

OCR - Rapport de Soutenance Final

Groupe : L.Tesler&Co



Thomas Petel

Théophile Samanos

Etienne Lazar

Tables des matières (Provisoire)

1. Introduction

1.1 Contexte du Projet

1.2 Le groupe

1.3 Répartitions des tâches

1.3 Structure du Rapport

2. Avancement du Projet

2.1 Prétraitemet des Images

2.1.1 Redressement Manuel et Automatique

2.1.2 Élimination des Bruits et Parasites, lissage

2.1.3 Passage en Nuances de Gris

2.1.4 Renforcement des Contrastes

2.2 Détection de la grille

2.2.1 Filtre de Sobel

2.2.2 Détection des lignes par la transformée de Hough

2.2.3 Détection des cases dans la grille

2.3 Reconnaissance de Caractères (OCR)

2.3.1 Conception du Réseau de Neurones

2.3.2 Entraînement du Réseau de Neurones

2.4 Résolution de Sudoku

2.4.1 Algorithme de Résolution

2.5 Interface Graphique

2.5.1 éléments de l'interface

2.5.2 Backend

3. Perspectives et Défis Futurs

3.1 Améliorations Possibles

3.2 Défis Envisagés

4. Conclusion

4.1 Bilan du Projet

4.2 Apprentissages Tirés

4.3 Remerciements

5. Annexes

5.1 Code Source

5.2 Exemples d'Images Traitées

5.3 Documentation Supplémentaire

1- Introduction

1.1-Contexte du Projet

L'objectif de ce projet était de réaliser un logiciel de type OCR (Optical Character Recognition) qui résout une grille de sudoku.

Nous devions réaliser une application prenant en entrée une image représentant une grille de sudoku et qui affichait en sortie la grille résolue. Une interface graphique a dû également être réalisée, 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 de sortie devait également pouvoir être sauvegardée.

De plus, notre application devait posséder un aspect “apprentissage” qui pourrait être séparé de la partie principale. Ce dernier permettrait d’entraîner notre réseau de neurones. Le résultat de cet apprentissage aurait par la suite été sauvegardé et rechargé.

1.2- Le Groupe

Thomas Petel, chef de projet :

Cette expérience en tant que chef de projet pour notre OCR Sudoku Solver a été enrichissante et formatrice.

Depuis le rapport de mi-projet, notre groupe a pu faire des avancées significatives concernant le développement de notre application. Nous avons également rencontré certains problèmes qu'il a fallu surmonter. Cela nous a permis de mieux comprendre la procédure à suivre afin d'assurer le meilleur fonctionnement possible de l'application.

Le sujet original a continué de grandement m'intéresser. Il offrait en effet une perspective par rapport aux travaux précédents en S2.

En outre, ce projet a été un véritable socle permettant une évolution de mes compétences, notamment en ce qui concerne le leadership et ma compréhension des nuances de la réalisation de projets complexes.

Malgré quelques regrets quant à certains aspects de ce projet, je suis fier des progrès que nous avons pu réalisés jusqu'à présent. Je reste heureux à l'idée de présenter les résultats de notre travail lors de cette soutenance finale.

Théophile Samanos,

Je suis satisfait de ce projet, et de ce que j'ai pu produire pour celui-ci. Les implémentations que j'ai réalisé, tels que le traitement d'image, l'affichage, ou la structure du réseau de neurone avec sa partie calculatoire, sont fonctionnelles. Ce projet a permis des applications concrètes de ce que nous avions vu en TP. La motivation était donc bien supérieure que lorsqu'il nous fallait faire de simples exercices. J'ai beaucoup apprécié la création du réseau de neurones. Par le passé, je m'étais déjà intéressé au domaine de l'intelligence artificielle, à travers une IA de puissance 4, et pouvoir continuer et approfondir ce domaine était très captivant. J'ai pu y découvrir de nouvelles approches, notamment lorsqu'il a fallu l'entraîner. J'ai pu découvrir les réels façons de corriger un réseau. En effet, j'appliquais un algorithme génétique, donc l'entraînement s'approche plus d'une sélection du meilleur algorithme, que d'un réel entraînement.

Etienne Lazaro,

Ce périple a connu des hauts et des bas. La réalisation de notre OCR Sudoku Solver a été une expérience formatrice et riche en apprentissage malgré de nombreuses difficultés rencontrées. Cependant, il convient de rappeler que notre groupe a aussi connu des avancées significatives pendant la conception de notre projet. Je suis moi-même fier d'annoncer que ce projet a été une réussite, car durant cette épreuve, nous avons affiné nos compétences en programmation, communication et présentation.

1.3- Répartitions des tâches

Membres Tâches	Thomas	Théophile	Etienne
Gestion de l'image			
Rotation		Dev*	
Transformée de Hough	Dev		Dev
Segmentation			Dev
Détection des carrés		Dev	Dev
Traitement de l'image	Dev	Dev	
Reconnaissance des chiffres			
Réseau de neurones		Dev	
Entrainement du Réseau de neurones	Dev		
Banque de données	Dev	Dev	
Sudoku			
Solver			Dev

*Développeur

1.4- Structure du Rapport

Ce rapport détaille l'état d'avancement de notre projet pour la soutenance visant à présenter le rendu final. Nous présentons ici l'ensemble de notre travail, mettant en lumière les réalisations accomplies et les défis rencontrés au cours du développement. Chaque section offre des explications détaillées sur nos progrès, démontrant ce qui a été accompli jusqu'à présent, ainsi que les éventuelles difficultés surmontées. Vous trouverez dans ces pages une analyse approfondie de notre démarche, des choix effectués et des étapes clés franchies dans le cadre de la réalisation de notre projet OCR Sudoku Solver.

2-Avancement du projet

2.1- Prétraitement des Images

2.1.1- Redressement Manuel et Automatique

Parmi les traitements d'image à réaliser, la rotation de l'image est une étape qui permet au réseau d'être bien plus performant.

Il y aura deux utilisations de la rotation de l'image : une manuelle et une automatique. Nous détaillerons ici la rotation manuelle, car l'application automatique fait appelle à d'autre algorithme, tel que la méthode de Hough, que nous vous détaillerons plus bas.

En l'absence d'interface graphique, la rotation manuelle se fait pour l'instant en ligne de commande.

Nous pouvons effectuer une rotation d'autant de degré souhaité. La rotation s'effectue autour du centre de l'image.

Même si ce n'est pas une phase très complexe comparé à certains autres corps du projet, tel que le réseau de neurone ou la détection des bords, nous avons tout de même rencontré des difficultés.

Tout d'abord, un problème de placement de pixels. En effet, la rotation se basant sur des valeurs de cosinus et sinus, la valeur retournée par l'équation de rotation ne couvrait pas l'ensemble des pixels de l'image retournée. Il arrivait donc que l'image soit couverte de pixels noirs, parfois même à la hauteur de 20% ! Nous avons donc cherché à camoufler ce problème, notamment en re-parcourant l'image à la recherche de pixels noirs, afin de l'ajuster en la moyenne de la couleur de leur pixels voisins. Le problème est que, en plus d'ajouter un parcours de l'image, et donc d'augmenter le temps de traitement, cela ne reflète pas vraiment l'image, il permet de camoufler les défauts aux yeux de l'œil humain. Or, c'est un réseau de neurones qui cherchera la valeur des chiffres, et ajouter des pixels aux couleurs fantôme réduira sa capacité de reconnaissance, et donc augmentera drastiquement le temps d'apprentissage.

