maintenir une organisation efficace afin que le projet aboutisse et soit une réussite. Ce projet est une bonne opportunité pour moi de contribuer au développement d'un OCR qui me plaît et dont je peux être fier.
Je vais et j'ai déjà acquéri de nombreuses connaissances et normalement assez d'expérience pour pouvoir passer du rang de « gueux » au rang « soup lord ». Hâte de passer ce palier symbolique.

Jules GIROD :

Bonjour, je suis Jules GIROD, j'ai 19 ans, je suis passionné d'informatique depuis longtemps. Avant EPITA, je m'étais déjà intéressé à la programmation C et un peu à la SDL, ce qui m'a donné des facilités pour commencer celle-ci et expliquer des méthodes/fonctions à mes camarades.
J'ai également fait un petit peu de traitement d'image en Python, ce qui m'a permis de connaître un peu les méthodes afin de réduire le bruit d'une image ou encore la binarisation, etc.
Ce projet me permet donc de travailler en groupe avec mes camarades sur un sujet qui nous intéresse tous.

Esteban Arroyo :

Bonjour, je nomme Esteban Arroyo, j'ai 20 ans, je suis passionné d'informatique, c'est pour cela que j'ai préféré faire sti2d.
De plus, j'aime connaître les dernières news sur les nouvelles technologies qui vont sortir. J'adore prendre des photos et les partager au plus grand nombre.
Ce projet est très enrichissant, car il nous permet de connaître les subtilités du C, langage qui sera très utile pour l'année prochaine en ING1.

Romain Ludet :

Bonjour, je m'appelle Romain Ludet, j'ai 19 ans. Je me suis très tôt intéressé à l'informatique et j'ai su très vite que c'était ce que je voulais faire. J'ai ainsi eu la chance de ne pas avoir à me casser la tête pour trouver une orientation professionnel pendant le lycée.
J'ai commencé par faire des petits jeux sur Scratch lorsque j'étais au collège, puis je suis rapidement passé sur un langage de programmation car je trouvais trop de limites et de contraintes à Scratch. Je me suis donc jeté sur les bases du C avant finalement de changer et faire du Python en classe de Terminale en spécialité ISN.

3 Répartition des tâches

	Alexandre	Jules	Esteban	Romain
Application (interface et système)				
Chargement de l'image en mémoire		X	X	
Prétraitement manuel		X		X
Sauvegarde de la grille		X	X	
Affichage de la grille résolue	X	X	X	
Interface graphique	X	X	X	
Traitement de l'image				
Détection de la grille				X
Prétraitement automatique				X
Suppression des couleurs		X	X	
Détection des cases de la grille		X		X
Récupération des chiffres présents dans les cases				X
Reconnaissance de caractères	X		X	
Reconstruction de la grille			X	
Réseau de neurones				
Réseau de neurones OCR (et test XOR)	X			
Apprentissage par propagation des corrections	X			
Sauvegarde et chargement des poids du réseau	X			
Jeu d'images pour l'apprentissage	X			X
Résolution du sudoku				
Ouverture et sauvegarde du fichier contenant la grille		X		
Résolution de la grille		X		
Communication				
Site Web	X			
Rendus Overleaf	X	X	X	X

Légende : X = Contribution

4 GTK - Interface

Dans cette partie, je vais vous présenter notre interface GTK qui nous permettra de manière intuitive de charger des images et d'appliquer tous les pré-traitements de manière automatique nous permettant d'afficher notre sodoku résolu.

4.1 Glade

Au début de la création de notre interface, nous avons utilisé Glade qui nous a permis de créer une interface de A à Z.

Nous avons ajouté des boutons, des checks, des zones images pour l'affichage et les différentes fenêtres (fenêtre principale et des sous-fenêtres qui seront affichées selon les options choisies). Ce logiciel va donc nous permettre de les placer dans les différentes fenêtres et de créer un *.glade.

Pour utiliser ce *.glade, il faudra l'initialiser dans un programme C pour ajouter les fonctionnalités de chaque élément (Widget). Donc, maintenant, je vais vous présenter le menu principal.

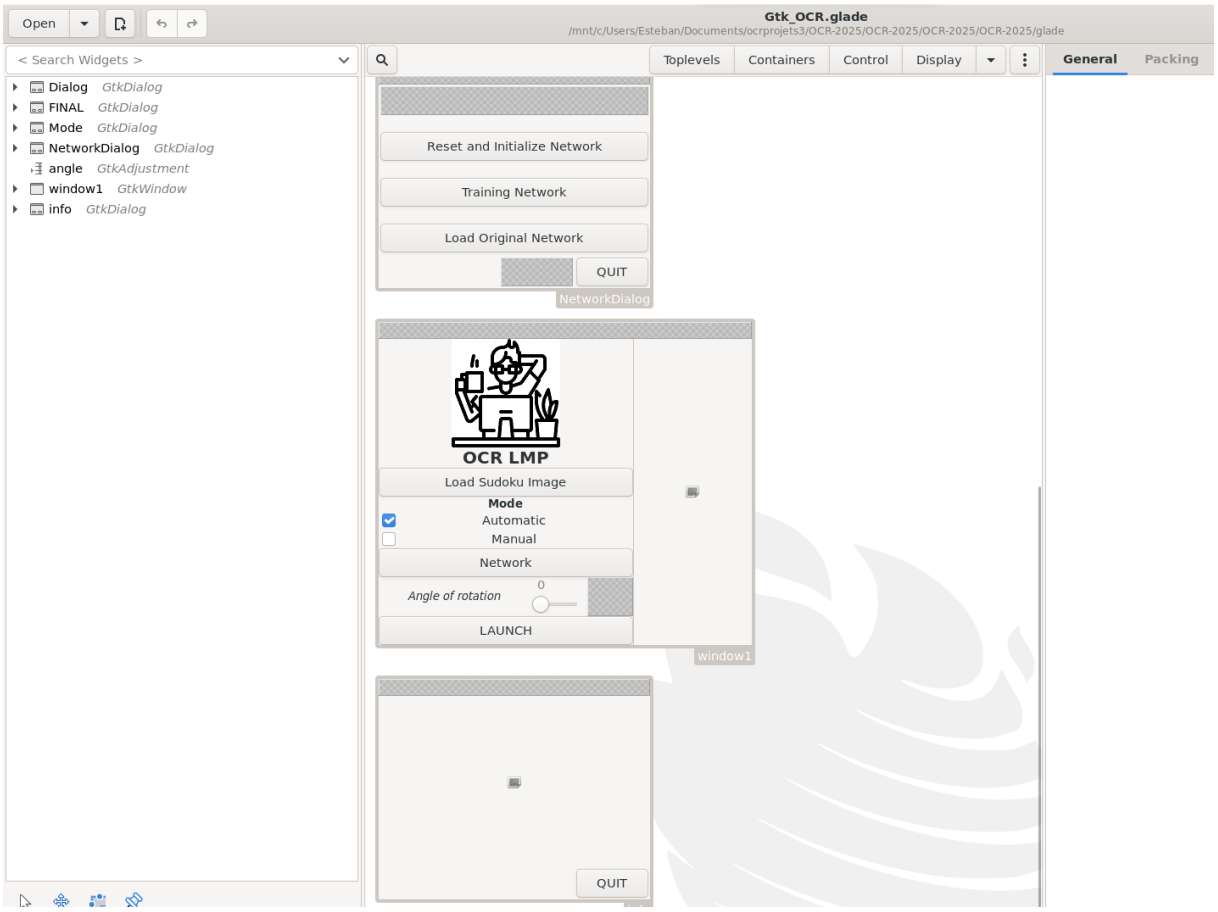


FIGURE 1 – Logiciel Glade

4.2 Menu Principal

Dans le menu ,il y a sur la droite une zone affichage pour l'image et sur la gauche ,il y a l'emplacement de chaque bouton .

Au-dessus de cette zone, nous avons un bouton qui va nous lancer un gtkFileChooser qui va nous permettre de choisir une image de sodoku. Lorsqu'il aura choisi une image, il sera affiché sur la zone d'affichage. Ensuite, nous avons deux checks, l'utilisateur pourra donc choisir entre un mode manuel ou automatique qui sera mis par défaut sur automatique.

En dessous, nous avons un bouton qui nous permettra de lancer une interface correspondant au Network.

De plus, nous avons GTKscale qui permettra à l'utilisateur de choisir l'angle de rotation de l'image et d'afficher la rotation en temps réel.

Pour finir, nous avons un bouton launch qui nous permettra de lancer une interface qui demandera si tous les paramètres sélectionnés précédemment comme la rotation sont corrects, avant finalement de lancer tous les programmes de résolution de la grille.

Je vais maintenant décrire en détail la fonctionnalité de chaque bouton.

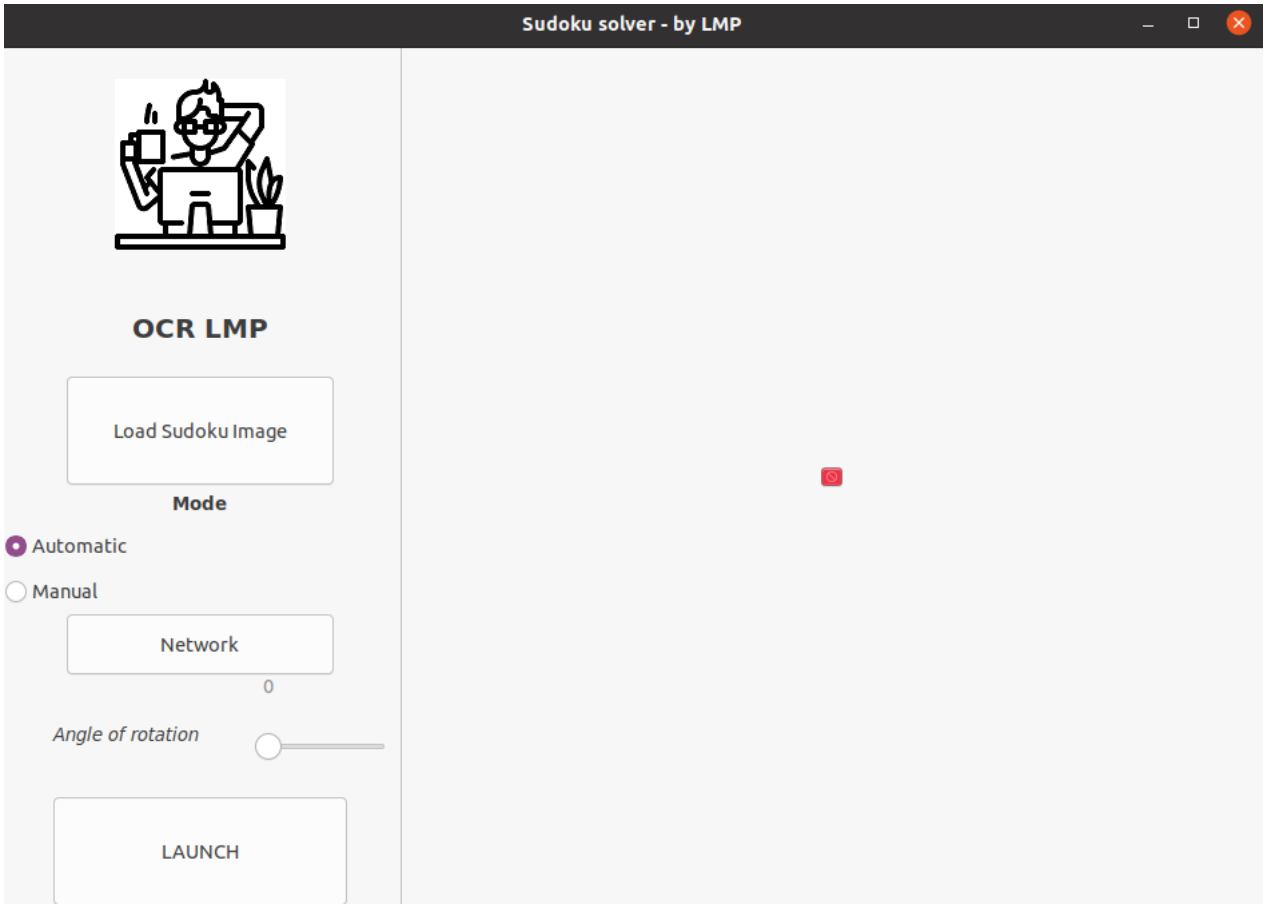


FIGURE 2 – Menu Principal

4.3 Fonctionnalités

4.3.1 Mode Manuel

Lorsque l'utilisateur sélectionnera le mode manuel, il passera en mode manuel. Cela veut dire que lorsque il cliquera sur le bouton Launch, il passera le traitement optimal qui s'appliquera à l'image, et il pourra visionner s'il le souhaite le rendu de chacun des traitements.

4.3.2 Mode Automatique

Lorsque l'utilisateur sélectionnera le mode automatique, il passera en mode automatique. Cela veut dire que lorsqu'il cliquera sur le bouton Launch, le traitement optimal s'appliquera à l'image.

4.3.3 Bouton Network

Lorsque l'utilisateur cliquera sur ce bouton, cela lancera une fenêtre de dialogue. Sur cette fenêtre, il y a trois boutons, le premier bouton permet de remettre à zéro le réseau neuronal, le deuxième bouton permet d'entraîner le réseau neuronal et le dernier permet de charger le réseau neuronal d'origine.

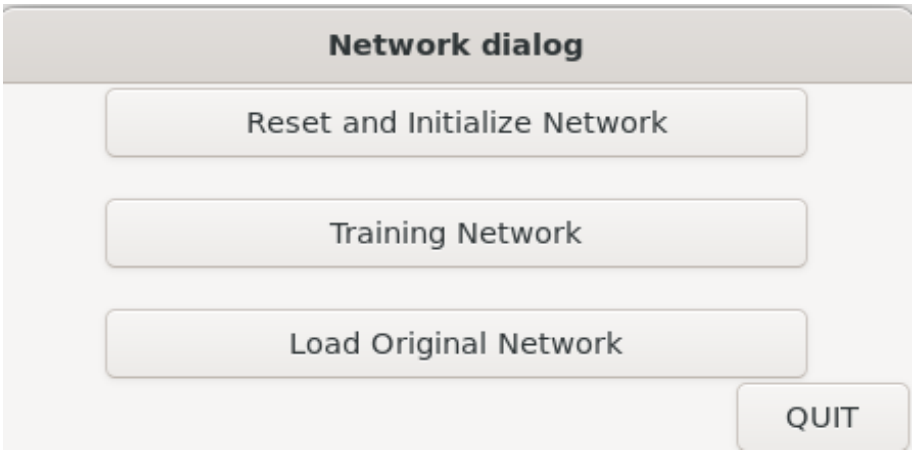


FIGURE 3 – Interface Network

4.3.4 Boutton Launch

Lorsque l'utilisateur cliquera sur le bouton Launch comme dit précédemment. Cela lancera une interface qui demandera si tous les paramètres mis précédemment comme la rotation sont corrects. Avant de lancer tous les programmes de résolution de la grille.

