

EPITA – PROJET OCR

RAPPORT DE SOUTENANCE FINALE

Reconnaissance Optique de Caractères et Résolution de Mots-Mêlés

Groupe : prs B2 G10

Benjamin Raccah (Interface et Pré-traitements)

Georges Savini (Chef de projet et Solver)

Guilhem Petit (Détection et Segmentation)

Kevin To (Intelligence Artificielle)

Date : 15/12/2025

Année Universitaire 2025-2026

Table des matières

1	Introduction et Contexte du Projet	4
1.1	Présentation du sujet	4
1.2	Évolution depuis la première soutenance	4
2	Organisation et Gestion de Projet	6
2.1	Répartition des rôles	6
2.2	Outils de communication et de suivi	7
3	Architecture Technique	10
3.1	Le Pipeline de Traitement	10
3.2	Structure des Fichiers	11
4	Interface Graphique et Traitement d'Image	13
4.1	Conception de l'interface	13
4.2	La Rotation d'Image (Deskewing)	14
4.3	Le Nettoyage (Preprocessing)	15
5	Segmentation et Analyse Structurelle	16
5.1	Le défi de la structure variable	16
5.2	Découpage en cellules (Slicing)	17
5.3	Traitement du "Niveau 3" (Les Oiseaux)	18
6	Réseau de Neurones et Apprentissage	19
6.1	Théorie Mathématique	19
6.1.1	Fonction d'activation	19
6.2	Architecture du Réseau	19

6.3	Entraînement (Training)	20
7	Algorithmique de Résolution	21
7.1	Lecture et structure de données	21
7.2	Algorithme de recherche	21
8	Difficultés Rencontrées et Solutions Apportées	23
8.1	L'intégration GTK et le Threading	23
8.2	La convergence du Réseau de Neurones	23
8.3	La Segmentation des lettres collées	24
9	Manuel d'Utilisation	25
9.1	Pré-requis Système	25
9.2	Compilation	26
9.3	Utilisation de l'Interface Graphique	26
9.3.1	Étape 1 : Chargement	27
9.3.2	Étape 2 : Pré-traitement	27
9.3.3	Étape 3 : Résolution	28
9.4	Dépannage courant	28
10	Bilan et Conclusion	29
10.1	Bilan Technique	29
10.2	Apprentissages Personnels	29
10.3	Conclusion	30
A	Annexes : Code Source Intégral	31
A.1	Point d'entrée : main.c	31
A.2	Interface Graphique : gui.c	33
A.3	Intelligence Artificielle : networks.c	35
A.4	Algorithme Solver : solver.c	37
A.5	Makefile	39
B	Bibliographie et Références	41

Remerciements

Nous tenons tout d'abord à remercier l'équipe pédagogique d'EPITA et nos assistants (ACU/YAKA) pour leur encadrement tout au long de ce semestre. Leurs conseils avisés lors des suivis de projet nous ont permis de surmonter les obstacles techniques et de mieux structurer notre approche.

Nous remercions également l'ensemble des membres du groupe pour leur investissement, leur patience et leur capacité à travailler en équipe malgré les périodes de forte charge de travail. Ce projet a été une véritable aventure humaine et technique.

Chapitre 1

Introduction et Contexte du Projet

1.1 Présentation du sujet

La Reconnaissance Optique de Caractères (OCR) est un domaine de l'intelligence artificielle et du traitement du signal qui consiste à traduire des images de textes imprimés ou dactylographiés en fichiers de texte éditables par un ordinateur.

Dans le cadre de notre formation de deuxième année à l'EPITA, nous avons été chargés de réaliser une application complète en langage C capable non seulement de lire des caractères, mais d'appliquer cette lecture à un contexte ludique précis : la résolution automatique de grilles de mots-mêlés.

Le défi est double : il s'agit d'abord de traiter une image brute (parfois bruitée, mal éclairée ou inclinée) pour en extraire l'information pertinente, puis d'utiliser des algorithmes de recherche pour résoudre le puzzle.

1.2 Évolution depuis la première soutenance

Lors de notre première soutenance, nous avions présenté une "Preuve de Concept" (POC). À ce stade, notre application était rudimentaire :

- La détection de la grille était statique et ne supportait pas les variations.
- La reconnaissance de caractères se faisait par une méthode naïve de comparaison pixel par pixel (XNOR), très sensible aux changements de police ou de taille.
- L'interface graphique était minimale.