Pour pallier ce problème, nous avons inversé notre façon de construire l'image tournée : au lieu de parcourir l'image donnée, puis de placer le pixels étudié sur une autre image, nous allons maintenant parcourir l'image à envoyer, et chercher quel pixels doit se trouver à cette position. Cela permet d'attribuer à chaque pixels de la nouvelle image un couleur présente sur l'image d'origine. Les pixels ne pouvant pas être placés (les pixels que l'on cherche à l'extérieur de l'image d'origine par exemple) restent alors à leur couleur par défaut, c'est-à-dire gris.

La fonction de rotation n'est pas parfaite. Par exemple, les pixels présents aux angles de l'image d'origine sont perdus, car, lors de la rotation, ils sont censés être placés en dehors de la nouvelle image. Une solution pour pallier à ce problème serait de dézoomer l'image de sorte que toute l'image donnée puisse tenir dans la nouvelle image tournée. On s'assure ainsi que toutes les cases de la grille de sudoku donnée restent visibles, quel que soient leur coordonnées.

La rotation automatique appelle évidemment les fonctions de rotations manuelles. Reste donc à connaître l' angle avec lequel il est nécessaire de tourner l'image. Pour cela, nous faisons appel à toutes les fonctions de traitement, afin de pouvoir appliquer la méthode de Hough. Cette méthode, dont nous en parlerons plus en détail dans la partie "Détection des lignes par la transformée de Hough", permet de détecter les différentes lignes présentes sur l'image. Grâce à cela, mais surtout grâce à la trigonométrie, il est aisément de connaître l'angle formé entre la droite et le bord gauche de la fenêtre. Par conséquent, on parvient à récupérer l'angle, et donc d'effectuer la rotation.

Cette approche permet une optimisation. En effet, il est nécessaire d'appliquer les différents traitements d'image afin d'obtenir une image pure et facilement interprétable par la transformée de Hough. Le traitement de l'image est déjà effectué, il n'est donc pas nécessaire d'appliquer une fois de plus toutes ces opérations.

2.1.2 Élimination des Bruits et Parasites, lissage

Dans cette section, nous allons expliquer plus en détail l'ensemble des filtres qui ont été appliqués à nos images ainsi que leur implémentation utilisée dans notre programme de réduction de bruit, `noise_reduction.c`.

Filtre de Contraste :

Le filtre de contraste vise à accentuer les détails de l'image. Pour ce faire, nous utilisons la fonction `image_levels`. Elle divise l'intervalle des couleurs en segments déterminés par la variable `n`, puis attribue à chaque segment une valeur représentative. Ainsi, l'image est étalonnée pour avoir une plus grande variété d'intensités, améliorant ainsi la visibilité des détails. Ensuite, la fonction `invert` est appliquée pour inverser les couleurs, créant un contraste supplémentaire entre les objets et l'arrière-plan.

Filtre Médian :

Le filtre médian est un outil puissant pour atténuer le bruit impulsif dans une image. La fonction `MedianFilter` calcule la médiane des valeurs des pixels environnants pour chaque pixel de l'image. Pour ce faire, elle effectue une copie du voisinage du pixel dans un tableau, trie ce tableau, puis assigne au pixel la valeur médiane. Cela réduit l'impact des valeurs aberrantes, telles que le bruit, tout en préservant les contours et les structures importantes de l'image.

Cette étape est vraiment la base de tout le traitement d'image. En effet, elle permet de supprimer des pixels "en trop", c'est-à-dire les pixels qui représentent une défaillance de l'image. En réduisant ces aberrations, on réduit drastiquement les faux-positifs qui pourraient survenir lors des prochains traitements, notamment lors de la détection des bords. En effet, un pixel, ou groupe de pixels, seuls sur une surface uniforme risquent d'être perçus comme une ligne ou un bord, et cette situation est à éviter à tout pris !

Méthode D'Otsu :

Sans pour autant modifier l'image, la méthode d'Otsu est au cœur du traitement d'image. En effet, pour les prochains traitements, il est nécessaire de passer l'image en noir et blanc. Ce passage est différent du passage en nuance de gris. En effet, on

appel ici à changer toute l'image avec des pixels soit noir, soit blanc. Il n'y a pas de valeurs intermédiaires. L'enjeu est ici de trouver la bonne valeur (le seuil) pour laquelle le fond se détache des objets présents. Les objets seront donc bien visibles, tant pour l'humain que pour la machine.

L'objectif de cette fonction est de trouver la valeur la plus optimale pour séparer les pixels du fond des objets présents.

Filtre Moyen :

Le filtre moyen est un filtre de lissage qui réduit les variations d'intensité dans une image. La fonction `AverageFilter` utilise un noyau de moyenne pondérée pour calculer la nouvelle valeur de chaque pixel. Ce noyau atténue les hautes fréquences de l'image, produisant un effet de flou qui contribue à adoucir les transitions entre les zones d'intensité, réduisant ainsi le bruit de haute fréquence.

Filtre de Seuillage Adaptatif :

Le filtre de seuillage adaptatif ajuste dynamiquement le seuil de binarisation en fonction du niveau de bruit dans l'image. La fonction `adaptiveThreshold` utilise un algorithme basé sur une estimation du niveau de bruit. Si le bruit est élevé, un seuil plus élevé est utilisé pour garantir la suppression du bruit sans perte excessive de détails.

Dilatation :

La dilatation est une opération morphologique visant à agrandir les régions d'intérêt dans une image. La fonction `dilate` parcourt l'image et examine chaque pixel. Si un pixel blanc est entouré de pixels blancs, il est conservé ; sinon, il est transformé en noir. Cette opération lisse les contours de l'image, éliminant ainsi les petites interruptions et rendant les formes plus cohérentes.

Filtre de Gauss :

Le filtre de gauss est utilisé pour réduire le bruit et lisser l'image. Comme toutes les étapes du traitement d'image, c'est un pas nécessaire pour parvenir à obtenir une image compréhensible par le réseau de neurones. Il existe plusieurs façons de l'appliquer, avec des filtres de différentes tailles. Nous utilisons un filtre 3x3. Pour chacun des pixels de l'image, on regarde les pixels autour de celui étudié. On

attribue ensuite une importance à chacun des pixels, afin de pouvoir les sommer. Cette importance est matérialisée par une valeur plus ou moins forte lors du calcul de la somme. Cette somme est ensuite divisée par la somme des poids. La valeur obtenue est donc la nouvelle valeur du pixels.

L'application du filtre de gauss a permis de diminuer le bruit de l'image, et d'obtenir une image plus homogène.

Inversion d'Image :

L'inversion d'image est réalisée par la fonction `NegativePictureIfNormal`. Elle vérifie si l'image est majoritairement composée de pixels blancs ou noirs. Si les pixels blancs prédominent, l'image est inversée. Cette opération permet d'ajuster l'apparence de l'image en fonction de ses caractéristiques globales.

Chaque filtre a été soigneusement sélectionné pour ses propriétés spécifiques et leur ordre d'application a été ajusté pour maximiser l'amélioration de l'image tout en minimisant la perte de détails. L'ensemble de ces filtres forme un processus de prétraitement cohérent, adaptatif et efficace pour le traitement d'images.

Pour concevoir ces filtres, nous avons eu besoin de mettre au point le fichier `verbose.c` qui joue un rôle essentiel en facilitant la communication et la gestion de l'interface utilisateur. Il fournit un mécanisme de gestion de la verbosité, permettant d'ajuster le niveau de détail des messages de sortie du programme. Les fonctions définies dans ce fichier, telles que `saveVerbose`, `changeImageGUI`, et `printVerbose`, sont conçues pour afficher des informations significatives à différentes étapes du traitement, que ce soit via la console ou une interface utilisateur graphique (GUI).