Donc lorsque l'utilisateur est s'attifait, il peut donc cliquer sur le bouton apply qui lancera les programmes de résolution de la grille donc quand le programme est terminé. Selon les modes choisis entre manuel ou automatique :

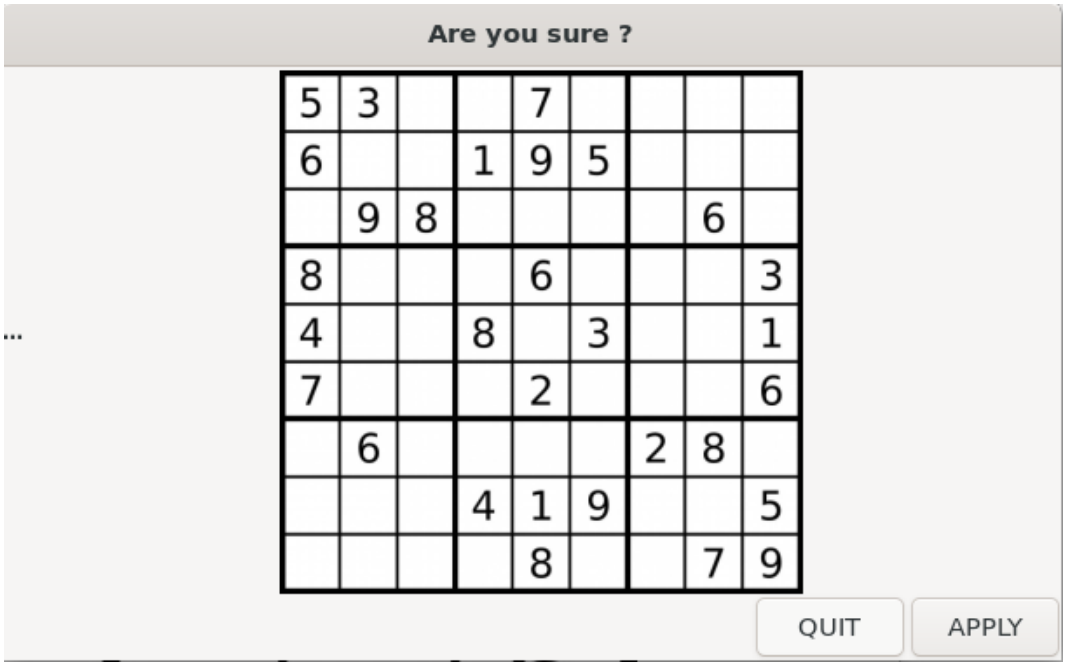


FIGURE 4 – Interface de vérification

AUTOMATIQUE :

Cette fonctionnalité affiche une interface qui montre le sodoku résolu et demande à l'utilisateur dans quel fichier il veut sauvegarder son fichier. Il peut quitter s'il ne veut pas sauvegarder le fichier.

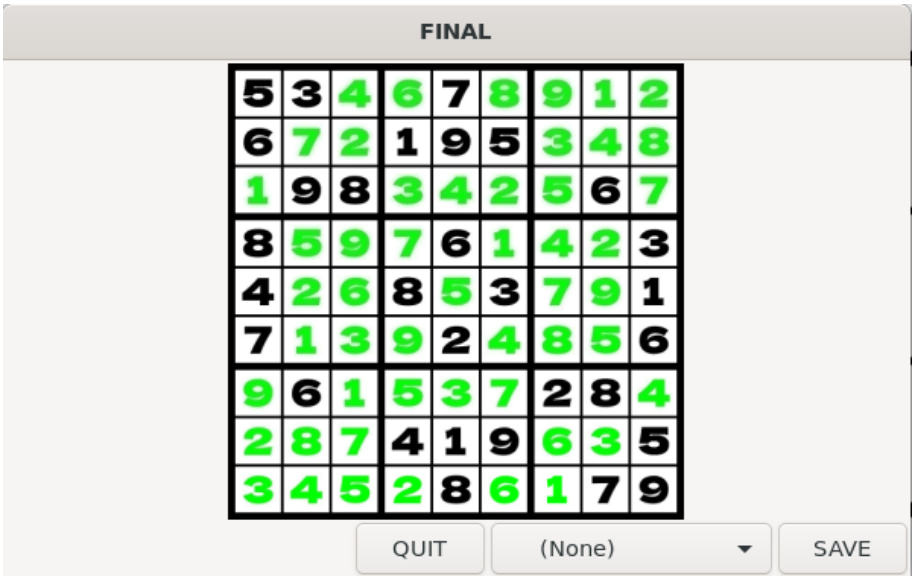


FIGURE 5 – Mode Automatique

MANUEL :

Cette fonctionnalité affiche une interface qui montre toutes les étapes avant de résoudre le sodoku. Ainsi, l'utilisateur peut afficher les étapes qu'il souhaite afficher sur l'interface principale.

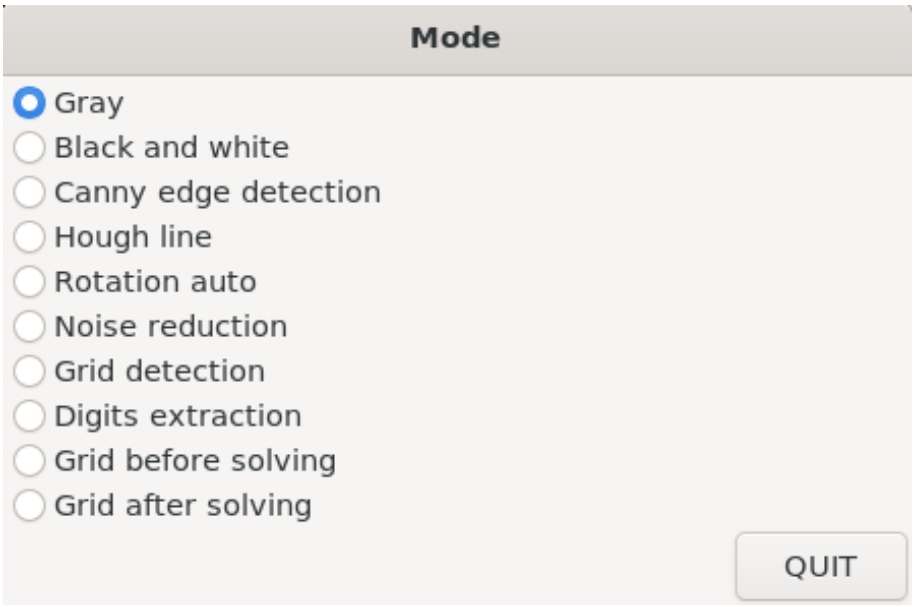


FIGURE 6 – Mode Manuel

4.4 Fenêtres de dialogue

Comme nous avons vu précédemment, chaque bouton utilise une interface externe. Ces interfaces externes permettent d'éviter de polluer l'interface principale et de moins se perdre entre les étapes : Nous avons deux types interface externes :
- La première est utilisée par le mode manuel et le bouton network. Dans ces interfaces, il n'y a que des boutons supplémentaires qui permettent d'afficher d'autre sous fonction.
- La deuxième est utilisée par tous les autres, elle permet d'afficher des images en dehors du menu principal.

5 Prétraitement

Le prétraitement d'une image est probablement la partie la plus importante afin d'avoir une reconnaissance de la grille la plus rapide et réussie possible. C'est pourquoi il faut procéder à plusieurs étapes successives.

6 Renforcement des contrastes

6.1 Chargement de l'image en mémoire

La première étape, la plus simple, est de charger l'image dans la mémoire.
Ainsi, le programme prend le chemin relatif ou absolu d'une image en paramètre. Cette image va être chargée en mémoire tout au long de l'exécution du programme.
Pour réaliser cela, nous avons utilisé la librairie : SDL1.2, ainsi que ses dérivées.

La bibliothèque nous permet de charger les images de type ".bmp". A terme nous offrirons la possibilité de charger des images.

Nous avons la possibilité via la SDL d'afficher l'image sur l'écran de l'utilisation. Pour ce premier rendu, nous avons fait le choix d'afficher la détection de la grille même s'il ne s'agit que d'une étape intermédiaire afin d'avoir un résultat concret de notre avancement.
A terme, nous afficherons l'image initiale résolue.

6.2 Conversion en niveaux de gris

La première étape pour retirer le bruit (les pixels parasites empêchent la reconnaisse de la grille) est de convertir notre image en nuances de gris.
Cette technique de conversion permet de retirer une majeure partie des détails inutiles à la reconnaissance de l'image.

Pour faire ce filtre, il suffit d'appliquer une formule mathématique 0.21 * r + 0.71f * g + 0.07 * b où r,g,b représentent la quantité de rouge, de vert et de bleu sur chaque pixel. Une fois la valeur obtenue on définit alors que la nouvelle couleur du pixel contiendra autant de rouge, de vert et de bleu, à savoir la valeur trouvée. On fait cela sur chaque pixel de l'image pour pouvoir obtenir une image en nuances de gris.

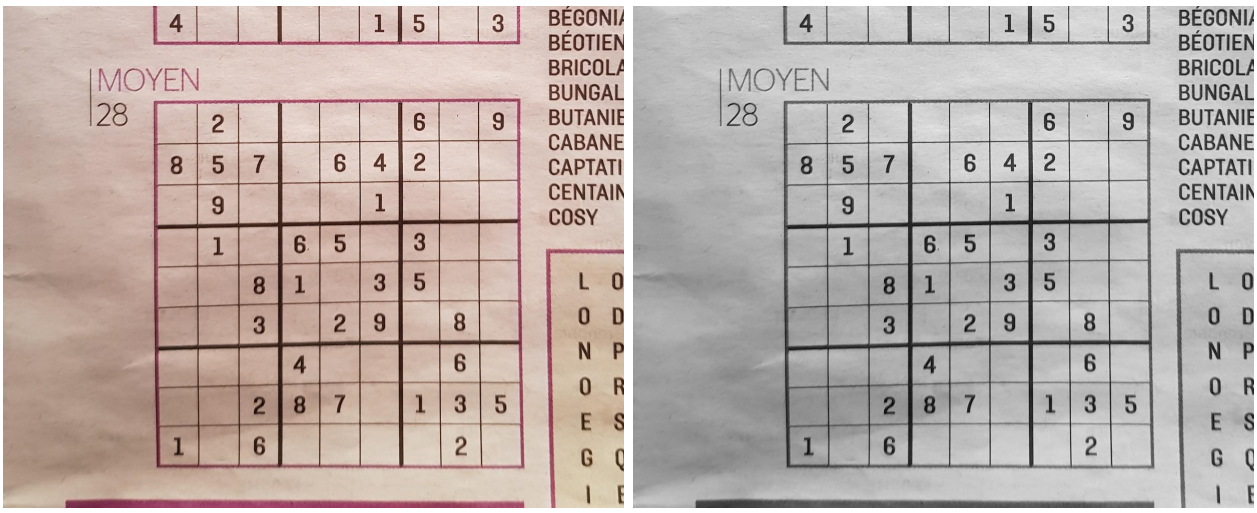


FIGURE 7 – Image convertie en nuances de gris

6.3 Conversion en noir et blanc / Binarisation

Pour convertir en noir et blanc la méthode la plus simple est d'étudier la valeur des pixels sur une image en nuance de gris, si la quantité de rouge (ou bleu ou vert car ils possèdent la même valeur) est supérieur ou égal 127 alors on peut convertir ce pixel en un pixel blanc à savoir r,g,b = 255,255,255 et si la valeur est inférieure alors r,g,b = 0,0,0, ce qui donne du noir.

Le problème majeur de cette méthode est que, pour les images avec peu de contraste, l'on perd beaucoup d'information et on se retrouve avec une image blanche. Il faut donc trouver un moyen de définir la valeur de seuil en fonction des contrastes globales de l'image.
C'est maintenant qu'entre en jeu la méthode d'Otsu.

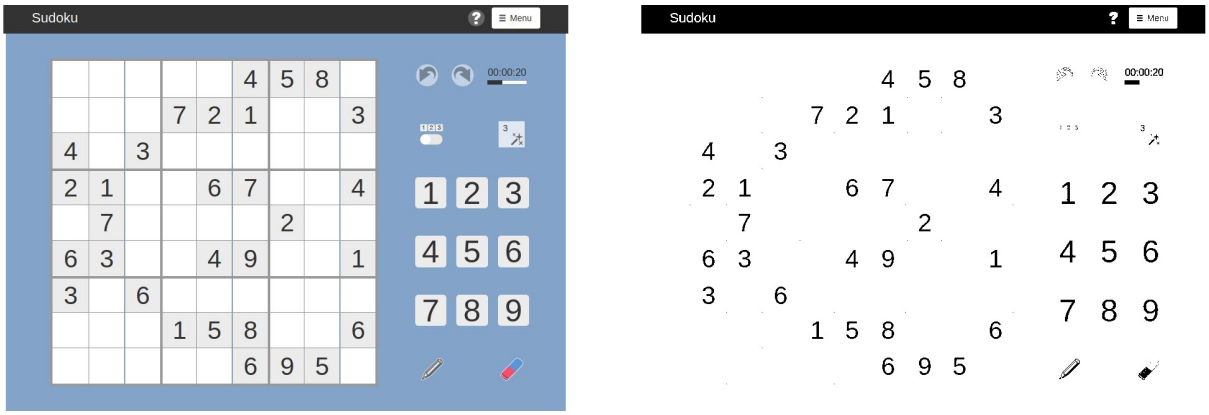


FIGURE 8 – Binarisation avec valeur de seuil = 127

6.4 Méthode de Otsu

La méthode d'Otsu est utilisée pour effectuer un seuillage automatique à partir de l'histogramme des pixels d'une image. L'algorithme suppose alors que l'image à binariser ne contient que deux types de pixels, noir ou blanc, ceux en premier plan et ceux en arrière-plan, puis calcule le seuil optimal afin d'obtenir une bonne séparation du premier plan et de l'arrière-plan.
Commme on peu le voir sur cette image, la méthode augmente grandement la performance de la binarisation.