Pour cette version finale, nous avons opéré une refonte majeure de l'architecture :

- Intégration d'un **Réseau de Neurones (Neural Network)** entraîné pour remplacer l'algorithme XNOR.
- Développement d'une **segmentation dynamique** capable de gérer des grilles complexes (Niveau 2 et 3).
- Ajout de fonctionnalités de **rotation automatique** et de nettoyage avancé.

Chapitre 2

Organisation et Gestion de Projet

Pour mener à bien un projet de cette envergure sur plusieurs mois, une organisation rigoureuse était indispensable. Nous avons adopté des méthodes inspirées des pratiques Agiles pour coordonner nos efforts.

2.1 Répartition des rôles

Dès le début du projet, nous avons identifié les forces de chacun pour répartir les tâches de manière optimale. Cette spécialisation n'a pas empêché l'entraide, mais elle a permis à chacun d'être responsable d'un module critique.

- **Georges Savini (Chef de Projet & Solver)** : Il a eu la lourde tâche de coordonner le groupe, de gérer le dépôt Git et de s'assurer que les délais étaient respectés. Techniquement, il a conçu le module final de résolution.
- **Benjamin Raccah (Interface & Pré-traitement)** : Il a fait le lien entre l'utilisateur et la machine. Son travail sur GTK et les manipulations matricielles pour la rotation des images a été central.
- **Guilhem Petit (Segmentation)** : Il a travaillé sur la "vision" du logiciel : comment découper une image en lettres exploitables. C'est le pont entre l'image brute et l'IA.
- **Kevin To (IA)** : Il s'est concentré sur la partie purement algorithmique et mathématique du réseau de neurones, passant du temps à entraîner le modèle et ajuster les hyperparamètres.

2.2 Outils de communication et de suivi

Nous avons utilisé plusieurs outils pour fluidifier notre travail :

- **Git & GitHub** : Pour le versioning du code. Nous avons travaillé avec des branches par fonctionnalité (`feature/gui`, `feature/network`) pour éviter les conflits sur la branche `master`.
- **Discord** : Pour les réunions hebdomadaires et le partage d'écran lors des sessions de débogage.
- **Trello** : Pour suivre l'état d'avancement des tâches (To Do, Doing, Done).

The screenshot shows a Git interface with the following elements:

- Current branch:** RotationFinal
- Last fetched:** 3 minutes ago
- Branches tab:** Active
- Pull requests tab:** Inactive
- Filter input field:** Filter
- New branch button:** New branch

Branch	Last updated
✓ RotationFinal	19 hours ago
✗ finition	7 hours ago
✗ loader	15 days ago

Other branches

Branch	Last updated
✗ origin/DetectionFinal	12 days ago
✗ origin/DetectionGuiGui	1 hour ago
✗ origin/FINALLLDETECTION	4 minutes ago
✗ origin/Interface2	10 days ago
✗ origin/InterfaceFV2	6 hours ago
✗ origin/InterfaceGS	10 days ago
✗ origin/MariageGS	7 days ago
✗ origin/Organisation	last month
✗ origin/Surlignement	6 hours ago
✗ origin/detection2	last month
✗ origin/detectionGs	last month

FIGURE 2.1 – Visualisation de l’arborescence Git et de la gestion des branches.

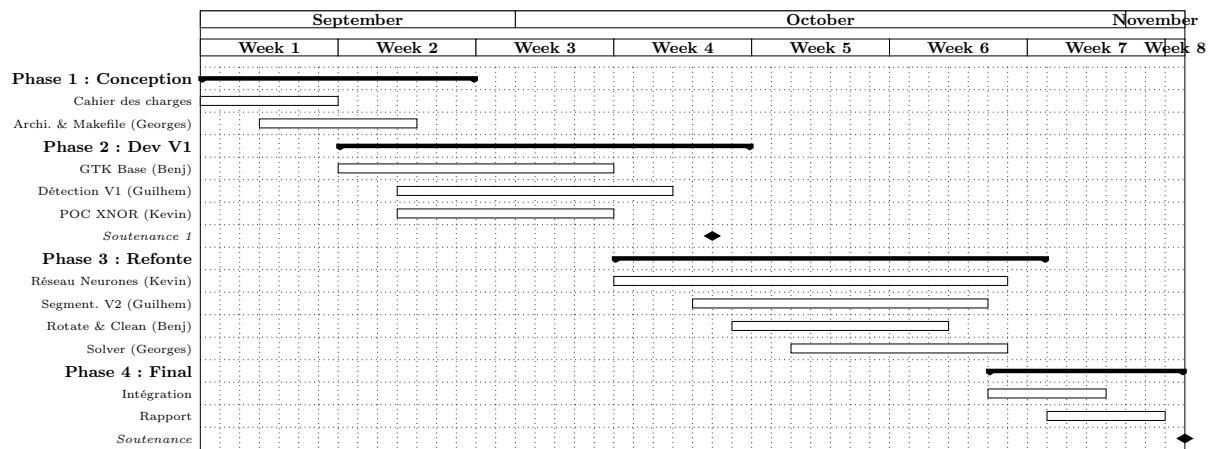


FIGURE 2.2 – Diagramme de Gantt simplifié de l'avancement du projet.