La fonction `saveVerbose` gère la sauvegarde d'une image et la libération de la mémoire associée. Elle affiche des messages informatifs sur la console si le niveau de verbosité est activé, tout en évitant l'affichage excessif lorsqu'il n'est pas nécessaire. La fonction `changeImageGUI` permet de mettre à jour l'interface graphique en affichant l'image sélectionnée et en ajustant la barre de progression. Enfin, la fonction `printVerbose` offre une option pour afficher des messages détaillés sur la console et les transmettre à la GUI, contribuant ainsi à une expérience utilisateur plus détaillée.

En résumé, le fichier `verbose.c` améliore l'expérience utilisateur en fournissant des informations contextuelles sur les opérations en cours, tout en offrant un contrôle sur le niveau de détail des messages générés par le programme de traitement d'image.

2.1.3 Passage en Nuances de Gris

L'une des étapes initiales de notre projet de résolution de Sudoku OCR est la transformation d'une image du Sudoku non résolu en une image en nuances de gris. Cette étape est essentielle pour faciliter le traitement ultérieur de l'image.

Nous utilisons une fonction spécifique appelée `loadPPMImage` pour charger l'image du Sudoku depuis un fichier PPM. Cette fonction lit l'en-tête PPM du fichier, extrait les informations essentielles, telles que la largeur, la hauteur et les valeurs de couleur, et alloue la mémoire nécessaire pour stocker les pixels de l'image.

Cf: Annexe 1

Une fois que l'image est chargée, nous parcourons chaque pixel de l'image et extrayons les informations de couleur. Dans notre structure de données, nous utilisons un type `Pixel` composé de trois composantes : rouge (`r`), vert (`g`), et bleu (`b`). Cependant, pour la conversion en nuances de gris, nous n'utilisons que la composante de luminosité.

La conversion en nuances de gris consiste à calculer la valeur de luminosité pour chaque pixel, généralement en utilisant une formule pondérée. Pour cela nous avons choisi d'utiliser la formule de luminance standard, également connue sous le nom de "NTSC grayscale".

Cela implique que nous prenons en compte la composante rouge (`r`), la composante verte (`g`) et la composante bleue (`b`) pour calculer la luminosité. Le résultat est une valeur de luminosité qui varie de 0 (noir) à 255 (blanc).

Cela se définit de la manière suivante:

```
unsigned char grayValue = (image->pixels[i][j].r + image->pixels[i][j].g + image->pixels[i][j].b) / 3;  
image->pixels[i][j].r = grayValue;  
image->pixels[i][j].g = grayValue;  
image->pixels[i][j].b = grayValue;
```

Avec `pixel`, et `image` étant des structures définies précédemment, `image` ayant un attribut `pixels`, qui constitue un tableau de `pixel`, ainsi que de ses dimensions hauteur et largeur:

```
typedef struct{
    unsigned char r, g, b;
} Pixel;

typedef struct {
    unsigned int width;
    unsigned int height;
    Pixel **pixels;
    char *path;
} Image;
```

La variable path est le chemin d'accès au fichier de l'image.

Ainsi, après avoir calculé la valeur de luminosité pour chaque pixel, nous stockons cette valeur dans la structure de l'image à la place des composantes de couleur originales (r, g, b). Cette transformation convertit effectivement l'image en nuances de gris, ce qui simplifie le processus de traitement ultérieur, notamment la segmentation des chiffres du Sudoku.

L'image résultante en nuances de gris sera ensuite utilisée pour la suite du traitement d'image.

2.1.4- Renforcement des Contrastes

L'étape suivante consiste à améliorer le contraste de l'image en nuances de gris. Cette amélioration vise à accentuer les distinctions entre les zones sombres et claires de l'image pour faciliter la détection des chiffres ainsi que la détection de ligne, utile pour la segmentation des cases de la grille de Sudoku.

La fonction blackandwhite est responsable de cette transformation. Elle parcourt chaque pixel de l'image et effectue une opération de seuillage. L'opération de seuillage est utilisée pour diviser les pixels en deux catégories distinctes : les pixels foncés et les pixels clairs.

Pour réaliser cela, il est nécessaire de trouver la valeur optimale pour détacher le fond de l'image des objets présents dessus. Or, nous avons précédemment réalisé une fonction qui permet de faire cela: c'est la méthode d'Otsu. Nous avons donc la valeur optimale pour séparer l'arrière plan du devant, nous l'appellerons N par la suite

Si la valeur moyenne du pixel est supérieure ou égale à N , cela signifie que le pixel est considéré comme clair. Dans ce cas, les composantes de couleur (r, g, b) du pixel sont définies à N , ce qui correspond à la couleur blanche.

Si la valeur moyenne du pixel est inférieure à N , le pixel est considéré comme foncé, et ses composantes de couleur (r, g, b) sont définies à 0, ce qui correspond à la couleur noire.

Cette opération de seuillage crée un contraste net entre les zones sombres et claires de l'image, rendant ainsi les chiffres plus visibles et facilitant la segmentation des cases du Sudoku. Les pixels sont convertis en noir et blanc, ce qui améliore la qualité de l'image en vue de l'étape suivante du processus OCR.

Cette étape pouvait causer des problèmes pour certaines images. Notamment si l'image choisie était globalement claire et que les traits des chiffres et des lignes étaient dans une couleur trop peu foncée alors ils s'en retrouvaient effacé après application de notre fonction. C'est l'une des raisons qui nous a poussé à mettre au point d'autres filtres à appliquer en amont pour pallier ce potentiel problème.

On obtient alors une image résultante en noir et blanc qui est maintenant prête pour la segmentation des cases du Sudoku, où nous allons extraire les chiffres individuels pour la résolution du Sudoku. L'amélioration du contraste est une étape cruciale pour garantir la précision de notre OCR Sudoku Solver en optimisant la lisibilité de l'image.

2.2 Détection de la grille

2.2.1 Filtre de Sobel

Le filtre de sobel permet d'aider énormément la transformée de Hough à détecter les bords il s'agit d'un opérateur de traitement d'image. C'est donc grâce à ce filtre de traitement d'image que la transformée de hough serait plus précises et efficaces car sobel est un type de filtre basé sur la convolution. Il utilise des noyaux de convolution, un pour la détection des contours horizontaux et l'autre pour la détection des contours verticaux. Ces noyaux sont appliqués à l'image d'origine en utilisant l'opération de convolution.

```
void sobelFilter(Image* image, Image* end)
```

La fonction sobelFilter prends donc en paramètre une image (l'image que l'on va traiter) ainsi qu'une autre image qui sera donc l'image résultante après traitement.

```
for(size_t y = 1; y < image->height - 1; y++)
{
    for(size_t x = 1; x < image->width - 1; x++)
    {
        size_t sumX = 0;
        size_t sumY = 0;

        for(int i = -1; i <= 1; i++)
        {
            for(int j = -1; j <= 1; j++)
            {
                Pixel* pix = &image->pixels[y+j][x+i];
                sumX += pix->r * GX[ i + 1][ j + 1]; // applique le masque sur le canal rouge
                sumY += pix->r * GY[ i + 1][ j + 1]; // applique le masque sur le canal rouge
            }
        }

        //int gradient = sqrt(sumRx*sumRx + sumRy*sumRy);
        int gradient = abs(sumX) + abs(sumY);
        if(gradient > 255)
            gradient = 255;

        end->pixels[y][x].r = gradient;
        end->pixels[y][x].g = gradient;
        end->pixels[y][x].b = gradient;
    }
}
```

2.2.2 Détection des lignes par la transformée de Hough

La transformée de hough une technique de traitement d'image utilisé pour détecter des lignes ce qui permet également de détecter des formes spécifiques dans une image.