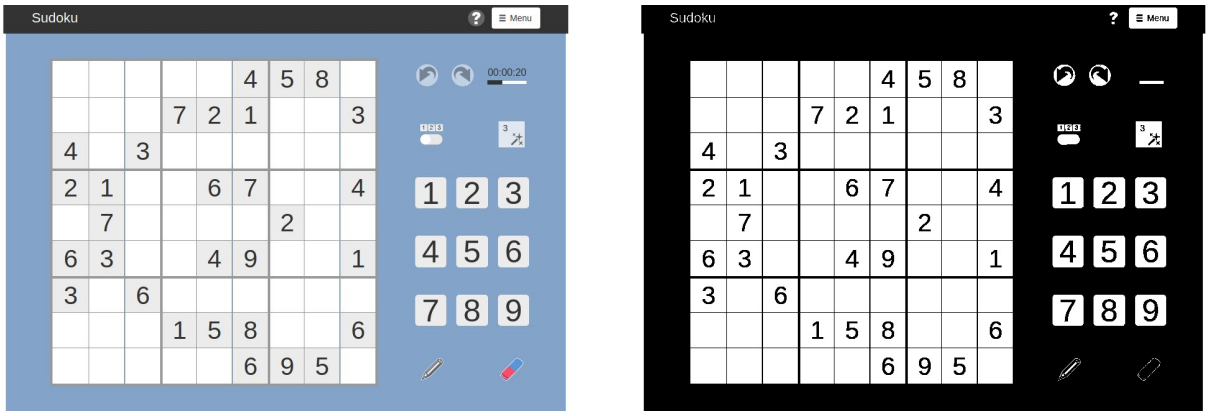
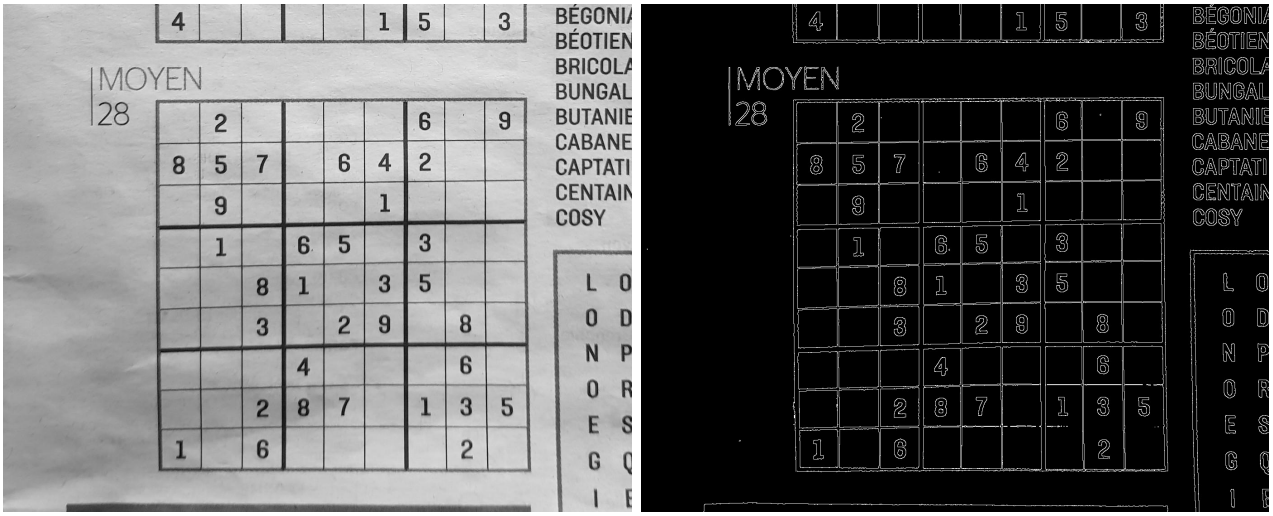


FIGURE 9 – Binarisation avec la methode d'Otsu / valeur de seuil = 195

6.5 Canny

Canny est une méthode pour traiter l'image, et en particulier ne garde que les coutours des figures géométriques. On lui donne une image en couleur (ou non) en entré et elle nous retourne une image en noir et blanc avec uniquement les contours. Cette étape est primordiale pour l'automatisation de la rotation de la grille, qu'on l'on explique dans le paragraphe associé.



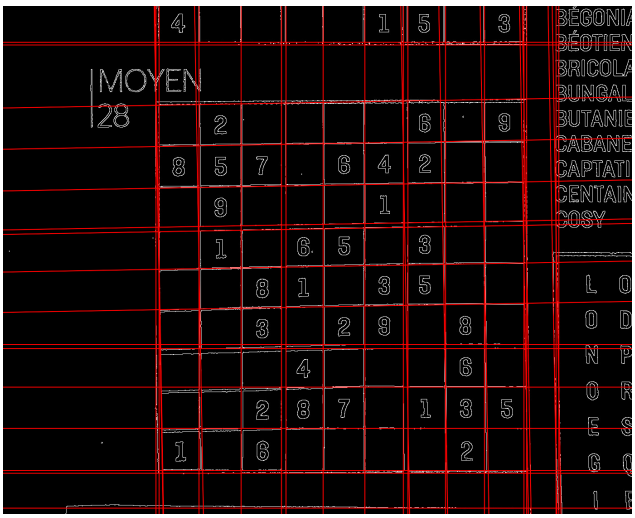


FIGURE 10 – Détection des bords via canny et détection des lignes via HoughLine

6.6 Hough line

Hough line est probablement la fonction la plus compliquer à coder car elle mélange les équations de droites, les sinus et cosinus ainsi que la base polaire. Ce qui est des éléments que nous manipulons rarement dans notre vie. Elle permet des détecte les lignes principales en ajoutant une contrainte de longueur sur la ligne, en effet, on peut choisir de ne détecter que les lignes de plus de 200 pixels de long ou alors au contraire détecter seulement les lignes de moins de 200 pixels de long. Vous l'aurez compris cette fonction correspond parfaitement à nos besoins car nous avons besoin de détecter les lignes de Sodoku de la grille.

Heureusement ou malheureusement pour nous, Romain avait commencer à la coder en C, cependant, après des heures de travail et quelque insomnie, son camarade Jules arrive avec HoughLine fonctionnelle grâce à au logiciel ImageMagick.
De plus, avec ImageMagick on peut lui faire générer un fichier contenant les coordonnées du points de départ de la droite et celles du point d'arrivée (Il ne s'agit pas réellement de droite mais plus de segment qui rempli parfaitement l'image), mais aussi les angles pour chaque droite avec l'axe des abscisse et la longueur des segments. C'est en réalité à partir de ce fichier et notamment de l'angle que l'on va pouvoir déterminer l'angle de rotation automatiquement.

6.7 Rotation de l'image

Pour le confort de l'utilisateur, notre solver doit pouvoir reconnaître la plus grande palette d'images possible et pas uniquement les images où la grille est parfaitement droite. C'est pourquoi il est important de prendre en compte les images où la grille est désaxée.
Plutôt que de faire une méthode de reconnaissance de la grille qui la trouve même si elle est en diagonale, il est préférable de faire une rotation avant tout traitement.

Lors de la première soutenance nous demandions à l'utilisateur de saisir manuellement l'angle de rotation (0 modulo 360 s'il ne veut pas faire de rotation), cependant cette technique est contraignante car on demande on l'utilisateur de saisir l'angle, ce qu'il n'est pas forcement nécessaire car on pourrait détecter automatiquement si la grille doit être tourné ou non.

Ca tombe bien car c'est exactement ce que nous avons fait.

Pour ce faire nous réalisons "Canny edge detection" de image magick afin de ne sortir uniquement les contours des figures géométrique par exemple sur un carré nous aurions une image noir avec en blanc uniquement les 6 arrêtes du carré. Cette étape permet des réduire la quantité d'information à traiter pour la rotation.

En effet, dans un Soduko ce qui va nous permettre de savoir de combien de degre il faut faire la rotation c'est la grille car si on arrive à obtenir deux côtés perpendiculaires de la grille on peut tout simplement savoir le nombre de degré il faut faire, cependant nous ne pouvons pas s'il s'agit d'une rotation vers la droite ou vers la gauche car une grille de Sodoku est identique peut importe le sens. C'est pourquoi nous avons ajouté deux boutons sur l'interface, une rotation de 90 degrés vers la droite ou vers la gauche.

Une fois qu'on a fait le pré-traitement canny sur l'image binarisée, on donne cette image à la fonction "Hough Line" afin d'avoir pour chaque droites principales les coordonnées de deux points ce qui nous permet d'avoir l'équation de la droite et donc l'angle avec l'horizontale.

Comme on peut le voir sur l'image ci dessous, les lignes rouges sont les lignes que la méthode "Hough Line" a réussi à trouver. Ainsi, on fait la moyenne des angles modolu 90 pour avoir le nombre de degré pour la rotation.

Une fois la moyenne calculé, si l'angle est compris entre [0,10[ou [80,90[alors on ne fait pas la rotation car notre découpage de grille marche pour des petits angles de rotation cependant sur l'image05.jpeg nous devons faire une rotation car sinon la détection de la grille ne marche pas.

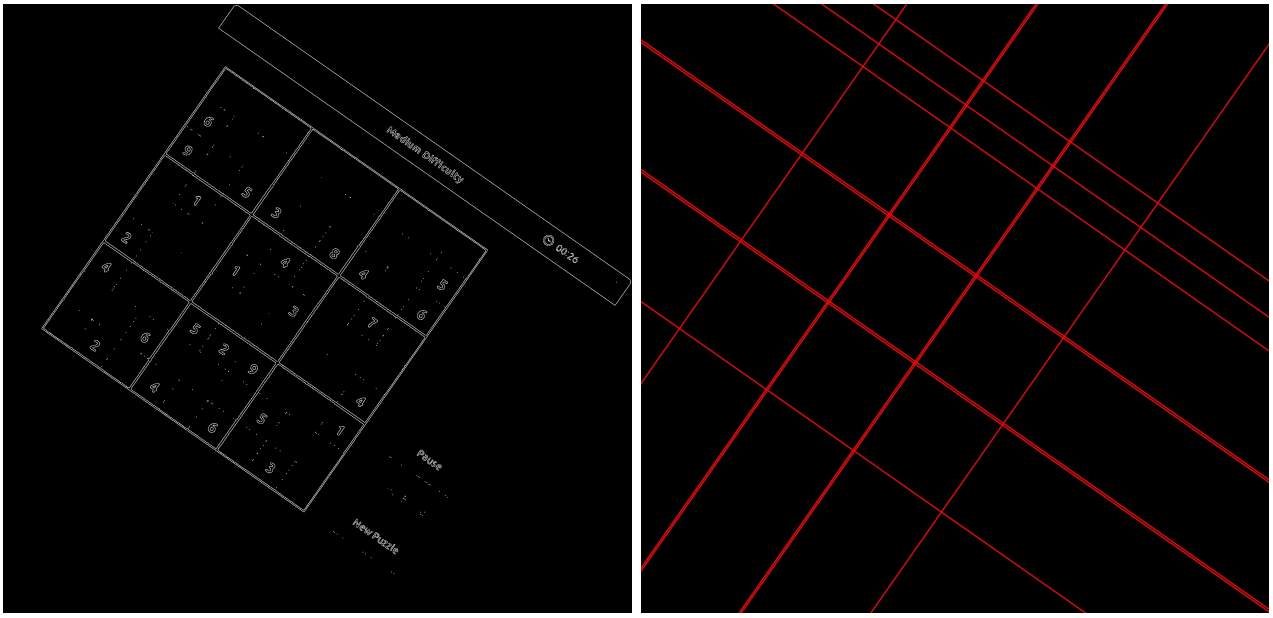


FIGURE 11 – Image après rotation et binarisation, angle = -35°

Lors de la première soutenance, afin de réalisé la rotation nous utilisions la fonction rotozoomsurface qui appartient à une librairie qui est une extension de la librairie SDL à savoir SDL_gfx.
Cependant, cela rajoute des bords blancs autours de l'image ce qui nous empêche de bien détecter la grille comme on vous l'expliquera dans le paragraphe associé à la détection de la grille.

Ainsi pour gagner en efficacité, c'est-à-dire recoder les fonctions rotations nous faisons appelle au logiciel ImageMagick qui est un outil de traitement d'image déjà installer sur les ordinateurs. Il permet ainsi de choisir la couleur du fond lors de la rotation par default le fond est blanc ce qui est parfait pour nous. Vous trouverez plus d'informations sur ImageMagick dans le paragraphe associé.

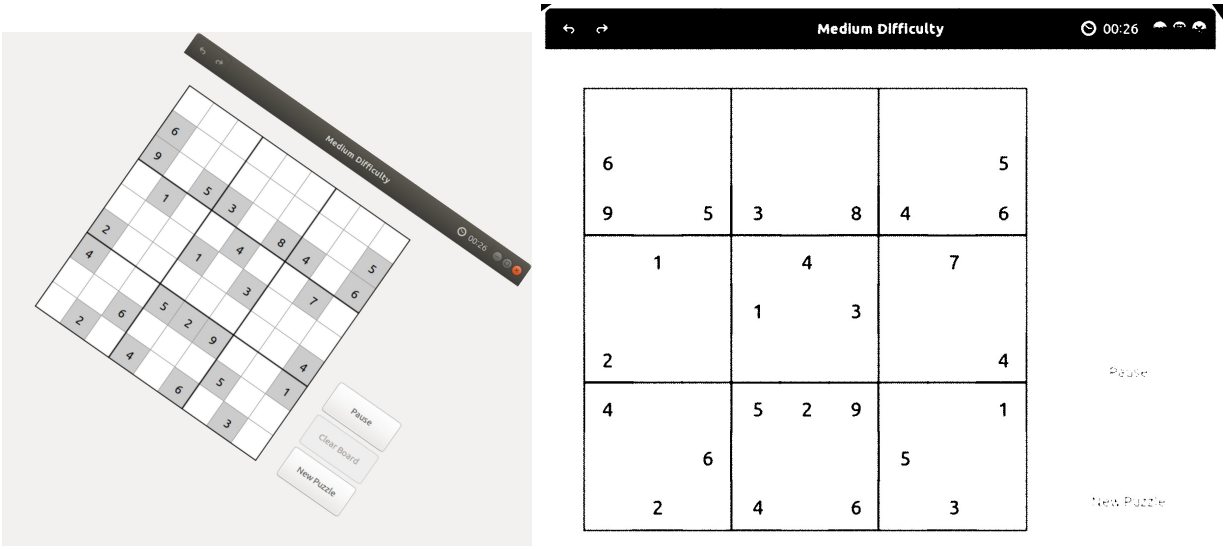


FIGURE 12 – Image après rotation et binarisation

6.8 Réduction du bruit

Toujours dans l'optique de ne garde que les détails qui nous importe, nous réalisons un flou sur l'image pour diminuer le bruit comme il peut y avoir sur l'image4 par exemple. Pour ce faire on crée une nouvelle image avec pour même dimension que l'image initiale, on réalise ensuite un parcours sur toutes les pixels de l'image et sur chaque pixel on fait la moyenne des couleurs des 8 pixels autour, pour ensuite ajouté ce pixel "moyen" à l'image créée précédemment. Ainsi comme vous pouvez le voir le bruit sur l'image4 à largement diminuer. Nous réalisons ce filtre sur l'image juste avant d'extraire la grille pour pouvoir détecter uniquement la grille et pas le bruit comme faisant partie de la grille.