Chapitre 3

Architecture Technique

L'application est construite de manière modulaire. Le fichier `main.c` agit comme un chef d'orchestre qui appelle les différentes bibliothèques.

3.1 Le Pipeline de Traitement

Le flux de données suit une logique linéaire stricte pour garantir la stabilité :

1. **Input** :** L'utilisateur charge une image (JPG/PNG).
2. **Pré-traitement** (Module GUI) :** L'image est convertie en niveaux de gris, binarisée, et redressée (rotation).
3. **Segmentation** (Module Detection) :** L'image est découpée. On extrait deux zones : la grille et la liste de mots. Ces zones sont ensuite sous-découpées en "cellules" (petites images de lettres).
4. **Reconnaissance** (Module IA) :** Chaque cellule est envoyée au réseau de neurones qui retourne un caractère (char).
5. **Reconstruction** :** On crée une matrice de texte (fichier `.txt`) représentant la grille.
6. **Résolution** (Module Solver) :** On cherche les mots dans la matrice.
7. **Output** :** On affiche la solution à l'écran.

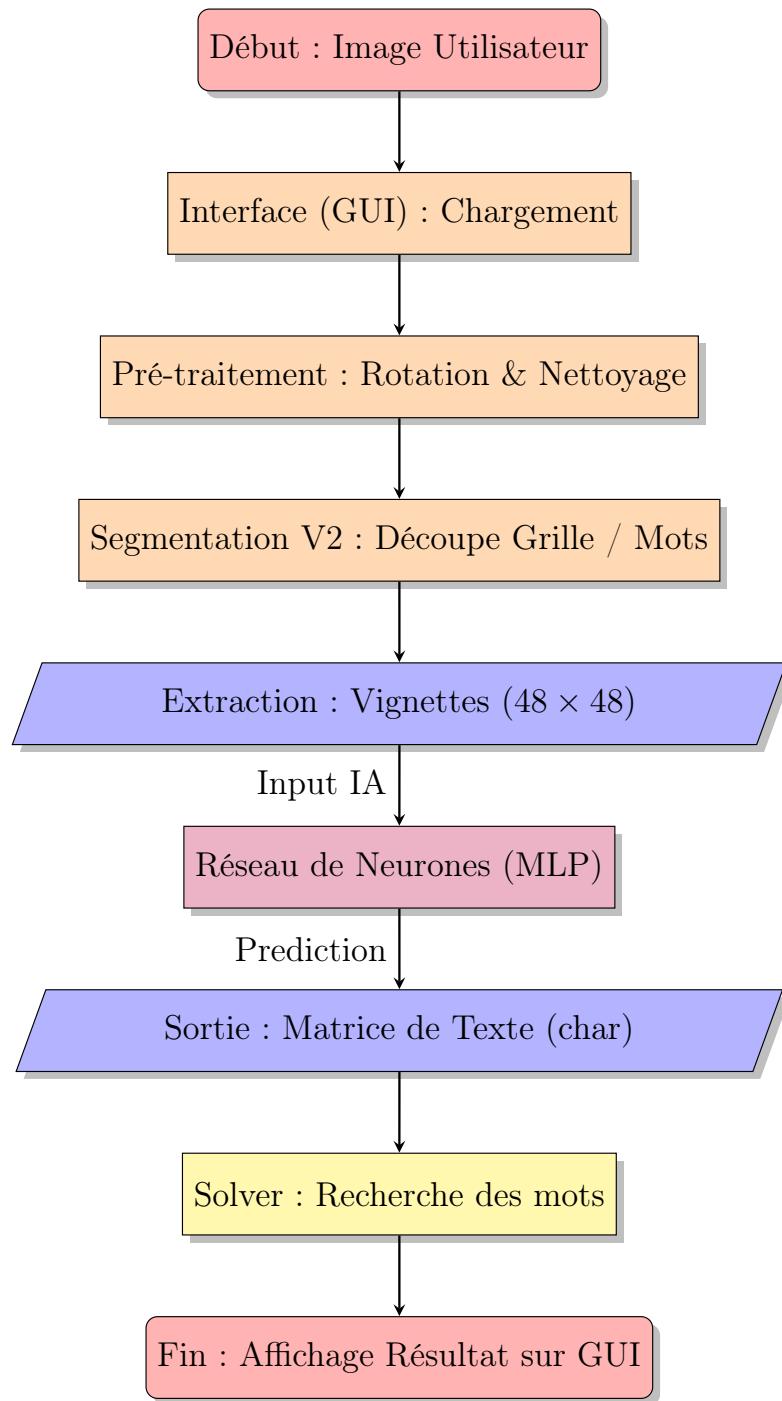


FIGURE 3.1 – Schéma global du flux de données dans l'application.

3.2 Structure des Fichiers

L’arborescence du projet reflète cette modularité :

- `/interface` : Contient `gui.c` et `gui.h`.

- **/rotations** : Contient les algorithmes de manipulation d'image.
- **/detectionV2** : Contient **detect_lettergrid.c** et la logique de segmentation.
- **/neuronne** : Contient le cœur de l'IA (**networks.c**) et le fichier de sauvegarde **brain.bin**.