Elle a été initialement conçus pour identifier les lignes droites, cependant il a depuis été étendue pour détecter d'autres formes géométriques (triangle, carrée...etc). Le principe fondamental de la transformée de Hough repose sur la représentation des formes, notamment des lignes, dans un espace que l'on nomme espace de paramètres. Afin de détecter des lignes dans une image, chaque pixel contribue à une accumulation dans cet espace de paramètres. En particulier, pour chaque pixel appartenant (potentiellement) à une ligne, plusieurs paramètres, tels que l'angle et la distance par rapport à un point de référence, sont utilisés pour générer une courbe dans l'espace de Hough. Quand une ligne est présente dans l'image, ces courbes convergent vers un point (un pixel dans une image) dans l'espace de paramètres, formant ainsi un pic. La détection des lignes se traduit par la localisation de ces pics, ce qui permet de reconstruire les caractéristiques géométriques présentes dans l'image. La transformée de Hough s'est avérée être un outil essentiel et utile dans divers domaines tels que la représentation par ordinateur, la reconnaissance de formes et la robotique, contribuant de manière significative à la détection de structures linéaires complexes que l'on peut trouver dans divers images.

Dans notre contexte, la transformée de Hough va nous permettre de détecter les lignes dans l'image contenant la grille de sudoku. Nous allons donc accéder aux coordonnées des lignes à l'aide d'une liste de struct que nous avons définie comme ceci :

```
typedef struct {
    int x1, y1, x2, y2;
    int dot;
    struct Line* next;
} Line;
```

Grâce aux coordonnées des lignes du sudoku, nous allons donc pouvoir déduire les cases du sudoku ainsi que la position exacte de la grille du sudoku dans l'image.

```
Line* HoughTransform(GdkPixbuf* pixbuf, int* countLine)
```

L'image sera donc traitée à l'aide de la librairie gtk déjà utilisée pour l'interface graphique.

Une fois les cases détectées, nous allons donc les passer au réseau de neurones qui vont reconnaître les chiffres contenus dans les cases puis les mètres dans un fichier acceptable pour le solveur.

Voici un extrait de la fonction hough :

```
unsigned int accumulator[2 * diagonal + 1][181];
memset(accumulator, 0, sizeof(accumulator));

unsigned int max = 0;
double rho;
int rh = 0;
int numberofmax = 0;

// Get pixel data
guchar* pixels = gdk_pixbuf_get_pixels(pixbuf);

for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        // Get the RGB values of the pixel
        int offset = y * gdk_pixbuf_get_rowstride(pixbuf) + x * gdk_pixbuf_get_n_channels(pixbuf);
        guchar red = pixels[offset];
        guchar green = pixels[offset + 1];
        guchar blue = pixels[offset + 2];

        // Check if the pixel is white (modify this condition based on your PPM format)
        if (red == 255 && green == 255 && blue == 255) {
            for (int theta = 0; theta < 181; theta++) {
                double th = theta * M_PI / 180;
                rho = x * cos(th) + y * sin(th);
                rh = rho + diagonal;
                accumulator[rh][theta] += 1;

                if (accumulator[rh][theta] > max) {
                    max = accumulator[rh][theta];
                }
            }
        }
    }
}
```

eolzaar@
Document
OCR-20

```

int linethreshold = max * 0.36;
for (int r = 0; r <= 2 * diagonal; r++) {
    for (int t = 0; t <= 180; t++) {
        int val = accumulator[r][t];
        if (val >= linethreshold) {
            numberofmax++;
        }
    }
}

*countLine = numberofmax + 1;
res = calloc(numberofmax + 1, sizeof(Line));
for (int i = 0; i < numberofmax + 1; i++) {
    res[i].dot = 0;
}

int indexline = 0;
for (int t = 0; t <= 180; t++) {
    for (int r = 0; r <= 2 * diagonal; r++) {
        int val = accumulator[r][t];
        if (val >= linethreshold) {
            double rad = t * M_PI / 180;
            float x1 = ((r - diagonal) * cos(rad));
            float y1 = ((r - diagonal) * sin(rad));
            float x2, y2;

            if ((t == 90)) {
                // Vertical line
                x2 = x1 + width - 1;
                y2 = y1;// + height - 1;
            } else {
                // Non-vertical line
                x2 = x1 + diagonal * sin(-rad);//(x1 + (r - diagonal) * sin(-rad));
                y2 = y1 + diagonal * cos(rad);//(y1 + (r - diagonal) * cos(rad));
            }
            res[indexline].x1 = ((int)x1);
            res[indexline].y1 = ((int)y1);
            res[indexline].x2 = ((int)x2);
            res[indexline].y2 = ((int)y2);
        }
    }
}

```

2.2.3 Détection des cases dans la grille

La détection des grilles dans la case est rendue possible grâce à la transfromée de Hough qui renvoie une liste de lignes. Chaque ligne dans cette ligne étant un struct possède des attributs qui corresponde aux coordonée des lignes dans un plan cartésien : (xdébut, ydébut) et (xfin, yfin)

Avec cette information en main nous allons donc pouvoir déterminer les points d'intersection sur l'image et donc détecter l'image. Nous procédons donc à un découpage de l'image suivi d'un découpage de cases.

Toutes les cases sont donc enregistrées chacune dans une image de format PPM

```
eolzaar@DESKTOP-EUGO300:/mnt/c/Users/eolza/Documents/Projet/S3/TP/Proj/epita-prepa-asm-PROJ-OCR-2027-prs-026/Decoupage$ ls
Makefile SUDOKU.ppm decoupage.c
eolzaar@DESKTOP-EUGO300:/mnt/c/Users/eolza/Documents/Projet/S3/TP/Proj/epita-prepa-asm-PROJ-OCR-2027-prs-026/Decoupage$ make
gcc -Wall -Wextra decoupage.c -o decoupe
eolzaar@DESKTOP-EUGO300:/mnt/c/Users/eolza/Documents/Projet/S3/TP/Proj/epita-prepa-asm-PROJ-OCR-2027-prs-026/Decoupage$ ls
Makefile SUDOKU.ppm decoupage.c decoupe
eolzaar@DESKTOP-EUGO300:/mnt/c/Users/eolza/Documents/Projet/S3/TP/Proj/epita-prepa-asm-PROJ-OCR-2027-prs-026/Decoupage$ ./decoupe SUDOKU.ppm
eolzaar@DESKTOP-EUGO300:/mnt/c/Users/eolza/Documents/Projet/S3/TP/Proj/epita-prepa-asm-PROJ-OCR-2027-prs-026/Decoupage$ ls
Makefile case_0_6.ppm case_1_5.ppm case_2_4.ppm case_3_3.ppm case_4_2.ppm case_5_1.ppm case_6_0.ppm case_6_8.ppm case_7_7.ppm case_8_6.ppm
SUDOKU.ppm case_0_7.ppm case_1_6.ppm case_2_5.ppm case_3_4.ppm case_4_3.ppm case_5_2.ppm case_6_1.ppm case_7_0.ppm case_7_8.ppm case_8_7.ppm
case_0_0.ppm case_0_8.ppm case_1_7.ppm case_2_6.ppm case_3_5.ppm case_4_4.ppm case_5_3.ppm case_6_2.ppm case_7_1.ppm case_8_0.ppm case_8_8.ppm
case_0_1.ppm case_1_0.ppm case_1_8.ppm case_2_7.ppm case_3_6.ppm case_4_5.ppm case_5_4.ppm case_6_3.ppm case_7_2.ppm case_8_1.ppm decoupage.c
case_0_2.ppm case_1_1.ppm case_2_0.ppm case_2_8.ppm case_3_7.ppm case_4_6.ppm case_5_5.ppm case_6_4.ppm case_7_3.ppm case_8_2.ppm decoupe
case_0_3.ppm case_1_2.ppm case_2_1.ppm case_3_0.ppm case_3_8.ppm case_4_7.ppm case_5_6.ppm case_6_5.ppm case_7_4.ppm case_8_3.ppm
case_0_4.ppm case_1_3.ppm case_2_2.ppm case_3_1.ppm case_4_0.ppm case_4_8.ppm case_5_7.ppm case_6_6.ppm case_7_5.ppm case_8_4.ppm
case_0_5.ppm case_1_4.ppm case_2_3.ppm case_3_2.ppm case_4_1.ppm case_5_0.ppm case_5_8.ppm case_6_7.ppm case_7_6.ppm case_8_5.ppm
eolzaar@DESKTOP-EUGO300:/mnt/c/Users/eolza/Documents/Projet/S3/TP/Proj/epita-prepa-asm-PROJ-OCR-2027-prs-026/Decoupage$ make clean
rm -f decoupe case_*.*.ppm
eolzaar@DESKTOP-EUGO300:/mnt/c/Users/eolza/Documents/Projet/S3/TP/Proj/epita-prepa-asm-PROJ-OCR-2027-prs-026/Decoupage$ ls
Makefile SUDOKU.ppm decoupage.c
eolzaar@DESKTOP-EUGO300:/mnt/c/Users/eolza/Documents/Projet/S3/TP/Proj/epita-prepa-asm-PROJ-OCR-2027-prs-026/Decoupage$
```