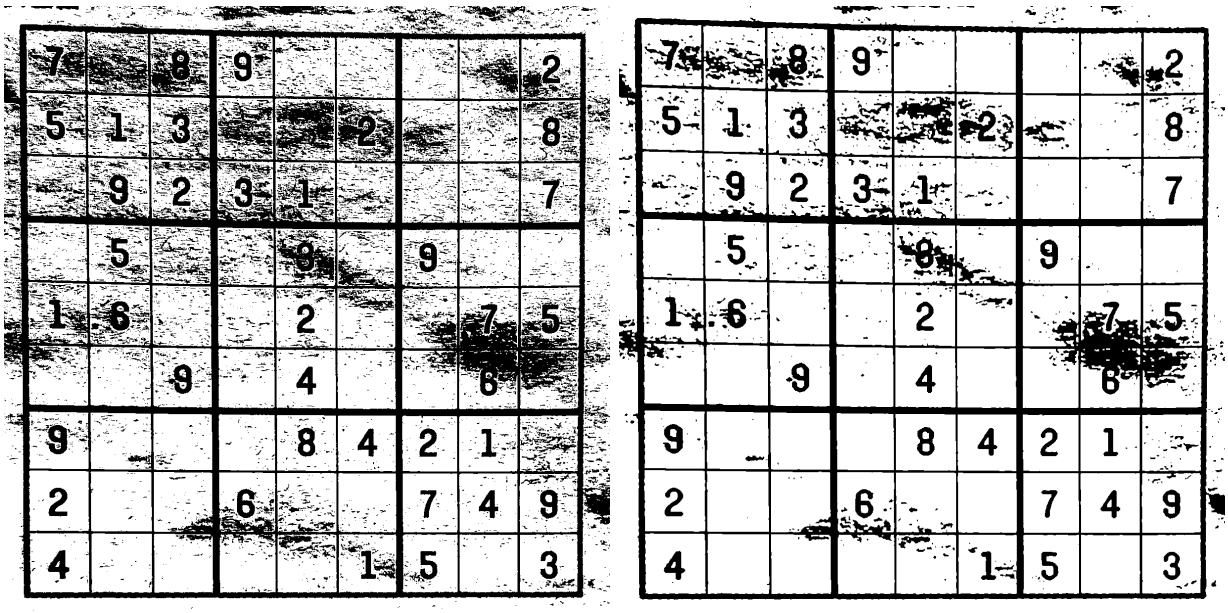


FIGURE 13 – Réduction de bruit

7 Détection de la grille

La détection de la grille est la deuxième étape la plus importantes après le pré-traitement. Nous souhaitons détecter la grille de la manière la plus rapide possible mais aussi de la manière la plus optimale possible malgré le bruit ou les imperfections de chaque image. Ce sont justement les différences entre chaque image qui rend la tâche compliquée. Chaque image est unique et pourtant, il faut réussir à l'aide d'une même détection à détecter correctement la grille. Voici donc les différentes étapes pour y parvenir.

7.1 Image connexe

La première étape consiste à détecter tous les composantes connexes de l'image. Une composante connexe est un groupe de pixels noirs qui ont pour caractéristique d'être tous voisins directs en diagonale ou sur les côtés. Pour ce faire, nous créons d'abord un tableau de même taille que l'image que nous initialisons à -1. Nous allons ensuite parcourir l'entièreté de l'image en appelant la fonction neightbors_grille() si un pixel noir est trouvé et que la case lui correspondant dans le tableau n'est pas différente de -1.

La fonction neightbors_grille() va se charger de trouver et marquer tous les voisins du pixel donné en paramètre et de leur attribuer pour chaque pixel d'un même élément un nombre. Ce nombre va correspondre au numéro de la composante connexe. Ainsi, une fois la grille totalement parcourue, nous obtenons toutes les composantes connexes de l'image et leur nombre.

7.2 La plus grande composante connexe

Après avoir réussi à stocker dans notre tableau toutes les composantes connexes de l'image, nous nous attaquons à la détermination de la grille. La grille est un élément de l'image avec généralement beaucoup de pixels collés les uns aux autres. Cette caractéristique nous permet de savoir que la grille est l'une des plus grande composante connexe de l'image. Seulement, si ce n'est pas le cas, comment procéder ?

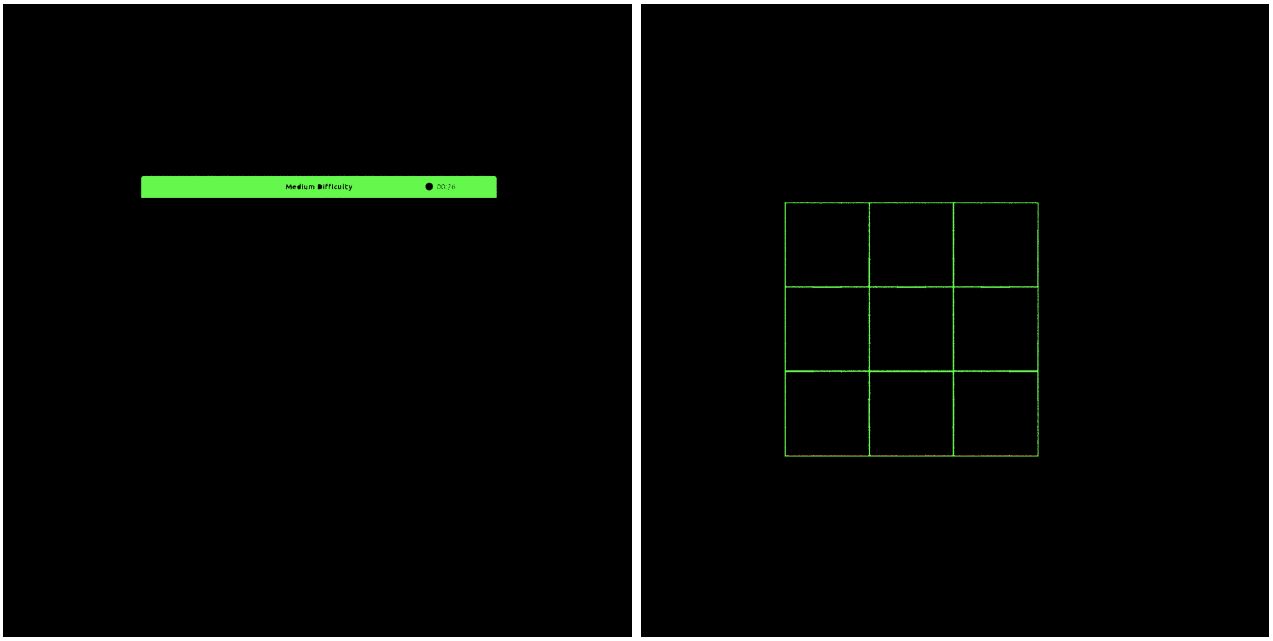


FIGURE 14 – Comparaison entre la plus grande composante connexe et la réelle grille

7.3 Déterminer à coup sûr la grille

Bien que nous ayons à notre disposition les différentes images connexes ainsi que leur taille, comment être sûr que nous avons bel et bien détecter la grille. Nous allons prendre chaque composante connexe de la plus grande à la plus petite. Pour chaque composante connexe, nous allons compter le nombre de colonnes et le nombre de lignes. En parcourant la composante connexe, on compte le nombre de fois où l'on rencontre 4 ou 9 intersections sur l'axe x et l'axe y. Nous détectons 4 ou 9 lignes/colonnes car sur certaines images, avec une qualité dégradée, seul les lignes principales de la grille sont visibles après la mise en noir et blanc. Si les nombres obtenu sur l'axe x et l'axe y sont supérieurs à 9, on est sûr d'avoir trouvé la grille de sudoku.

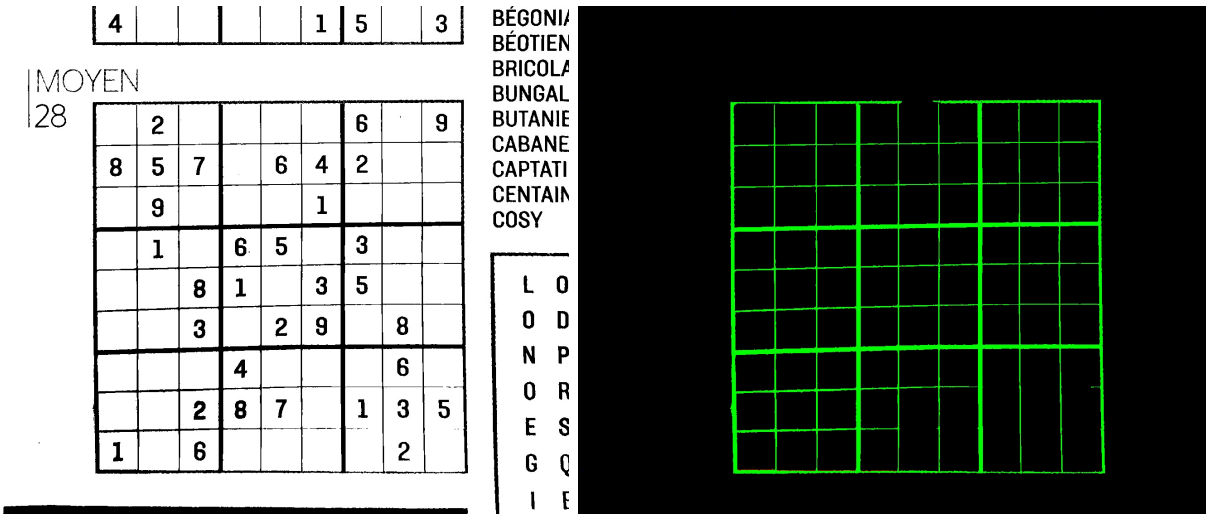


FIGURE 15 – Détection de la grille

7.4 Re-dimensionner l'image pour ne faire apparaître que la grille

Afin de découper notre image et d'obtenir tous les chiffres de cette dernière, nous devons retirer le plus de bruit possible. Même si nous avons trouvé la grille grâce aux composantes connexe, nous n'avons toujours pas les coordonnées des quatre coins de la grille. En effet, si l'image est de mauvaise qualité, le bruit collé à la grille aura été détecté comme faisant parti intégrante de la grille et cela risque de fausser la détection des cases. Pour éviter cela, nous allons donc de nouveau traiter la composante connexe obtenue. Nous allons stocker dans quatre variables les coordonnées du premier et du dernier pixel noir rencontré pour l'axe x et y à chaque fois que l'on a 10 ou 4 colonnes/lignes en sachant que l'on considère qu'il doit y avoir un espace minimum de 20 pixels en chaque ligne/colonne pour éviter de prendre en compte le bruit.

Pour obtenir les coordonnées exactes de la grille, il suffit de diviser la variable avec les coordonnées en x du premier pixel par le nombre de colonne, et de diviser la variable avec les coordonnées en x du dernier pixel par le nombre de colonne pour obtenir respectivement les coordonnées en x de la première colonne et de la dernière. Il en est de même pour l'obtention des coordonnées de la première et de la dernière ligne.

Nous n'avons plus qu'à couper l'image aux coordonnées des quatre coins de la grille pour obtenir une image contenant uniquement la grille. Cela signifie que le bord de gauche correspond à la première colonne, le bord de droite à la dernière colonne, le bord du haut à la première ligne et le bord du bas à la dernière ligne de la grille.

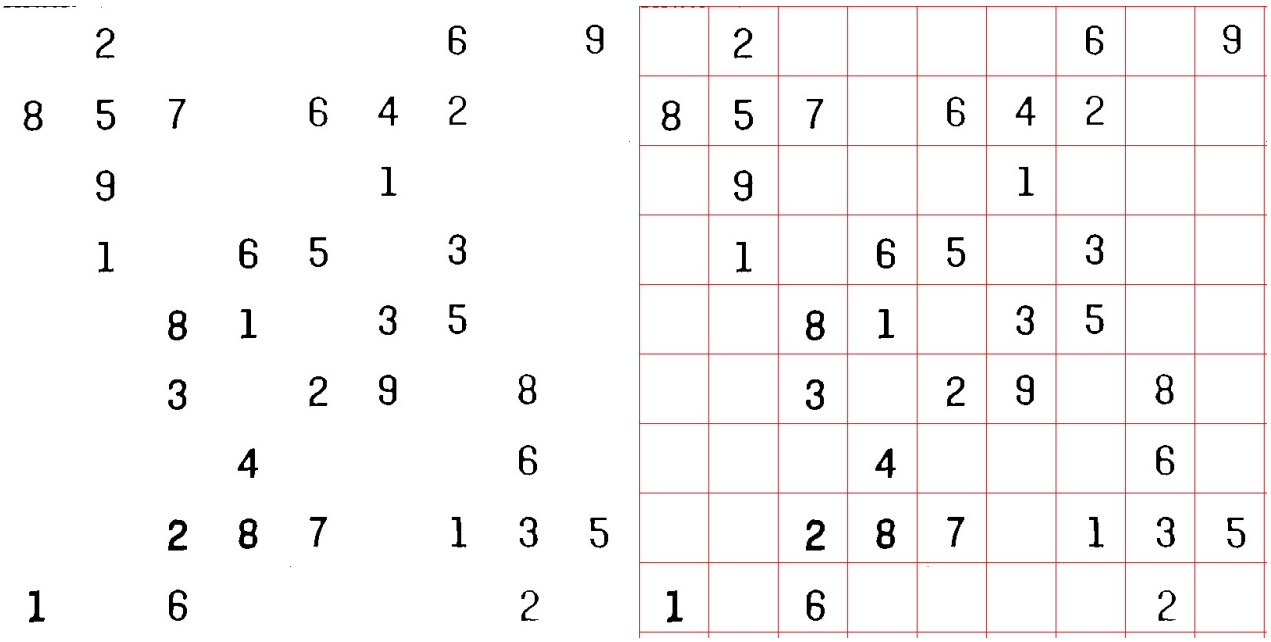


FIGURE 16 – Redimensionnement et affichage des lignes/colonnes

8 Découpage de la grille

Grâce au traitement précédent, nous avons directement une image avec la bonne rotation et avec les dimensions exactes de la grille.

8.1 Le découpage de la grille

Après avoir récupéré les coordonnées des dix lignes présentes en X en divisant la largeur de l'image par 9, et les coordonnées des dix lignes présentes en Y en divisant la hauteur de l'image par 9, le découpage de la grille peut commencer. Pour cela, il a fallu parcourir la grille en fonction des coordonnées. Il faut ainsi parcourir la grille à partir du premier point X jusqu'au suivant, et du premier point Y jusqu'au suivant, et ceci ainsi de suite pour toutes les coordonnées (X,Y) des lignes.
Pour chaque parcours, il faut créer une nouvelle image ayant pour dimensions celles qui séparent les lignes,

puis la remplir en copiant les pixels de l'image de référence en remplaçant les pixels correspondant à la grille par des pixels blancs pour faciliter par la suite le traitement des chiffres par l'IA. Une fois cela fait, avant de passer aux coordonnées suivantes, il faut sauvegarder l'image, dans un dossier spécial pour éviter d'encombrer le dossier actuel. Chacune des 81 images est sauvegardée avec comme nom les coordonnées du positionnement de sa case dans la grille.