- **/solver** : Contient l'algorithme de recherche de mots.

Cette séparation nous a permis de compiler chaque module indépendamment grâce à un **Makefile** unifié, conçu par Georges.

Chapitre 4

Interface Graphique et Traitement d'Image

L'interface graphique (GUI) développée par Benjamin est la vitrine de notre projet. Elle a été codée en C avec la bibliothèque **GTK+ 3.0**.

4.1 Conception de l'interface

Nous avons voulu une interface sobre et fonctionnelle. Elle se compose d'une fenêtre principale affichant l'image chargée et d'une barre d'outils latérale ou inférieure proposant les actions principales : "Load", "Auto Rotate", "Clean", "Save" et "Solve".

L'utilisation de `GdkPixbuf` a été centrale pour la gestion des pixels en mémoire. C'est une structure qui permet de manipuler les images pixel par pixel, ce qui est indispensable pour nos algorithmes de traitement.

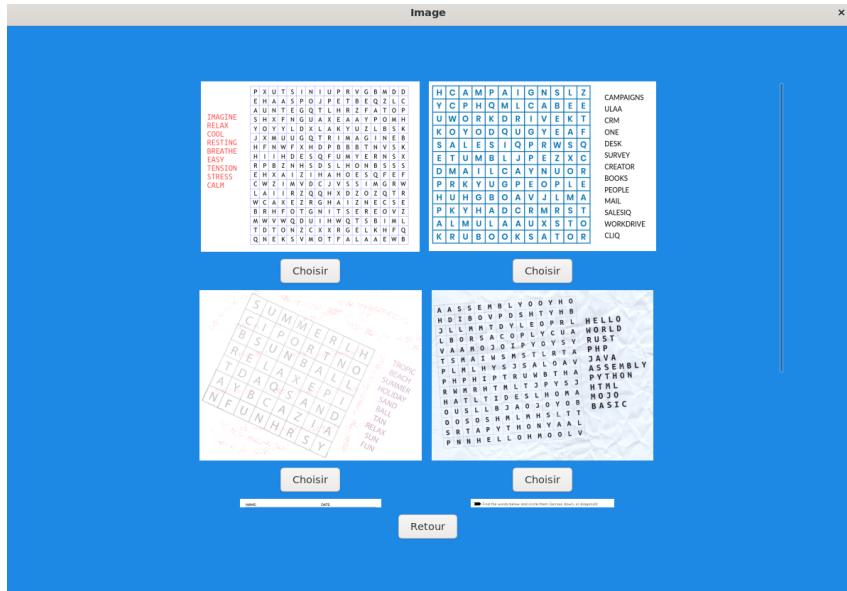


FIGURE 4.1 – Fenêtre principale de l’application permettant le chargement de l’image.

4.2 La Rotation d’Image (Deskewing)

Un des points forts de notre projet est sa capacité à traiter des images prises de travers. La fonction `rotate_pixbuf_any_angle` implémente une rotation matricielle. Pour chaque pixel (x', y') de l’image de destination, on calcule son antécédent (x, y) dans l’image source via la matrice de rotation inverse :

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x' - x_c \\ y' - y_c \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix}$$

Cette approche évite les trous dans l’image finale.