Ces images seront donc traitées par le réseau de neurones pour détecter les chiffres et ensuite les mètres dans un format acceptable pour le solveur.

```
void decouperImage(FILE *image, int largeur, int hauteur, int nbColonnes, int nbLignes) {
    int largeurCase = largeur / nbColonnes;
    int hauteurCase = hauteur / nbLignes;
    for (int i = 0; i < nbLignes; i++) {
        for (int j = 0; j < nbColonnes; j++) {
            fseek(image, (i * hauteurCase * largeur + j * largeurCase) * 3, SEEK_SET);
            char *nomFichier = malloc(20 * sizeof(char));
            sprintf(nomFichier, "case_%d_%d.ppm", i, j);
            FILE *caseImage = fopen(nomFichier, "wb");
            free(nomFichier);
            fprintf(caseImage, "P6\n%d %d\n255\n", largeurCase, hauteurCase);
            for (int y = 0; y < hauteurCase; y++) {
                for (int x = 0; x < largeurCase; x++) {
                    char pixel[3];
                    fread(pixel, sizeof(char), 3, image);
                    fwrite(pixel, sizeof(char), 3, caseImage);
                }
                fseek(image, (largeur - largeurCase) * 3, SEEK_CUR);
            }
            fclose(caseImage);
        }
    }
}
```

2.3 Reconnaissance de Caractères (OCR)

2.3.1 Conception du Réseau de Neurones

Le réseau de neurones est une partie essentielle de ce projet. En effet, une implémentation mal faite ou implantée tardivement efface toutes les avancées produites dans le projet. Nous l'avons donc conçu dès le début du projet.

La façon dont nous avons codé ce réseau de neurones permet à celui -ci d 'être très facilement adapté à d'autres projets. En effet, pour déclarer en première instance, il suffit de donner en paramètre une liste d 'entiers représentant les nombres de neurones dans chaque couche. Les fichiers de sauvegarde gardent en mémoire les informations relatives au réseau, tels que le nombre de neurones dans chaque couche, leur poids, ou encore la taille du réseau lui -même. Toutes ces informations sont ensuite prises en compte lors du chargement du réseau. Il n'est donc pas nécessaire de se préoccuper des changements engendré si l'on ajoute des couches ou des neurones

Cette approche permet à un code compilé d 'être extrêmement polyvalent. En effet, pas besoin de changer manuellement la taille des poids ou des neurones à chaque implémentation différente. Un code compilé peut charger plusieurs réseaux de neurones, quelle que soit leur taille.

Plus concrètement, le réseau est l'assemblage de deux structures: les neurones et les layer. Les neurones contiennent évidemment la liste des poids, mais aussi le nombre de poids présents. Cette structure est semblable à celle des layer; ils contiennent une liste de neurones, et le nombre de neurones présent dans la couche. L'IA est donc composée d'une liste de layer, représentant la liste des différentes couches que l'on veut donner à notre réseau. A noter que la première couche donnée n'existe pas, elle sert simplement à indiquer à la deuxième couche de neurone combien d'informations elle doit accepter en entrée.

Une autre avantage de cette implémentation, est qu'il est assez facile et lisible de rédiger les différentes fonctions vital du réseau, telles que sa sauvegarde, le chargement depuis un fichier, ou encore l'exécution même du calcul pour cette IA. Il suffit simplement d'implémenter une fonction pour traiter un neurone, puis un layer, et d'appliquer le traitement voulu pour chacune des couches de l'IA.

Nous utilisons la fonction sigmoïde en tant que fonction d'activation. Nous l'avons choisie, car elle permet de mieux résoudre les problèmes non linéaires, tels que la fonction XOR, ou encore la reconnaissance de caractères. Cette fonction nous paraissait plus appropriée par rapport aux autres fonctions existantes, telles que la fonction ReLU, tanh, ou la fonction softmax.

Pour le moment, notre réseau de neurones à été seulement entraîné sur la fonction XOR. En effet, c'est un cas plutôt facile à gérer ; nous connaissons toutes les valeurs d'entrée possible, ainsi que leur résultat. Nous avons besoin de n'appliquer aucun traitement préalable, ce qui augmente drastiquement la simplicité, tout en diminuant les sources d'erreurs potentielles.

Le réseau de neurones est donc déjà capable d'appliquer la fonction XOR. Pour cela, comme il s'agit d'un problème non linéaire, il fallait au réseau de neurones une couche entre la couche d'entrée et celle de sortie.

Nous avons commencé à l'entraîner sans couche intermédiaire. Mais face aux résultats, soit incorrects dans certains cas, soit sans aucune précisions (des résultats retournant 0,65 environ), il a donc fallu faire des recherches et ajouter une couche supplémentaire.

Pour l'implémentation finale de ce réseau, donc pour la reconnaissance des chiffres, nous nous renseignerons avant de commencer tout entraînement, car le temps de calcul et de traitement rendra celui-ci beaucoup coûteux en temps, mais aussi en puissance de calcul.

Pour entraîner notre réseau de neurone à reconnaître les chiffre, nous allons utiliser les images présentes sur:

<https://knowyourdata-tfds.withgoogle.com/#tab=STATS&dataset=mnist>.

Nous l'utiliserons car cette base de données est vraiment immense – elle contient plus de 70 000 images – et qu'elle est déjà labellisée. Cela permettra d'entraîner l'IA sur un immense jeu de données, sans que nous ayons besoin d'étiqueter chacune de ces images. Le seul problème est que les images sont écrites blanc sur fond noir, mais une simple inversion des couleurs permettra d'avoir une base de donnée saine et correspondant à nos besoins.

Cela prendra certes un peu plus de temps, mais notre OCR sera vraiment très performant, même sur les images de faible qualité.

2.3.2 Entraînement du Réseau de Neurones

En ce qui concerne l'entraînement du réseau de neurone, il a fallu concevoir une fonction de backpropagation.