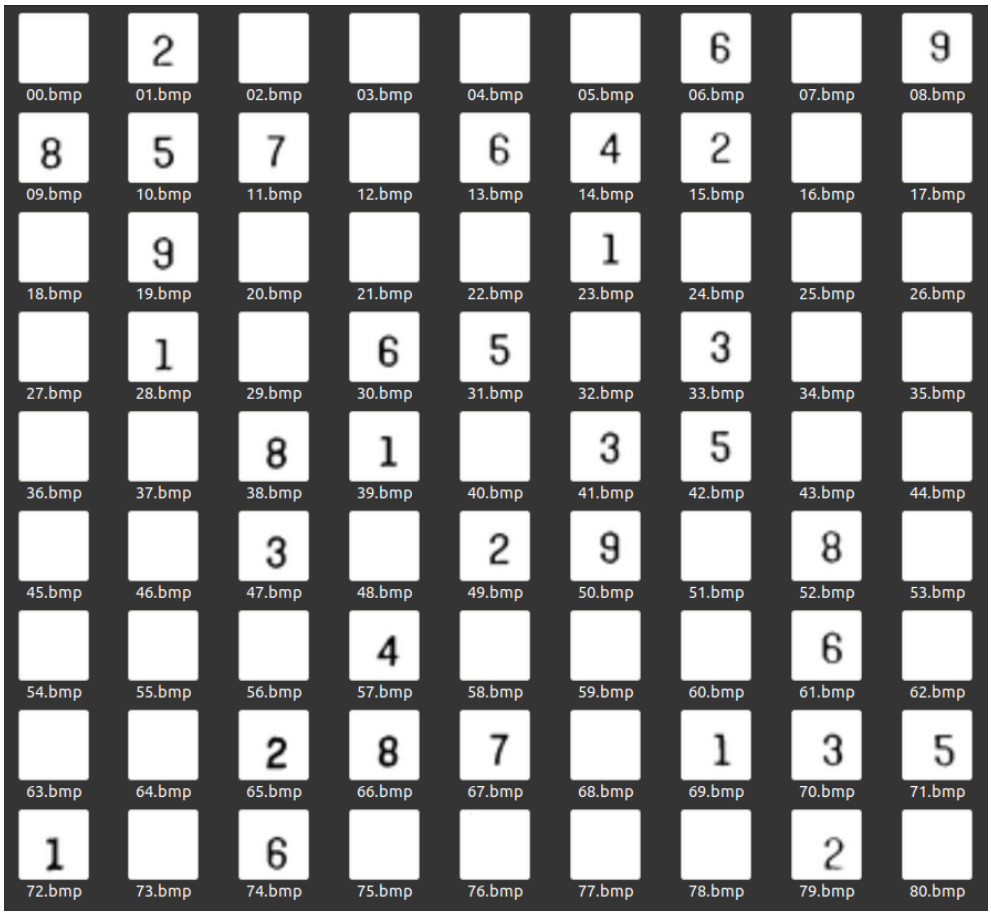


FIGURE 17 – Découpage des images

9 ImageMagick

Pour gagner du temps, nous avons également utilisé imagemagick. Il s'agit d'un logiciel de traitement d'image, il est surtout utiliser pour afficher une image depuis un terminal mais il permet également de faire des fonctions tel que la conversion en noir et blanc, la rotation d'une image, la fusion de plusieurs images, hough line, perspective et j'en passe.

Étant installer sur les ordinateurs de l'école, ce dernier est utilisé par tout les étudiants sans même savoir la puissance de ce logiciel, en effet, la majorité des groupes l'ont utilisé pour afficher les images dans le terminal avec la commande "display" suivi du chemin de l'image.

Nous nous sommes permis de l'utiliser pour faire la rotation de la grille, hough line, la création de l'image résolue et canny edge detection.

FIGURE 18 – Logo du logiciel "imagemagick"

Comme vous pouvez le voir sur la figure ci dessus, le logiciel illustre parfaitement la magie et le côté enfantin que devient l'OCR.
Cependant, nous n'avons pas toujours utilisé imagemagick par exemple, la conversion en nuance de gris, la binarisation, le flou, la détection de la grille via "composantes connexes".

10 Récapitulation des étapes

10.1 Les étapes finales

— Conversion en nuance de gris
— Conversion en noir et blanc
— Détection des figures avec canny
— Détection lignes avec Hough line
— Rotation
— Réduction du bruit
— Détection de la grille
— Découpage de la grille
— Passage des chiffres dans l'IA
— Sauvegarde des chiffres reconnus
— Résolution de la grille

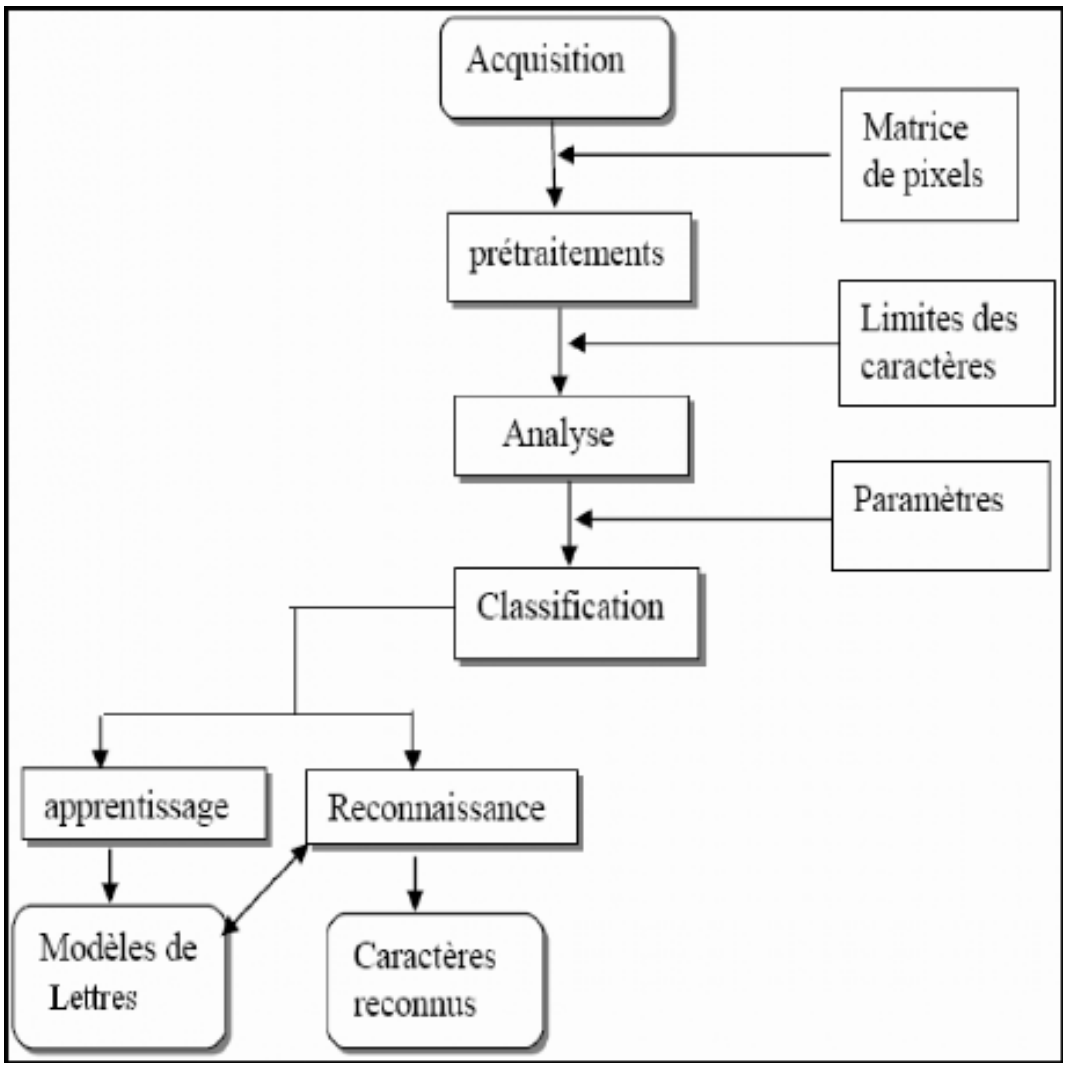


FIGURE 19 – Récap des étapes

10.2 Les autres pistes que l'on a essayé

Pour la détection des différentes composantes connexes, nous avons d'abord essayé d'implémenter la fonction neightbours_grille() de façon récursive mais cela s'est révélé être un échec. En effet, les images étant de dimension importante, la récursion était bien trop grande et le langage C n'était pas capable de la faire. Nous avons eu le droit à une petite erreur de "segmentation fault (core dumped)" qui nous à fait déboguer pendant un long moment avant d'en trouver la cause : une trop grande récursion.

Pour la détection de la grille nous avons essayer d'implémenter "blob detection" qui est très similaire à "composantes connexes". En effet, avec blob nous aurions pu récupérer facilement les 4 coordonnées des sommets de la grille et donc d'appliquer une fonction de perspective sur la grille, cela aurait permis d'aplatir l'image pour obtenir un carré parfait. Cependant nous n'avons pas réussi à implémenter blob et donc réaliser perspective.

Voici un résultat que nous aurions pu avoir sur l'image 6 (celle où la fonction perspective est le plus utile)

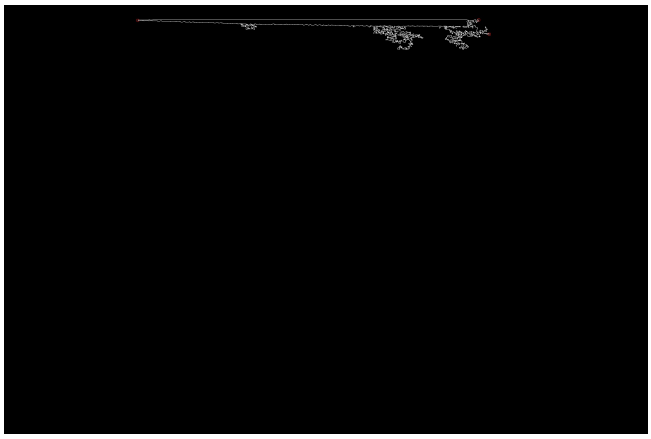


FIGURE 20 – Ici "blob detection" ne marche pas et détecte du bruit comme étant la zone la plus grande

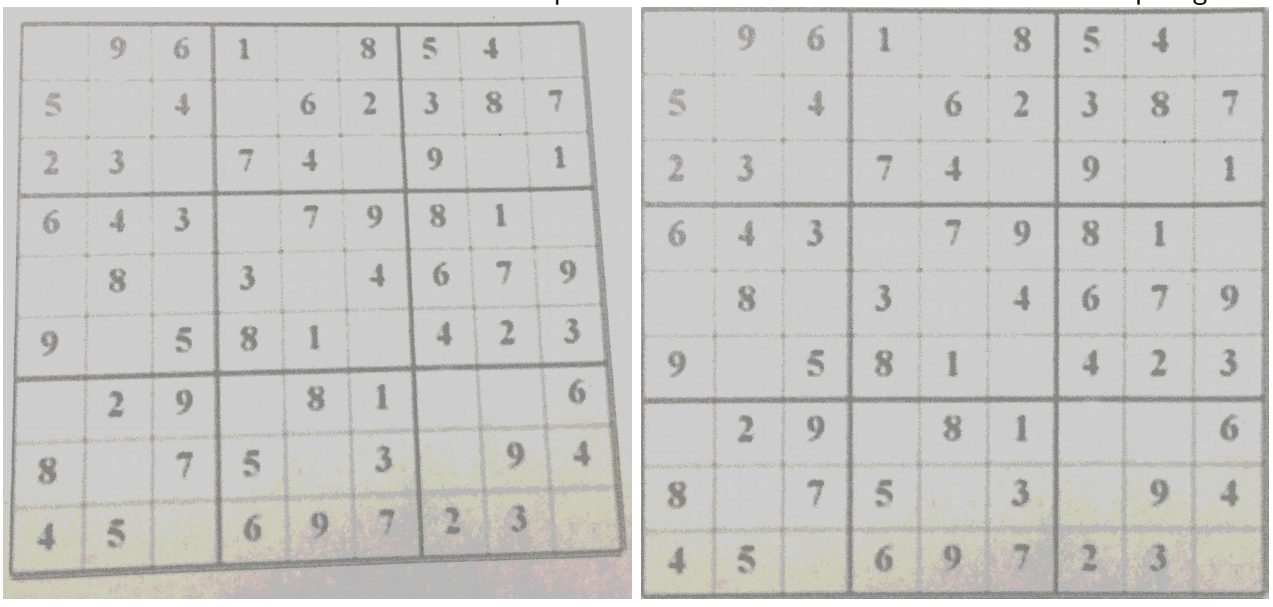


FIGURE 21 – Example de perspective sur l'image6

Nous avons également expérimenter de faire une augmentation des contrastes pour pouvoir bien différencier les couleurs et donc avoir un meilleur résultat avec "Otsu" et "Canny". Malheureusement pour obtenir un résultat signification ne devons appliquer une dizaine de fois l'augmentation des contrastes et cela prend du temps. De plus, il s'avère que ce n'est pas forcement nécessaire.

La dilatation est un procédé qui consiste à épaissir les lignes déjà présentent sur la grille de sudoku. Pour ce faire, il faut parcourir chaque pixel d'une image en noir et blanc. La méthode consiste à ajouter des pixels noirs autour d'un pixel déjà noir environ 5 suffises. Ce procédé permet d'élargir et de combler certaines lignes. Cela élargit également légèrement les chiffres, d'où le problème majeur car à cause de cela les chiffres deviennent trop gros et donc difficile à reconnaître pour un humain alors n'imaginons pas pour une IA.