L’option **“Auto Rotate”** tente de détecter l’angle d’inclinaison dominant (via une détection de lignes de Hough simplifiée ou une analyse de variance) pour redresser l’image automatiquement.

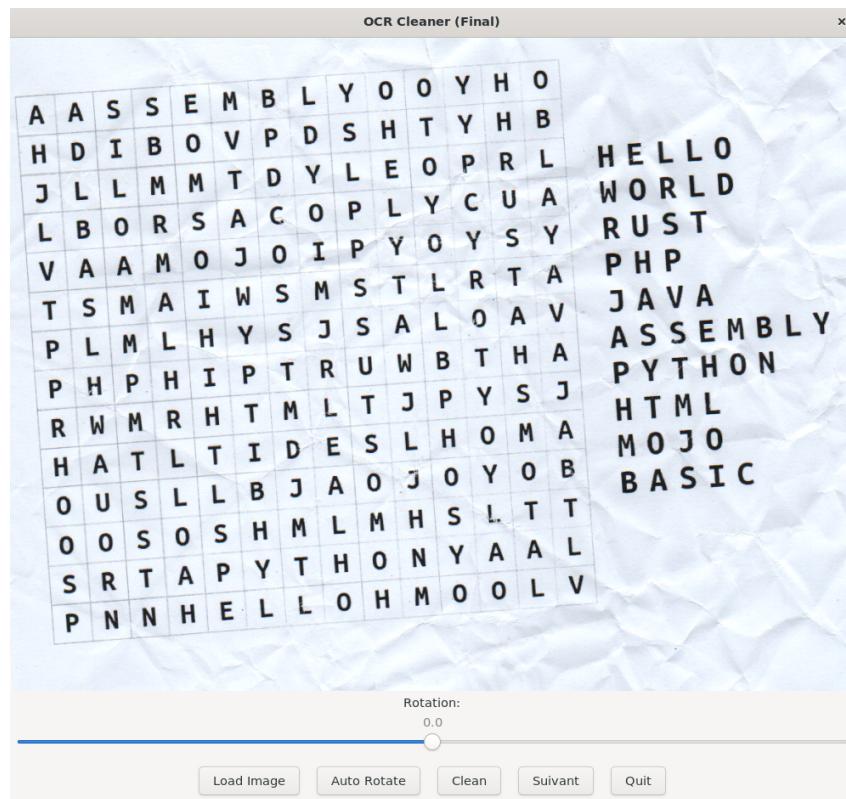


FIGURE 4.2 – Interface de pré-traitement : Rotation manuelle et automatique.

4.3 Le Nettoyage (Preprocessing)

Avant d'envoyer l'image à la détection, le bouton "Clean" applique plusieurs filtres :

1. **Niveaux de gris :** Moyenne pondérée des canaux RGB ($0.299R + 0.587G + 0.114B$).
2. **Contraste :** Augmentation dynamique pour bien séparer le noir du blanc.
3. **Binarisation :** Application d'un seuil pour n'avoir que du noir pur et du blanc pur.

Chapitre 5

Segmentation et Analyse Structurelle

Ce module, développé par Guilhem, est probablement le plus complexe en termes de logique impérative. Il se situe dans le dossier `detectionV2`.

5.1 Le défi de la structure variable

Contrairement au niveau 1 où la grille est toujours au même endroit, les niveaux supérieurs mélangent grille et texte. L'algorithme parcourt l'image verticalement et horizontalement pour calculer des histogrammes de densité de pixels noirs.