La backpropagation, ou rétropropagation en français, est une technique fondamentale dans l'entraînement des réseaux de neurones, jouant un rôle crucial dans l'ajustement des poids du réseau pour minimiser l'erreur de prédiction. Le processus commence par la phase de feedforward, où les données d'entrée sont propagées à travers le réseau jusqu'à la couche de sortie, générant ainsi une prédiction. En comparant cette prédiction à la vérité recherchée, l'erreur est calculée. Grâce à cela, le gradient est calculé. Il mesure à quel point chaque poids contribue à l'erreur globale entre la sortie réelle du réseau et la sortie attendue. C'est à ce stade que la backpropagation intervient. Elle consiste à rétro propager cette erreur en sens inverse, de la couche de sortie jusqu'à la couche d'entrée, en ajustant les poids des connexions à chaque couche dans la direction opposée au gradient pour minimiser cette erreur. La backpropagation se répète sur plusieurs itérations (aussi appelé époques) du jeu de données d'entraînement, ajustant progressivement les poids du réseau pour améliorer ses performances. Cette approche itérative permet au réseau de neurones de converger vers des poids optimaux, améliorant ainsi sa capacité à généraliser et à effectuer des prédictions précises sur de nouvelles données.

De plus, pour évaluer l'efficacité de notre fonction de rétropropagation, nous avons mis au point une fonction 'calculateLoss' qui permet de calculer l'erreur quadratique moyenne de notre réseau de neurone en prenant en compte ses résultats et ceux attendus. Plus l'erreur diminue avec l'entraînement, meilleure est notre fonction.

Enfin, pour rendre notre entraînement efficace et automatique, nous avons codé la fonction 'train' qui sert à lancer notre fonction 'backpropagation' un nombre de fois déterminé sur une banque de données passé en paramètre constitué des images à traiter ainsi que du résultat attendu. Ainsi cette fonction modifie et améliore continuellement les poids de notre réseau qui sont sauvegardés après chaque session d'entraînement. De plus, nous avons fait en sorte que cette fonction puisse suivre l'évolution de l'erreur quadratique moyenne afin de nous permettre de mieux suivre les changements de poids et adapter son apprentissage en modifiant les données ou le learningRate qui détermine l'ampleur des changements effectués sur les poids à chaque rétropropagation.

2.4 Résolution de Sudoku

2.4.1 Algorithme de Résolution

Le programme que nous avons conçu constitue une solution ingénieuse pour résoudre des Sudokus en utilisant un algorithme glouton récursif. L'algorithme s'attaque au défi en explorant de manière exhaustive toutes les possibilités pour chaque case du Sudoku, adoptant une approche méthodique pour garantir la découverte de la solution optimale. L'étape initiale consiste à identifier la première case vide dans le puzzle. Par la suite, l'algorithme entre dans une phase récursive où il évalue systématiquement toutes les valeurs envisageables pour cette case, en faisant des choix en fonction de critères intelligents. Ces critères incluent l'élimination de valeurs impossibles selon les règles bien définies du Sudoku, telles que l'unicité des chiffres dans chaque ligne, colonne et bloc. Lorsqu'une valeur est attribuée, l'algorithme se déplace vers la case suivante et réitère le processus. Si une impasse est atteinte, l'algorithme fait preuve de rétroactivité, revenant en arrière pour explorer d'autres options jusqu'à ce qu'une configuration viable soit trouvée pour l'ensemble du puzzle. Cette approche récursive et gloutonne garantit une exploration exhaustive de toutes les alternatives, assurant ainsi la résolution complète et efficace du Sudoku.

La fonction qui détermine si l'on placé num dans grid[row][col] :

```
int is_valid(int row, int col, int num)
{
    for (int i = 0; i < SIZE; i++) {
        if (grid[row][i] == num)
            return 0;
    }
    for (int i = 0; i < SIZE; i++) {
        if (grid[i][col] == num)
            return 0;
    }
    int startRow = row - row % 3, startCol = col - col % 3;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (grid[i + startRow][j + startCol] == num)
                return 0;
        }
    }
    return 1;
}
```

La fonction qui résout parcours et résout le sudoku

```
int solve_sudoku()
{
    int row;
    int col;
    int found = 0;
    for (row = 0; row < SIZE; row++) {
        for (col = 0; col < SIZE; col++) {
            if (grid[row][col] == 0) {
                found = 1;
                break;
            }
        }
        if (found) break;
    }

    if (found == 0) return 1;

    for (int num = 1; num <= SIZE; num++) {
        if (is_valid(row, col, num)) {
            grid[row][col] = num;
            if (solve_sudoku()) return 1;
            grid[row][col] = 0;
        }
    }
    return 0;
}
```

2.5 Interface Graphique

2.5.1-Éléments de l'interface

L'interface est très importante; elle fait le lien entre l'utilisateur et la machine. L'interface doit être suffisamment facile et intuitive pour être rapidement prise en main.

Nous avons cherché à imposer le moins de contraintes possibles à l'utilisateur. Les boutons et éléments présents ne représentent donc que le strictement nécessaire.

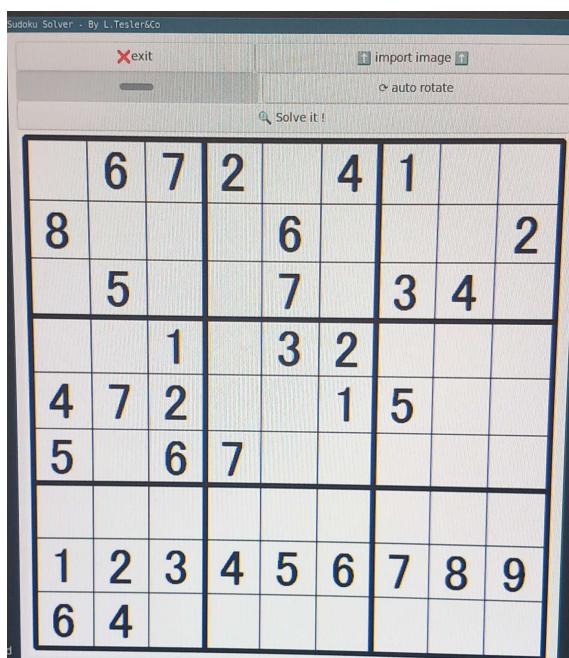
Par exemple, par esthétisme, le logiciel affiche en temps réel les opérations effectuées, par exemple lors du traitement d'image. Mais il n'est pas nécessaire à l'utilisateur d'appliquer par lui-même les différents traitements, en les sélectionnant via les différents boutons.

L'interface comporte plusieurs éléments :

- Un premier bouton permet de charger une image, afin que celle-ci puisse être traitée par les différents algorithmes de ce projet.

- La deuxième ligne concerne la rotation. Un curseur à gauche permet d'effectuer une rotation manuelle. Pour une plus grande précision, ou tout simplement pour choisir le meilleur angle, il est également proposé de laisser la machine effectuer une rotation automatique. De plus, cela permet déjà d'effectuer un prétraitement de l'image, et donc d'optimiser légèrement le processus. Évidemment, le temps gagné est assez faible; les fonctions de traitement d'image étant plutôt optimisées.

- Enfin, le bouton permettant la résolution finale du sudoku.



Comme vous pouvez le constater, cette interface est assez simpliste. En effet, nous avions déjà un léger retard, et il nous paraissait plus important de se concentrer sur les parties fonctionnelles et concrètes à la résolution du sudoku, plutôt que de chercher à peaufiner en détail l'interface.

Nous avons de nombreuses difficultés pour réaliser l'interface.

Au commencement, il fallait décider des différents éléments qui seront présents dessus. Il est en effet assez difficile d'insérer des images ou des boutons au centre d'un travail déjà produit !

Une première version a été réalisée. Elle était bien moins intuitive que celle actuelle.