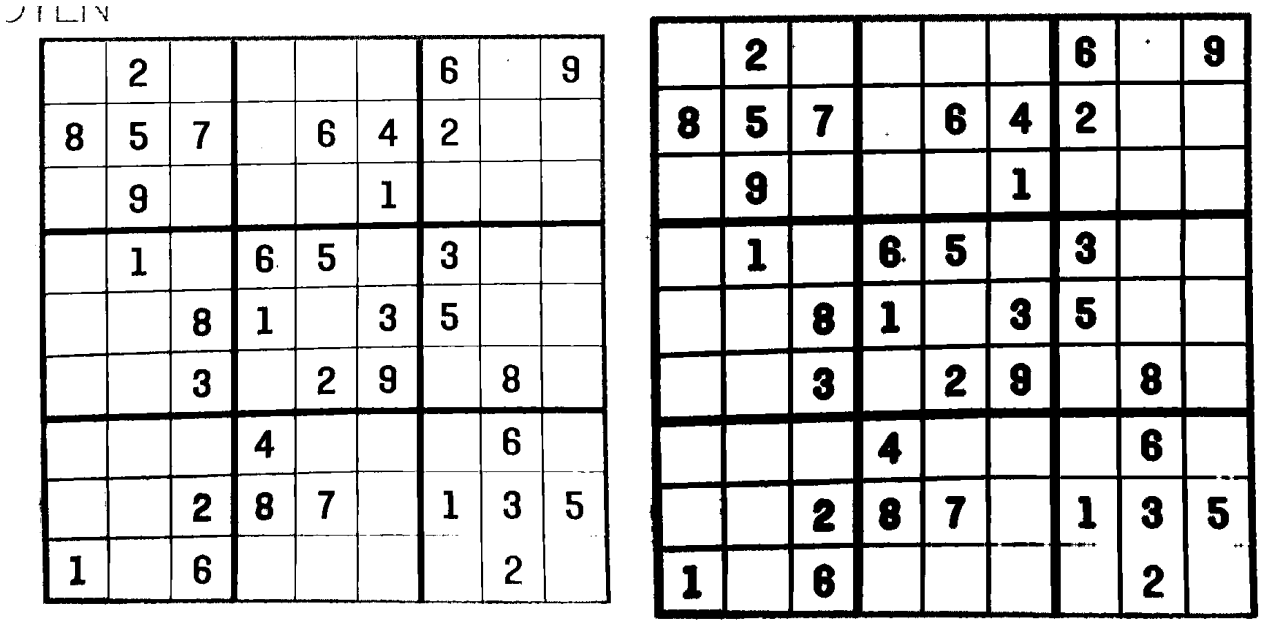


FIGURE 22 – Exemple de dilatation des lignes

11 Réseau de neurones

11.1 Qu'est-ce qu'un réseau de neurones ?

Un neurone est une représentation mathématique et informatique du neurone biologique. Il reproduit certaines de ses caractéristiques biologiques, en particulier les dendrites, axones et synapses, au moyen de fonctions et de valeurs numériques. Un neurone élémentaire peut manipuler des valeurs binaires, représentées par 0 et 1, ou réelles.

L'ensemble que forme l'agencement de ces neurones inter-connectés est un réseau neuronal capable d'apprendre et de se modifier en fonction de cet apprentissage. Il est à sens unique et est constitué de couches de neurones dans lesquelles la sortie d'un neurone ne peut seulement alimenter l'entrée d'un neurone situé dans la couche postérieure.

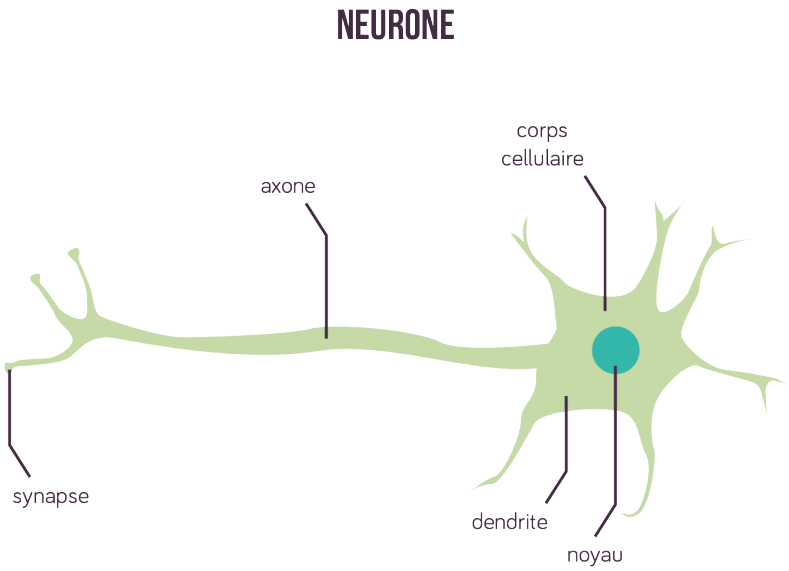


FIGURE 23 – Schéma d'un neurone en biologie

11.2 L'architecture des réseaux de neurones

Les réseaux de neurones sont constitués de plusieurs couches (au minimum 2), elles même construites par le rassemblement d'un ou plusieurs neurones. Ces couchent portent un nom en fonction de leur place dans le réseau.

- **Couche d'entrée :** les neurones de cette couche contiendront les informations / données que l'on souhaite tester

- **Une ou plusieurs couches cachées :** les neurones contenus dans cette couche entrent dans un processus de calcul complexe. Leurs valeurs sont seulement exploitées par le réseau de neurone et participe ainsi au bon apprentissage de celui-ci.

- **Couche de sortie :** les neurones de sortie nous donneront le résultat au problème présenté en couche d'entrée.

Les neurones qui composent une couche sont reliés à l'ensemble des neurones de la couche qui suit, sauf dans le cas de la couche de sortie qui marque la fin du réseau. Il peut arriver que les neurones d'entré soient connectés à la couche de sortie bien qu'il y ait des couches cachées entre celles-ci, dans le but d'améliorer l'apprentissage.

Chaque connexion entre deux neurones possède un poids qui permet de faire les différents calculs de propagation qui interviennent dans les réseaux neuronaux.

De plus, chaque neurone possède une valeur de sortie (comprise entre 0 et 1) qui est la valeur à tester pour la couche d'entrée et le résultat d'un calcul pour les autres couches, cette valeur correspond à l'information que chaque neurone transmet aux neurones suivants.

11.3 Implémentation de l'architecture

Pour la création de l'architecture complète du réseau neuronal, nous avons tout d'abord créé la structure du neurone dans un fichier à part puis de même pour celle correspondante aux couches du réseau. L'ensemble de ces nouvelles structures sont ensuite rassemblées dans un nouveau fichier en fonction des quantités et proportions demandées par l'utilisateur du réseau avec notamment l'initialisation des poids entre chaque neurone de couches adjacentes.

11.4 Apprentissage

Le but de l'apprentissage est de permettre au réseau de neurones de s'améliorer en apprenant littéralement de ses erreurs. Cet apprentissage se réalise à travers plusieurs étapes afin d'obtenir de nouvelles valeurs de poids que l'on pourra mettre à jour dans l'ensemble du réseau. En somme, cela consiste simplement à modifier les poids de chaque connexion.

Plus les cycles d'apprentissage sont nombreux, plus les poids des connexions entre les neurones convergent vers leur valeur optimale lors du travail du réseau neuronal. Cela le mène donc à un améliorer ses taux de réussite face à la résolution de problèmes auxquels il se forme.

Le pas d'apprentissage est une valeur qui permet de régler la vitesse à laquelle le réseau apprend mais aussi la précision de cet apprentissage. Plus le pas sera grand, plus l'apprentissage sera rapide, moins il sera précis, et inversement.

11.5 La propagation vers l'avant (de l'entrée vers la sortie)

L'algorithme de calcul en avant permet de trouver la(les) valeur(s) de sortie du réseau de neurones en fonction de la(les) valeur(s) d'entrée en parcourant le réseau du début vers sa fin et en appliquant les différents calculs que nous allons définir.

La première étape est de calculer une valeur d'activation pour chacun des neurones autrement appelé le biais. Ce biais est une fonction de combinaison elle-même jointe à une fonction d'activation. La fonction de combinaison retourne une valeur calculée à partir des différentes entrées et des poids qui y sont associés.

Cette valeur est ensuite passée en argument à la fonction d'activation dont le résultat sera la sortie finale du neurone. Nous avons choisi d'utiliser la fonction sigmoïde pour la couche cachée, et la fonction softmax pour la couche de sortie (seulement pour l'IA de l'OCR), deux fonctions qui ont la particularité de donner des images entre 0 et 1 avec une progression à partir de petits débuts qui s'accélère et se rapproche d'un point culminant à la fin de l'intervalle.

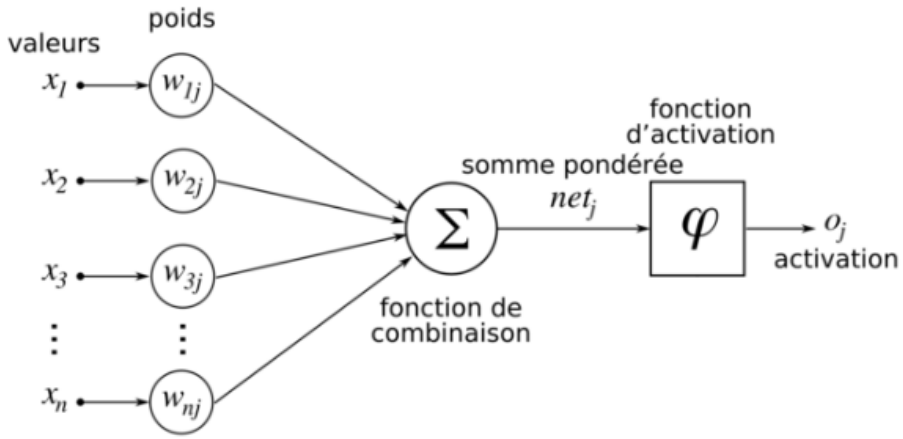


FIGURE 24 – Propagation avant dans le réseau

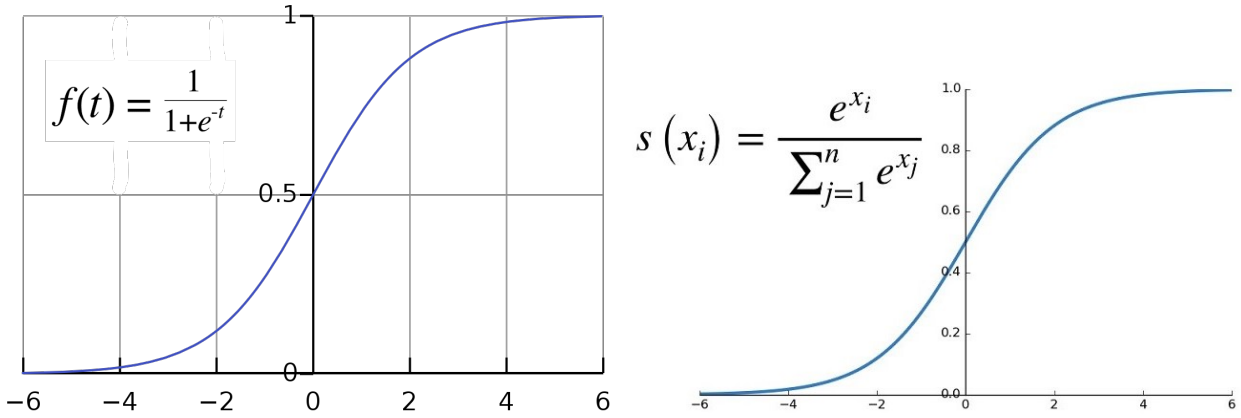


FIGURE 25 – Fonctions d'activations Sigmoïde et Softmax

Comme on peut le remarquer, la courbe de ces deux fonctions sont similaire, mais elle n'ont pas les mêmes propriétés pour autant. En effet, la fonction sigmoïde est utilisée pour la régression logistique à deux classes, tandis que la fonction softmax est utilisée pour la régression logistique multiclasse, elle est donc plus adaptée calculer la probabilité de sortie d'évènements qui ne sont pas binaires.

11.6 La rétropropagation du gradient (de la sortie vers l'entrée)

Cet algorithme en particulier qui intervient dans l'apprentissage. Il permet de modifier les poids des neurones pour devenir plus précis. Appliquer cet algorithme nécessite la connaissance de la valeur de sortie que l'on récupère après l'essai de certaine valeur en entrée. Cette valeur récupérée est ensuite comparée au résultat que l'on devrait obtenir pour les mêmes valeurs en entrée. Que les valeurs comparées soient égales ou non, le processus s'opère donc en quatre étapes distinctes.

- Tout d'abord il faut effectuer une propagation en avant dans le réseau neuronal pour les entrées de chaque essai et stocker ce que renvoi le réseau à chacun des tests.

- Ensuite par un calcul, on récupère les valeurs de sortie du réseau neuronal que l'on injecte dans la dérivée des fonctions sigmoïdes et softmax (qui sont les mêmes). Ensuite, on récupère la valeur d'erreur représentée par la différence entre les résultats collectés et les résultats que l'on devrait normalement obtenir, l'ensemble multiplié par la dérivée précédemment récupérée (valeurs théoriques).
La fonction sigmoïde est utilisée pour la régression logistique à deux classes, tandis que la fonction softmax est utilisée pour la régression logistique multiclass.

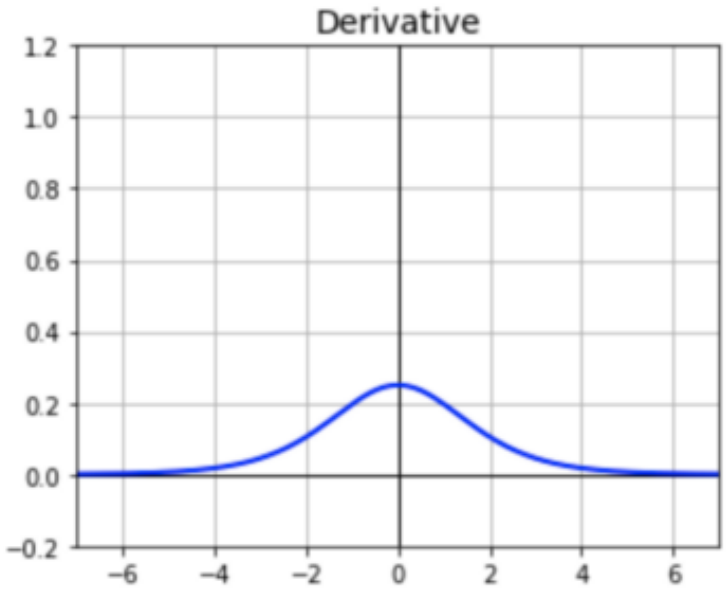


FIGURE 26 – Dérivée des fonctions sigmoïde et softmax

- L'algorithme met à jour le poids de chaque connexion en partant de la dernière couche de neurones vers la première. Cet algorithme remonte l'ensemble du réseau d'où le nom rétropropagation. Les poids synaptiques qui contribuent plus à une erreur seront modifiés de manière plus importante que les poids qui provoquent une erreur marginale.

- Afin que l'apprentissage soit efficace, il faut répéter cet algorithme un grand nombre de fois afin que le réseau finisse par toujours avoir une bonne réponse (pour un pas peu élevé, il faut compter entre 1000 et 10000 générations pour une simple porte logique).

11.7 Réseau de neurones : Porte XOR et portes logiques

Pour notre réseau de neurones capable d'apprendre la porte XOR, nous utilisons un réseau de la forme 2-2-1 c'est-à-dire 2 neurones d'entrée dans la première couche, 2 neurones dans la seule couche cachée que comporte notre réseau et un seul neurone pour la couche de sortie. Pour faire fonctionner ce réseau correctement, il est ensuite possible de choisir les entrées possibles avec les sorties correspondantes (ici 4 voir la table de vérité).