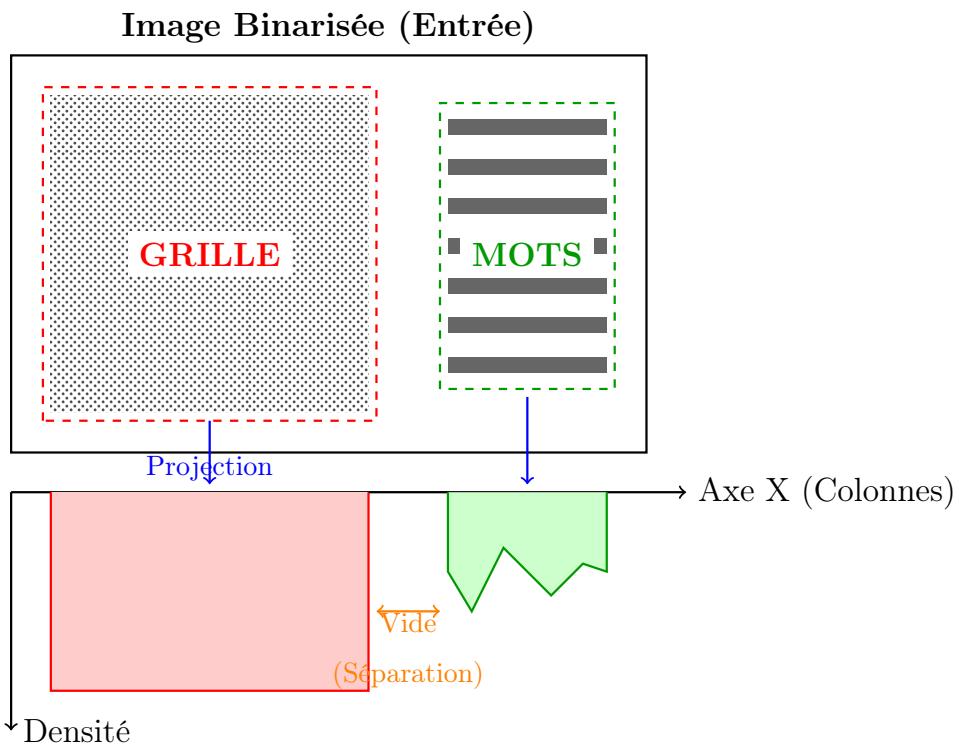


FIGURE 5.1 – Principe de la projection de densité verticale pour séparer la grille de la liste de mots.

Si une zone présente une forte densité régulière de noir, elle est identifiée comme la grille. Les zones périphériques de densité moindre sont identifiées comme la liste de mots.

5.2 Découpage en cellules (Slicing)

Une fois la grille localisée (coordonnées $x_{min}, y_{min}, x_{max}, y_{max}$), il faut extraire chaque case. L'algorithme `detect_letters_in_grid` utilise une détection de lignes et de colonnes pour quadriller la zone. Chaque intersection définit une cellule.

Ces cellules sont ensuite redimensionnées en 48×48 pixels et sauvegardées temporairement dans le dossier `cells/`. C'est une étape cruciale : si le découpage est mauvais (lettre coupée en deux), l'IA ne pourra pas la reconnaître.

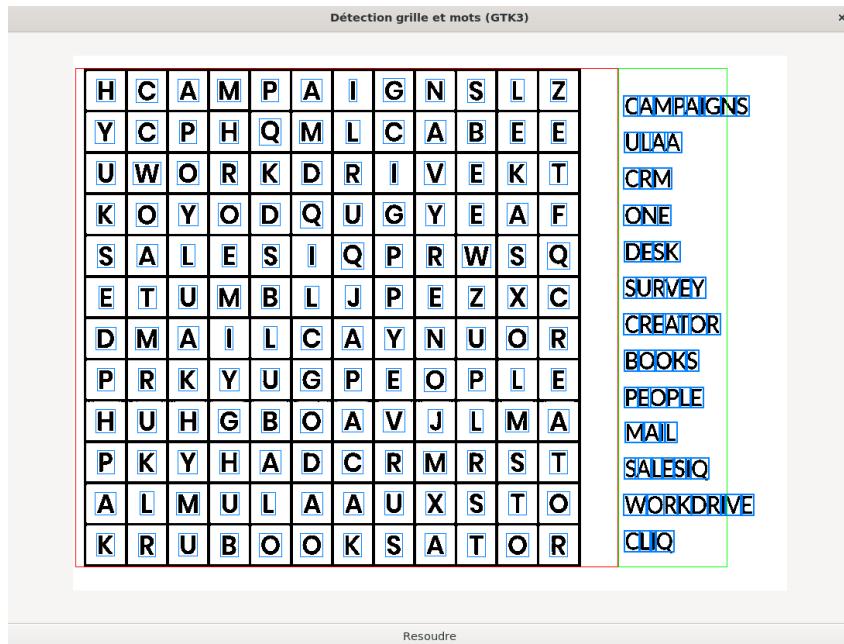


FIGURE 5.2 – Résultat de la segmentation : La grille est encadrée en rouge, les mots en vert.

5.3 Traitement du "Niveau 3" (Les Oiseaux)

Le niveau 3 introduisait du bruit sous forme de dessins (oiseaux) collés à la grille. Une suppression naïve du bruit effaçait aussi les bordures de la grille. La solution trouvée a été de : 1. Déetecter le grand cadre extérieur. 2. Supprimer tout ce qui touche ce cadre (oiseaux inclus). 3. **Re-dessiner artificiellement** un cadre noir propre autour de la zone détectée pour aider le découpage suivant.