Tout d'abord, il fallait impérativement l'exécuter via un terminal de commande. En effet, nous n'avions pas implémenté de bouton pour quitter l'application. Il fallait donc, soit interrompre le programme, soit forcer l'arrêt en arrêtant le processus. Cette approche peu viable a donc été abandonnée, au profit d'un simple bouton.

On remarque par ailleurs qu'avant l'actuel bouton de sortie, se trouvait un champ de texte. C'était la partie la moins intuitive de cette ancienne interface. Il fallait écrire le nom de l'image à charger (voir même son chemin, si celle-ci se trouvait dans un autre dossier) pour uploader sur l'interface !

Bien sûr, cela posait beaucoup de problèmes, notamment au niveau de la simplicité d'accèsibilité. Il était fréquent que des erreurs de frappe gênent l'envoi de l'image.

Il a donc fallu changer cette fonction, au profit d'un chargement plus classique.

2.5.2-Backend

Cette interface a dû être sujette à des optimisations. En effet, il est assez coûteux en temps d'écrire et de lire dans la mémoire de l'ordinateur. Ce temps peut être imperceptible, mais pas lorsqu'il est répété mainte et mainte fois !

Un cas concret concerne la rotation de l'image. Il est assez coûteux en ressource mais aussi en temps de récupérer l'image affichée, pour ensuite la donner à différents traitements, notamment la rotation. De plus, pour pouvoir offrir une certaine fluidité, il faut appliquer la rotation très souvent, à chaque changement de valeurs. Il est donc nécessaire d'optimiser au maximum cette partie-là, car le moindre temps perdu entraîne de grosses secondes de latence. Agréable oeil . Il a donc été choisi d'avoir

3 - Perspectives et Défis Futurs

3.1 Améliorations Possibles

En ce qui concerne notre projet, de nombreuses améliorations sont possibles, notamment sur notre implémentation du transformé de hough utilisé pour la détection de ligne. De plus, une meilleure mise en œuvre de notre réseau de neurones aurait pu faciliter la mise au point de l'entraînement de ce dernier qui s'est avéré être particulièrement complexe et chronophage.

En outre, il aurait été envisageable, si une meilleure organisation de notre groupe avait pu être instaurée, de perfectionner le traitement d'image afin d'éviter certains problèmes liés à des images aux caractéristiques irrégulières, et moins conforme à ce que nous avions l'habitude de traiter.

Concernant notre groupe, il aurait été judicieux d'être plus vigilant quant aux travail fournis par chaque membre du groupe. Des vérifications des tâches effectuées auraient sans doute permis d'éviter certaines complications survenues ultérieurement. En effet, lors du développement du projet, le fait que certaines implantations n'aient pas été fonctionnelles a grandement freiner sa progression et sa réussite.

Nous avons également rencontré certaines difficultés concernant la gestion du temps. Ainsi, il serait important de porter particulièrement notre attention sur ce point pour nos futurs projets.

3.2 Défis Envisagés

Nous regrettons de ne pas avoir été en mesure de mener à bien les objectifs secondaires du projet malgré nos nombreux efforts.

En effet, les fonctionnalités supplémentaires, telles que la reconnaissance de chiffres manuscrits et la résolution d'hexadokus, auraient grandement enrichi notre projet. Ayant acquis de nouvelles expériences, la réalisation de ces fonctions, même après le rendu du projet, présenterait pour nous un véritable intérêt.

4 - Conclusion

4.1 Bilan du Projet

En conclusion de ce projet OCR Sudoku Solver, nous sommes fiers de présenter une application aboutie et fonctionnelle, résultat de notre engagement collectif.

Depuis la phase de conception, et jusqu'à la réalisation finale, notre équipe a surmonté de nombreux défis techniques et a pris des décisions réfléchies pour aboutir à une solution intégrée. La combinaison de la reconnaissance optique de caractères, du réseau de neurones, et de l'algorithme de résolution de sudoku a démontré son efficacité dans la résolution automatisée de grilles.

L'interface graphique offre une expérience utilisateur intuitive, renforçant l'accessibilité de notre application.

Ce parcours a été une occasion précieuse d'apprentissage et de croissance, témoignant de notre capacité à aborder des problématiques complexes ainsi que notre aptitude à nous adapter lors de difficultés rencontrées.

4.2 Apprentissages Tirés

Thomas,

J'aurais tiré de ce projet de nombreux apprentissages.

En premier lieu, j'ai pu constater l'ampleur des responsabilités que représente le rôle de chef de projet. La gestion d'une équipe et l'organisation au sein de celle-ci étaient des tâches nouvelles et complexes pour moi. Je n'avais auparavant jamais dû gérer le temps et les membres d'un groupe, en les relançant ou les motivant par exemple. Avoir pu endosser ce rôle m'a permis d'acquérir des compétences qui me seront utiles pour de futurs projets ou dans ma vie professionnelle.

Ce projet m'a également apporté l'opportunité d'en apprendre plus concernant de nouveaux domaines qui m'étaient étrangers comme la gestion d'image ou les réseaux de neurones. J'ai beaucoup aimé travailler sur ces problématiques.

Ainsi, la réalisation de ce projet a été une expérience personnelle enrichissante qui m'a plu.

Théophile,

Ce projet a été très instructif. J'ai beaucoup appris dans le domaine de l'intelligence artificielle. Ce projet m'a permis d'apprendre de vraies fonctions concernant leur apprentissage. Cette expérience m'a formée dans les bonnes utilisations du langage C, notamment au niveau de la gestion de la mémoire, ou des différentes optimisations possibles.

J'ai également beaucoup apprécié le traitement d'image. Tout comme l'apprentissage du réseau, il s'agit de beaucoup de mathématiques, ce qui est assez difficile de prime abord. Cependant, j'en tire une certaine fierté d'avoir su comprendre et appliquer tous ces principes.

Etienne,

Durant ce projet, j'ai pu acquérir de nombreuses compétences malgré les difficultés rencontrées. Je suis pas moins satisfait du travail que nous avons fourni et de notre cohésion de groupe qui n'a jamais été aussi haute. Ce projet m'a surtout permis d'acquérir beaucoup d'expérience que ce soit dans la gestion du temps, la communication dans le groupe, la programmation également la présentation orale d'un projet. Sans compter l'apprentissage des compétences dans le domaine technique comme le traitement d'image ou encore les réseaux de neurones.

4.3 Remerciements

Il nous a paru important de remercier tous ceux qui ont aidé à la réalisation de ce projet, que ce soit des amis, des camarades de classe, ou d'illustres inconnus sur des forums, leurs aides furent précieuses et nous leur en sommes sincèrement reconnaissant.