Table de vérité XOR (OU exclusif)		
A	B	S
0	0	0
1	0	1
0	1	1
1	1	0

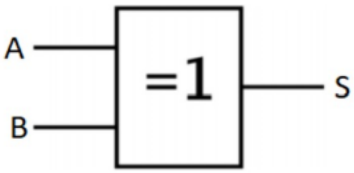


FIGURE 27 – Table de vérité du XOR et sa représentation

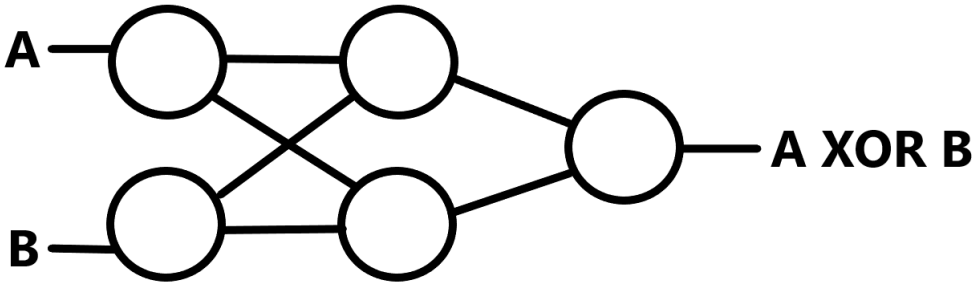


FIGURE 28 – Représentation de la construction neuronale de la porte XOR

Nous avons choisi un petit pas (pas = 0.15) pour l'apprentissage de notre réseau neuronal, ce qui nécessite un grand nombre de générations pour arriver à de bons résultats, environ 10000 pour une efficacité complète. Comme il est possible de le remarquer aux premières générations, le résultat renvoyé pour les 4 possibilités sont très souvent fausses. Notons aussi que nous utilisons la fonction d'activation sigmoïde dans l'ensemble du réseau car la solution est seulement binaire.

Les poids du réseau après apprentissage sera sauvegardé dans un fichier nommé « weights.txt ».

Il est ensuite possible de demander à tester des valeurs, cette fois non pas dans un but d'apprentissage, pour vérifier que la sortie renvoie le bon résultat.

11.8 Réseau de neurones : Détection de caractères

Une fois, les chiffres définis, l'IA commence à comparer ces derniers avec un groupe d'images déjà traitées par l'OCR ou prédéfinies qui lui permettent de proposer des hypothèses sur la signification de ces caractères. L'IA se base sur ces hypothèses afin d'analyser les courbures des lignes en chiffres et du vide en image sans chiffre. Après avoir passé en revue toutes les hypothèses, le programme propose au final le contenu susceptible d'être conforme à ce qui est représenté dans l'image de début.

11.9 Convertion des images

Chaque ligne de pixels des images est convertie une à une en 1 si le pixel est noir et 0 sinon dans une liste de la taille de l'image 28x28 = 784. Chaque élément de la liste sera ensuite donné en entrée à chaque neurone de la couche d'entrée de notre IA pour procéder aux traitements et / ou à l'apprentissage.

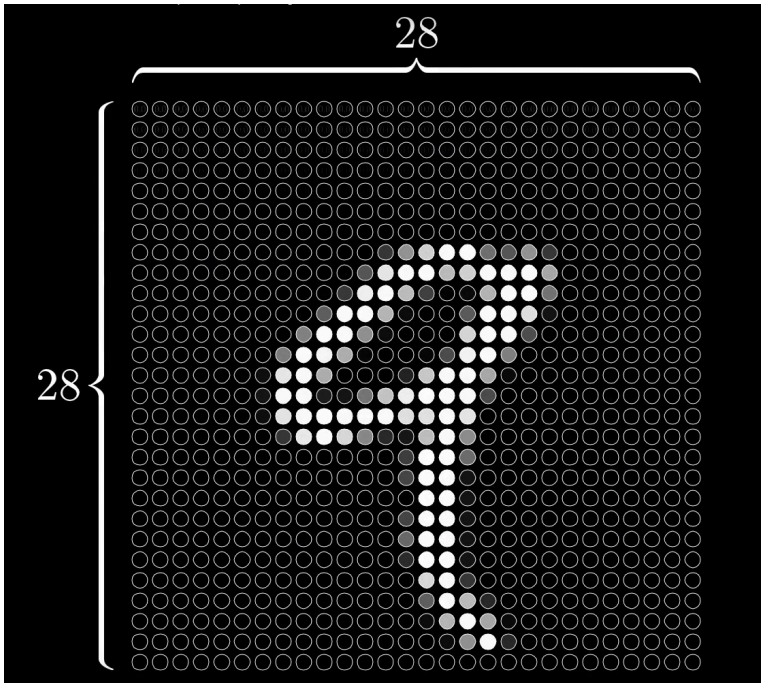


FIGURE 29 – Convertion d'une image représentant un 9 en liste

11.10 Architecture

Pour un traitement d'image optimal, nous avons choisi d'utiliser une structure 784-20-10. C'est à dire que nous avons 784 neurones dans la couche d'entrée qui prendront les valeurs des pixels de l'image ligne par ligne (déjà précisé dans la partie "Convertion des images").
Ensuite, 20 neurones sont disposés dans la couche cachée et sont responsable de la reconnaissance des courbures / formes types de chacun des numéros qui sont à l'image.

Enfin, pour la dernière couche, 10 neurones sont créés et renvoient la probabilité de sortie de chacun des 10 chiffres possible. Il nous suffit donc de récupérer la probabilité la plus élevée, qui sera selon l'IA la solution : le chiffre présent sur l'image.

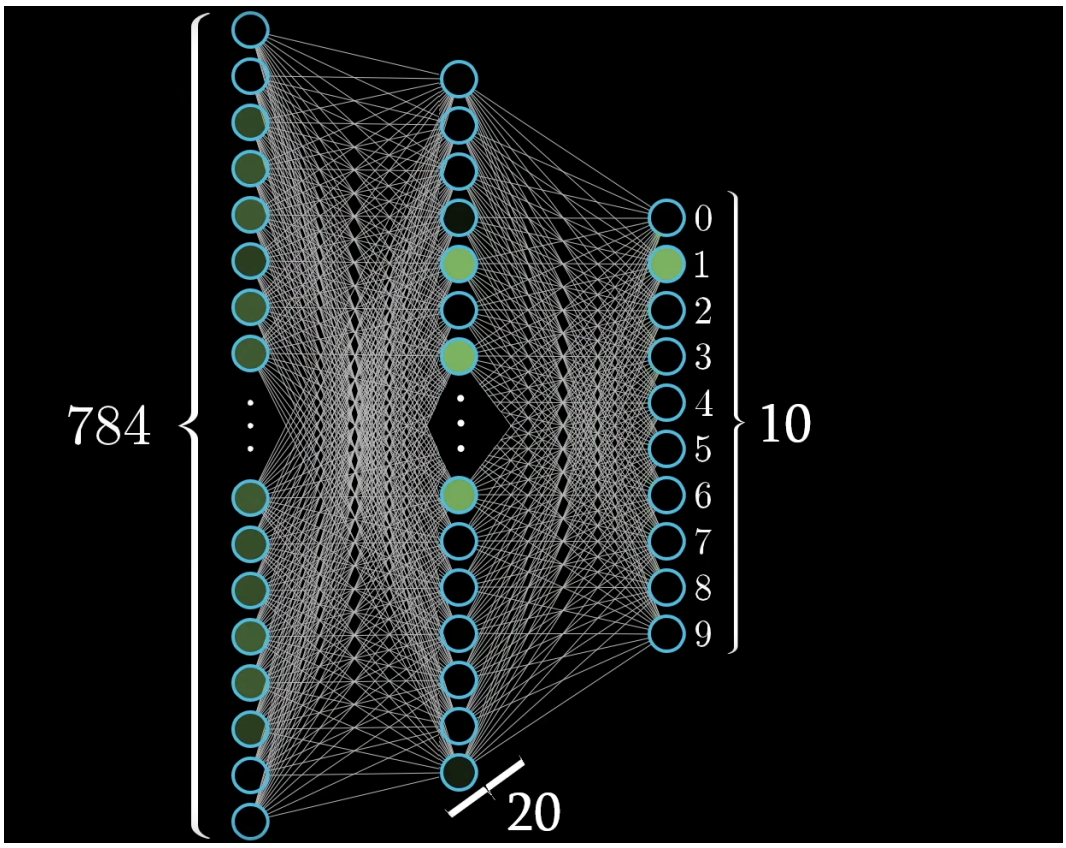


FIGURE 30 – Architecture du réseau neuronal de notre OCR

11.11 Détection des images

La structure de l'IA, elle seule, ne suffira jamais à reconnaître l'ensemble des chiffres une fois celle-ci initialisée. En effet, il faut qu'elle passe par un apprentissage intensif. C'est pour cette raison que nous avons utilisé une banque d'image provenant de l'ensemble des cases découpées d'un Sudoku par notre prétraitement. En effet, après un grand nombre de génération, elle a été capable de détecter les chiffres avec une très grande précision.

```
Epoch 1000   | Position Found = 0 (Output: 0.998954); Expected 0 OK
             |-> Error = 0.000001
             | Position Found = 1 (Output: 0.998099); Expected 1 OK
             |-> Error = 0.000003
             | Position Found = 2 (Output: 0.998366); Expected 2 OK
             |-> Error = 0.000002
             | Position Found = 3 (Output: 0.998842); Expected 3 OK
             |-> Error = 0.000001
             | Position Found = 4 (Output: 0.997152); Expected 4 OK
             |-> Error = 0.000008
             | Position Found = 5 (Output: 0.996508); Expected 5 OK
             |-> Error = 0.000011
             | Position Found = 6 (Output: 0.999822); Expected 6 OK
             |-> Error = 0.000000
             | Position Found = 7 (Output: 0.999974); Expected 7 OK
             |-> Error = 0.000000
             | Position Found = 8 (Output: 0.999135); Expected 8 OK
             |-> Error = 0.000001
             | Position Found = 9 (Output: 0.999825); Expected 9 OK
             |-> Error = 0.000000
-----> ErrorMax: 0.000011
```

FIGURE 31 – Apprentissage de notre IA - Génération 1000

12 Le Sudoku

12.1 Résolution du Sudoku

Contrairement à l'être humain, avec un ordinateur on peut essayer toutes les combinaisons possibles pour un Sudoku et voir si cette dernière est valide. Aussi appelé méthode par force brute où méthode de "backtracking", l'algorithme que nous avons décidé d'implémenter repose sur 4 étapes.
Premièrement on vérifie la validité du fichier grid_00, on le parse pour le convertir en tableau de caractère que l'on peut facilement manipuler en C.

Deuxièmement, on essaye sur chaque case en partant de la plus en haut à gauche puis de la droite vers la gauche de mettre la valeur 1. Si la valeur est déjà présente dans la ligne/colonne/carré alors on essaye avec 2 puis 3 etc, si on ne peut mettre aucune valeur alors notre grille est impossible à résoudre, il faut donc retirer les chiffres que nous avons entrée précédemment et recommencer avec le chiffre suivant, c'est pour cela que caractérisons l'algorithme de "backtracking", c'est la troisième étape.

Comme vous l'avez compris, nous allons faire des appels récursifs tant que nous n'avons pas trouver la combinaison qui nous fait remplir entièrement le tableau.

La complexité d'un tel algorithme est O(9^m) avec m le nombre de casses vides or ce n'est pas un problème car la pire des situations ne se produit presque jamais.

12.2 Sauvegarde de la grille

C'est la sauvegarde qui vient cloturer la méthode de résolution du Sudoku, nous l'enregistrons sous la forme la forme du cahier des charges dans le fichier grid_00.result, qui est identique au format du fichier initial grid_00.

12.3 Création de l'image finale

Une fois le sudoku résolu, nous voulez bien évidemment afficher le résultat à l'utilisateur plutôt que de le laisser chercher dans le fichier grid_00.result, c'est pourquoi nous réalisons la reconstruction de la grille à partir du fichier grid_00 et grid_00.result.

En effet, dans un fichier du projet, nous avons une grille complètement vide de 1000x1000 pixels, il y a seulement les contours noirs délimitant la grille, par la suite on converti le fichier grid_00 en un tableau de 81 éléments (charactère) si le caractère est un . alors on n'ajoute rien aux coordonnées de la première case, on passe donc au deuxième éléments s'il s'agit d'un chiffre par exemple un "2" alors on va chercher dans un dossier un "2" déjà découper pour le superposer sur la grille vide aux coordonnées x=122 et y = 16, qui correspondent à la deuxième case. On réalise cette étape 81 fois soit 9x9 afin de compléter la grille.

Ainsi on obtient un premier résultat comme ci dessous. Les chiffres présents dans grid_00 sont les chiffres reconnu par IA c'est à dire les chiffres déjà présent sur la grille de l'utilisateur par soucis de clarté nous les affichons en noir.

Ensuite nous réalisations la même chose sur le fichier grid_00.result cependant, sauf si la grille est insolvable on ne doit pas rester de case vide dans l'image de sortie. De plus, pour ne pas écraser de les

chiffres noirs on test avant si on avait pas déjà mis un chiffre lors du premier parcours, si c'est le cas alors on passe au chiffre suivant et si ce n'est pas le cas on ajoute le chiffre en vert cette fois ci et passe également au chiffre suivant jusqu'à arriver à la fin de la grille.

Et voilà, notre image de sortie est créé, on a également réaliser une copie de l'image avant de faire le deuxième parcours pour afficher à l'utilisateur s'il le souhaite via le mode manuel le grille reconnu par IA et donc avant résolution. Si l'utilisateur est en mode automatique alors on affiche une fenêtre avec l'image ainsi créée et si l'utilisateur est en mode manuel dans ce cas on affiche une boite d'outils lui permettant de choisir l'image créer lors d'une étape intermédiaire comme par exemple la réduction du bruit ou alors celle créer lors de la binarisation ou alors de la découpe de l'image, etc, etc.