Chapitre 6

Réseau de Neurones et Apprentissage

C'est ici que réside la plus grande avancée technique du projet. Kevin a implémenté un **Perceptron Multicouche (MLP)** complet en langage C, sans utiliser de librairie de haut niveau comme TensorFlow ou PyTorch.

6.1 Théorie Mathématique

Le réseau fonctionne par propagation avant et rétropropagation de l'erreur.

6.1.1 Fonction d'activation

Nous avons utilisé la fonction Sigmoïde pour les couches cachées, définie par :

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Cette fonction permet d'introduire de la non-linéarité dans le modèle, ce qui est indispensable pour apprendre des formes complexes comme des lettres.

Pour la couche de sortie, nous sommes passés à une fonction **Softmax**, qui transforme les sorties brutes en probabilités (la somme fait 100%).

6.2 Architecture du Réseau

Le réseau est défini dans `networks.h` et possède la topologie suivante :

- **Couche d'entrée :** 2304 neurones. Cela correspond à nos images de 48×48 pixels ($48 \times 48 = 2304$). Chaque pixel est une entrée (0 pour noir, 1 pour blanc).
- **Couche cachée 1 :** 64 neurones.
- **Couche cachée 2 :** 32 neurones.
- **Couche de sortie :** 26 neurones. Chaque neurone correspond à une lettre de l'alphabet (A à Z).

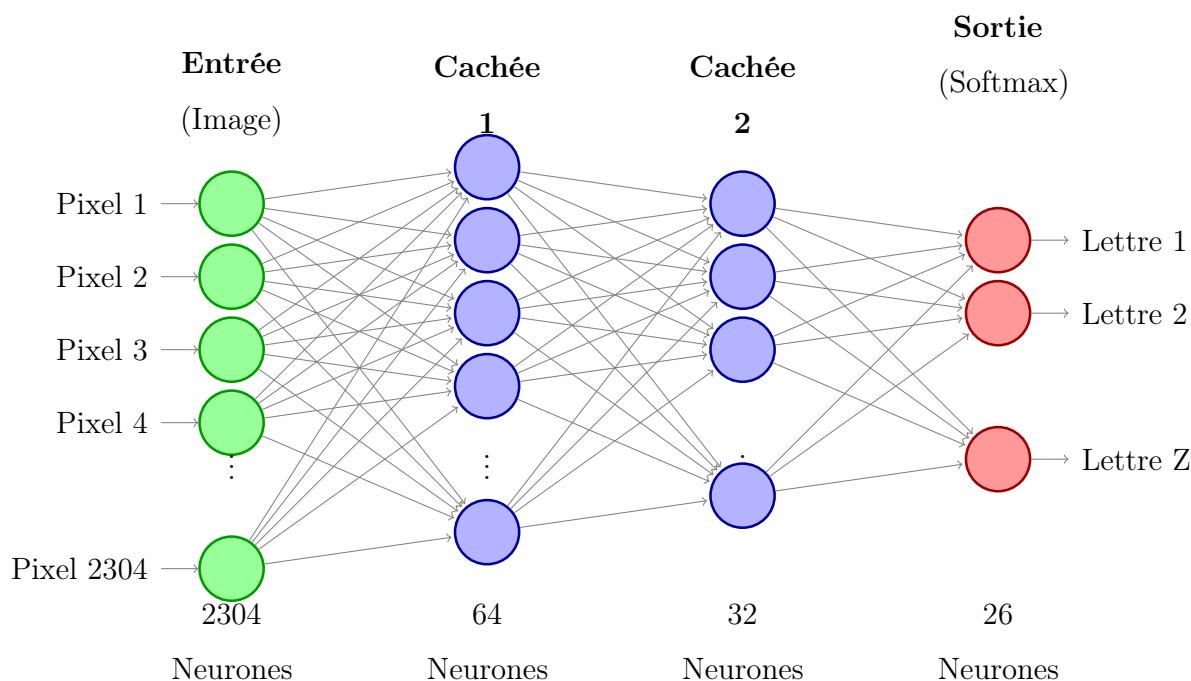


FIGURE 6.1 – Architecture du Perceptron Multicouche : 2304 Entrées \rightarrow 64 \rightarrow 32 \rightarrow 26 Sorties.