5. Annexes

5.1 Code Source

Annexe 1:

```
Image loadPPMImage(const char *filename) {
    Image image;
    FILE *file = fopen(filename, "rb");

    if (!file) {
        fprintf(stderr, "Erreur : Le fichier n'existe pas ou ne peut pas être ouvert.\n");
        exit(1);
    }

    // Lire l'en-tête PPM
    char magic[3];
    fscanf(file, "%2s", magic);
    if (strcmp(magic, "P6") != 0) {
        fprintf(stderr, "Erreur : Le format du fichier est incorrect.\n");
        fclose(file);
        exit(1);
    }

    fscanf(file, "%u %u", &image.width, &image.height);
    // Lire la valeur maximale (généralement 255 pour les images PPM)
    unsigned int maxVal;
    fscanf(file, "%u", &maxVal);

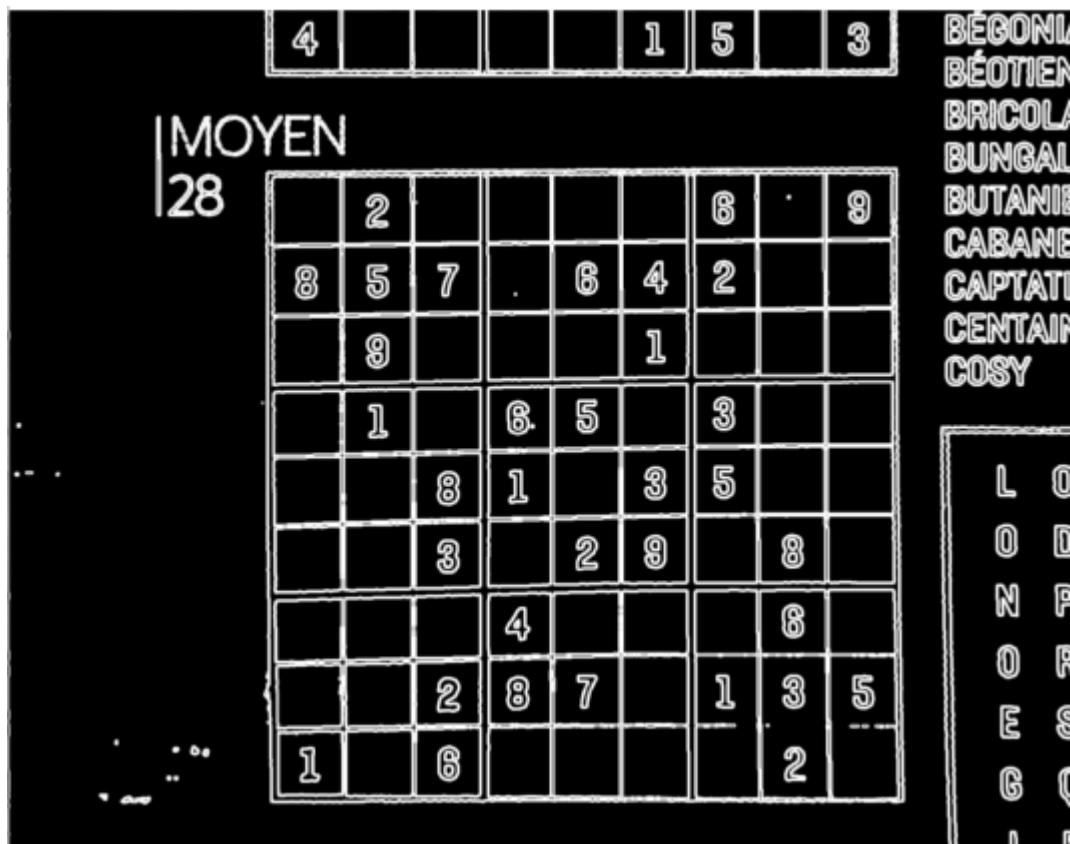
    // Allouer de la mémoire pour les pixels
    image.pixels = (Pixel **)malloc(image.height * sizeof(Pixel *));
    if (image.pixels == NULL)
    {
        printf("memory error");
        return image;
    }

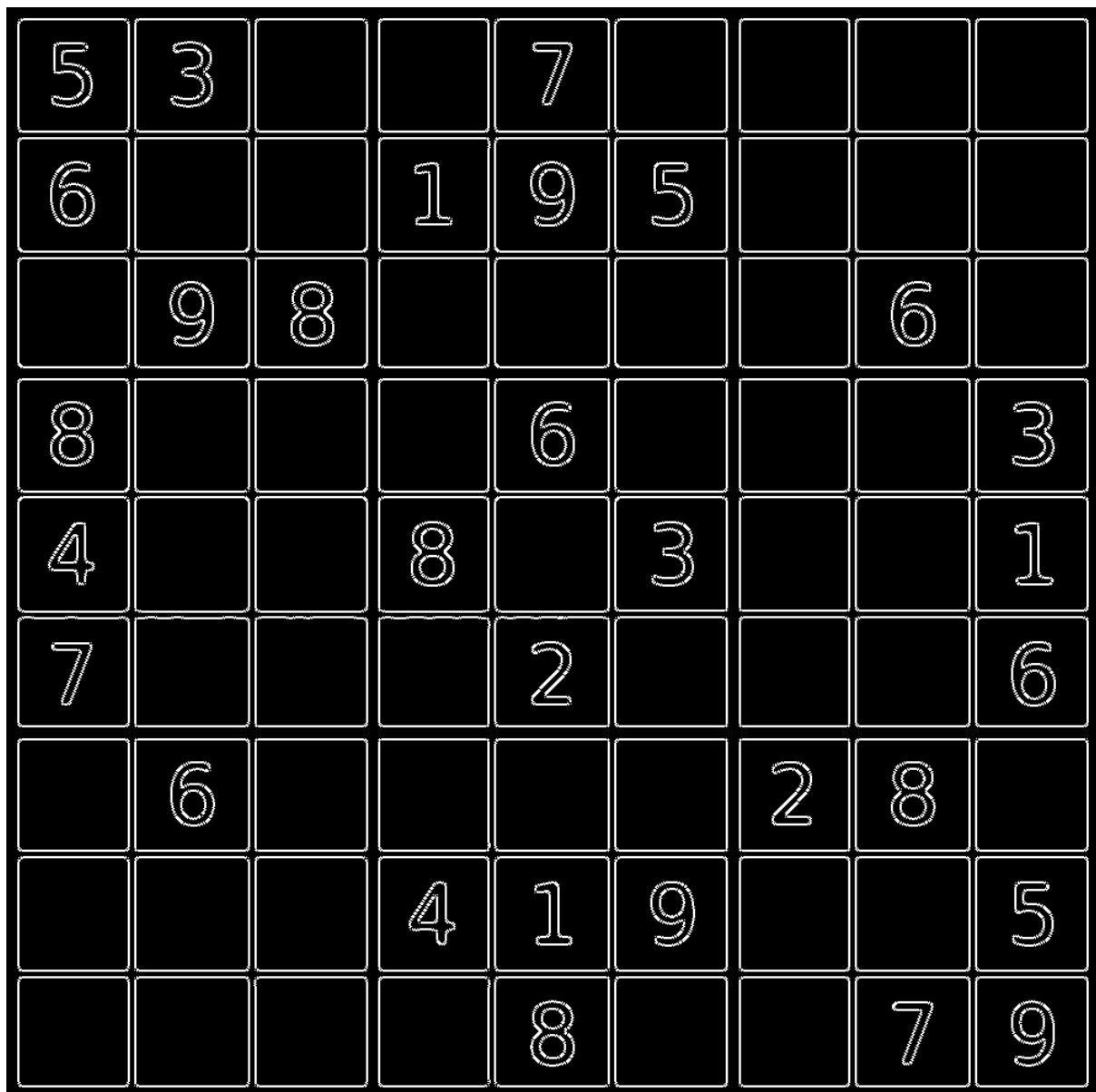
    for (unsigned int i = 0; i < image.height; i++) {
        image.pixels[i] = (Pixel *)malloc(image.width * sizeof(Pixel));
    }

    // Lire les données des pixels
    for (unsigned int i = 0; i < image.height; i++) {
        for (unsigned int j = 0; j < image.width; j++) {
            fread(&image.pixels[i][j], sizeof(Pixel), 1, file);
        }
    }

    fclose(file);
    image.path = strdup(filename);
    return image;
}
```

5.2 Exemples d'Images Traitées





5.3 Documentation Supplémentaire

Le site employé pour une partie de notre base de donné pour l'entraînement du réseau:

<https://www.kaggle.com/datasets/shreyasshrawage/digits>