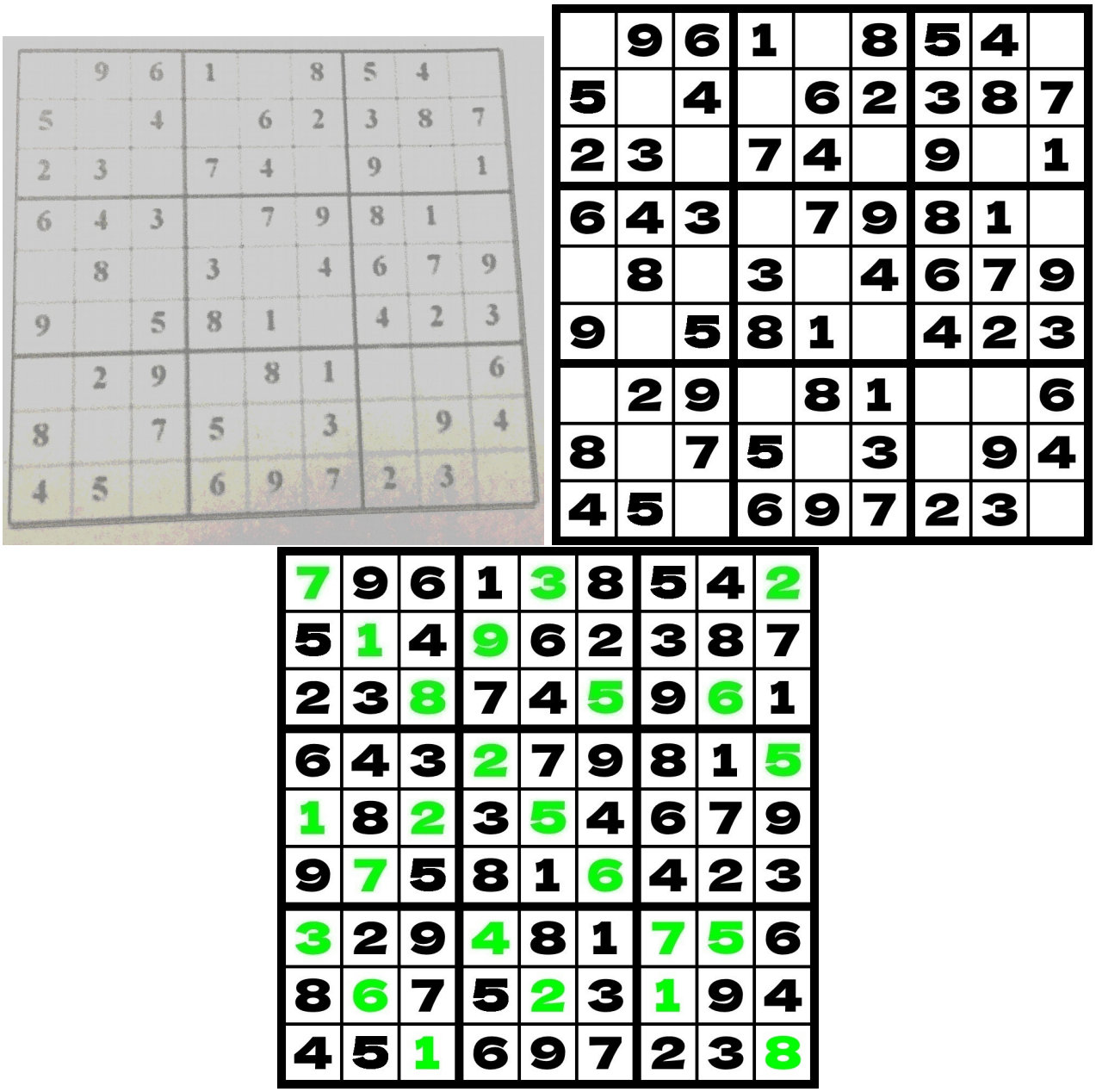


FIGURE 32 – Création des la grille finale sur l'image_06

13 BONUS - Le Site Web

Le site Web de notre projet est avant tout le site Web de notre projet (OCR - LMP). Ceux-ci sont respectivement représentés au travers de différents articles ainsi qu'un lien de téléchargement et autres interactions.

Voici le lien pour aller le consulter, **https ://topagrume.github.io/OCR_LMP/accueil.html**

13.1 La structure du site :

Il a tout d'abord fallu réfléchir à la structure du site, son arborescence, l'organisation de chacune des pages, la répartition des images, du texte, des liens, d'un historique de projet, de texte et de statistiques.

Nous avons décidé de choisir une structure simple composée d'un en-tête dans lequel on retrouve le nom de notre groupe, le nom et le lien vers les différentes pages du site ainsi qu'un bouton « télécharger » (Accueil, Documentation), puis dans la suite de la structure un corps de texte qui laisse libre recours à nos choix et enfin un pied de page destiné à n'être qu'un simple décor.

13.2 Le contenu du site :

13.2.1 La page d'accueil :

Bien en évidence, nous avons opté pour directement placer un bouton afin de renvoyer vers un lien de téléchargement du jeu fonctionnel. Dans le corps de la page, nous avons opté pour une présentation brève de notre groupe LMP répartie en différents onglets mis en parallèle avec des images représentatives, puis de nos valeurs en tant que développeur, une présentation brève de ce qu'est un OCR et de son fonctionnement et enfin un petit paragraphe sur notre politique ainsi que le choix de notre Logo.

Plus bas, il nous a semblé important présenter les différents membres du projet individuellement avec notamment leurs domaines de travail au sein de notre OCR. Les liens vers le Twitter et le GitHub de la majorité des membres est aussi renseigné sous chaque personne.

Enfin en pied de page, nous avons implémenté des statiques à titre purement informatif et esthétique sur le nombre de nos push, d'étapes réussies, de projets réussis et enfin le nombre d'heures qui ont été dédiées au projet dans son ensemble.

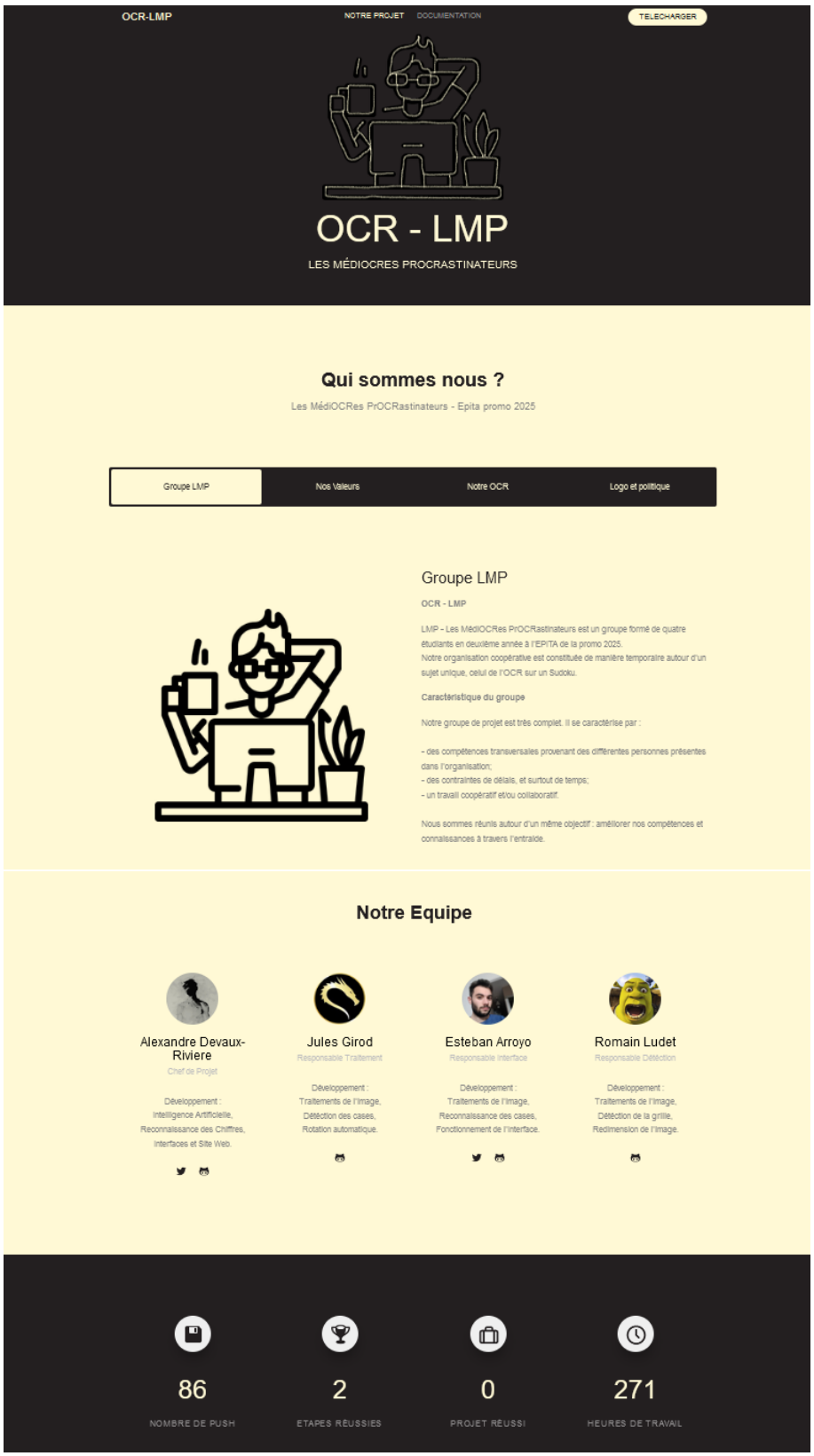


FIGURE 33 – Page d'accueil de notre site (partie 1)

13.2.2 La page des documents :

Ici nous avons rassemblé l'ensemble des documents qui nous ont été demandés durant l'avancement de notre projet à savoir notre première et deuxième soutenance. Elles sont présentées avec leurs différentes caractéristiques avec un bouton pour les télécharger.

Dans la lignée des présentations, nous avons rassemblé l'ensemble des logiciels et des outils qui nous ont permis de faire aboutir notre projet.

En pied de page nous avons simplement fait le choix de placer le logo de notre équipe.

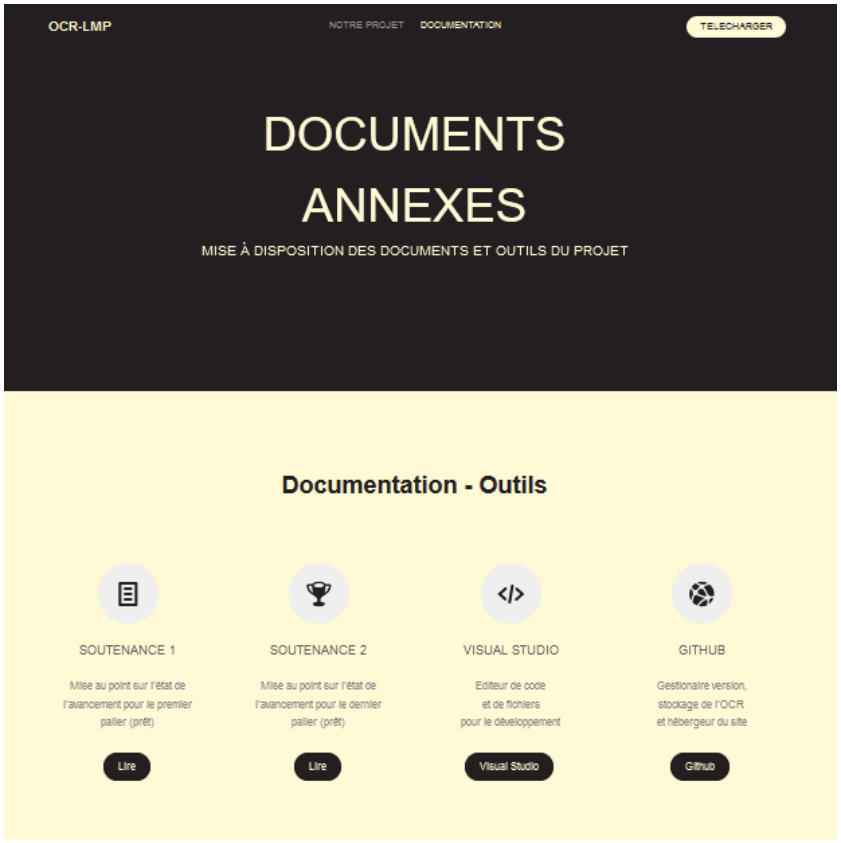


FIGURE 34 – Page de la documentation (partie 2)

14 Difficultés rencontrées

— Coder Hough line
— Découper de manière propre les cases du Sudoku
— Corriger l'appartion des "NaN" dans le réseau neuronal
— Trouver les bonnes constantes de pas dans l'IA

15 Ressenti personnel - fin de projet

Alexandre Devaux-Riviere :

Notre projet est fini. Un gros travail a été fait au niveau des procédés de traitements de l'image, de l'intelligence artificielle et surtout au niveau de la détection de la grille associée à la séparation de l'image.
Tout au long de notre projet, la cohésion était forte, malgré certains désaccords (normal dans un projet),au sein de notre groupe. Nous avions une bonne organisation, ce qui a pu nous soulager, surtout lorsque les périodes étaient chargées.
Personnellement, je considère ce projet comme une expérience très enrichissante que ce soit au niveau des notions de travail en groupe tout comme pour les connaissances en C que nous avons pu acquérir au cours de cette courte période.

Jules GIROD :

La fin du semestre 3 annonce également la fin du projet. Ce fût un immense plaisir de faire ce projet, comme dit précédent j'avais quelques petites connaissances en traitement d'image mais concrètement ce projet m'a permis de découvrir de nouvelle méthode de traitement d'image tel que "Otsu" mais aussi "composante connexe". J'ai également découvert les limitations du C ainsi que c'est message d'erreur pas très explicite comme par exemple : "Double free or Corruption" ou alors "Segmentation fault" quand on atteint la maximum de récursion.
'ai également beaucoup appris que la documentation était très importante avant de faire une fonction car on peut vite se retrouver à faire un truc qui existe déjà dans une librairie ou un logiciel comme ImageMagick qui nous fait gagner beaucoup de temps.
Pour conclure, ce projet sur un semestre m'a beaucoup instruit et faire découvrir deux nouvelles applications à la programme qui sont le traitement d'image et l'intelligence artificielle.

Esteban Arroyo :

Nous sommes à la fin de notre projet et la détection, l'IA et l'interface fonctionnent, nous pouvons être fier du résultat que nous avons obtenu. De plus malgré mes faibles expériences en C, en SDL et en GTK, le travail de groupe nous permet de s'entraider, de mieux comprendre les subtilités du C et ainsi avancer plus vite. Pour finir, je suis heureux de la bonne entente du groupe, car cela nous permet d'avancer sur de bonnes bases de travail. Pour finir, cette expérience m'a permis d'acquérir de nouvelles compétences en C qui me seront très utilisé pour mes années à venir.

Romain Ludet :

Ayant vu la présentation de ce projet par un étudiant lors d'une journée portes ouvertes de l'Epita, j'avais très envie de le réaliser à mon tour ! Le moins que l'on puisse dire, c'est que je ne suis pas déçu. C'est un super projet, très intéressant et il nous pousse à réfléchir et à nous documenter (la base en programmation). Je n'avais jamais utilisé auparavant la librairie SDL, mais j'avais de bonnes bases en C, donc je me suis rapidement adapté. D'autant plus en étant en vocal sur Discord avec un super groupe, notamment le soir ! Nous avons terminé notre projet et nous sommes fier de vous le présenter. Ce fut un réel challenge que nous avons relevé selon moi avec succès. De plus, j'ai beaucoup aimé travailler sur ce projet en groupe grâce à une bonne organisation et une bonne répartition des tâches de travail.

16 Nous contacter

— Alexandre Devaux-Rivière : alexandre.devaux-riviere@epita.fr
— Jules GIROD : jules.girod@epita.fr
— Esteban Arroyo: esteban.arroyo@epita.fr
— Romain ludet : romain.ludet@epita.fr