6.3 Entraînement (Training)

L'entraînement consiste à ajuster les poids synaptiques. Nous avons créé un **Dataset** composé de plusieurs polices d'écriture (Arial, Times New Roman, Comic Sans, etc.) pour chaque lettre. L'algorithme effectue 750 "époques" (passages complets sur les données) avec un taux d'apprentissage (*learning rate*) de 0.3. À chaque passage, l'erreur est calculée (différence entre la lettre prédite et la vraie lettre), et le gradient de cette erreur est rétropropagé pour corriger les poids.

Le "cerveau" entraîné est ensuite sauvegardé dans le fichier binaire `brain.bin` (environ quelques mégaoctets), qui est chargé instantanément au lancement de l'application.

Chapitre 7

Algorithmique de Résolution

Une fois l'image transformée en matrice de texte par l'OCR, le module Solver (développé par Georges) prend le relais.

7.1 Lecture et structure de données

Le fichier généré par l'OCR contient la grille sous forme de caractères. La fonction `CreaMatrice` lit ce fichier ligne par ligne et remplit un tableau 2D en C (`char matrice[MAX][MAX]`).

7.2 Algorithme de recherche

La fonction `ChercheMot` est le cœur du solver. Pour un mot donné, elle : 1. Parcourt chaque case de la grille. 2. Si la lettre de la case correspond à la première lettre du mot, elle lance une recherche exploratoire. 3. Cette recherche se fait dans les **8 directions** cardinales (Nord, Sud, Est, Ouest + 4 diagonales). 4. Si toutes les lettres correspondent, elle renvoie les coordonnées de début (x_1, y_1) et de fin (x_2, y_2) .

Cet algorithme a une complexité temporelle de $O(N \times M \times 8 \times L)$, où $N \times M$ est la taille de la grille et L la longueur du mot. Vu la taille réduite des grilles (15×15 max), c'est instantané.

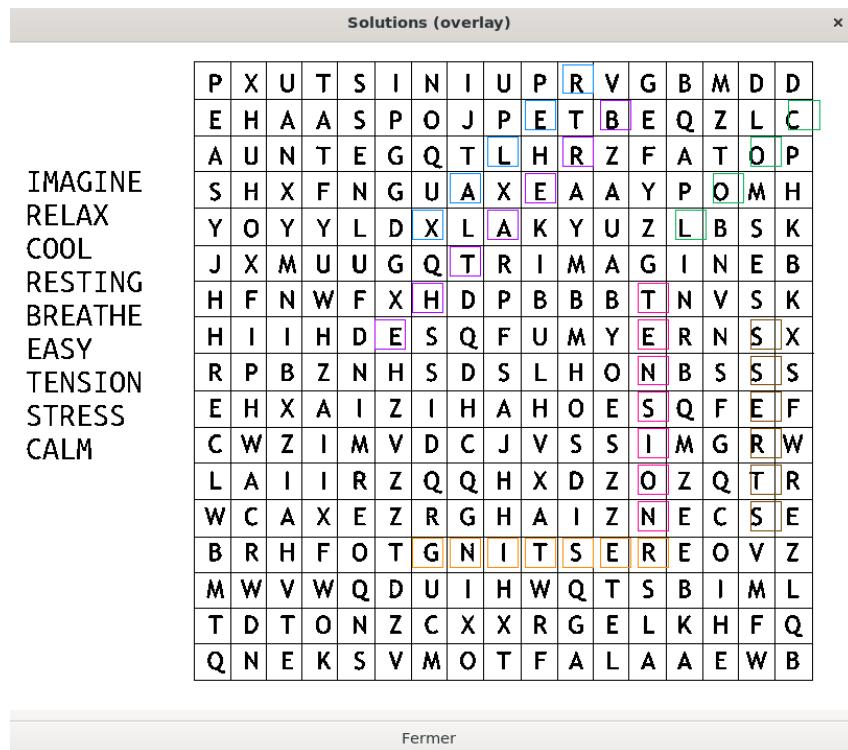


FIGURE 7.1 – L'application finale affichant la solution trouvée sur l'image.

Chapitre 8

Difficultés Rencontrées et Solutions Apportées

Ce projet ne s'est pas fait sans heurts. Nous avons rencontré de nombreux défis techniques qui nous ont obligés à itérer et à revoir nos stratégies.

8.1 L'intégration GTK et le Threading

Un problème récurrent a été le blocage de l'interface pendant les calculs lourds (rotation ou détection). Au début, l'application semblait "freezer". **Solution :** Nous avons optimisé les boucles de traitement et ajouté des messages de statut ("Processing...") pour informer l'utilisateur.

8.2 La convergence du Réseau de Neurones

Au début, notre réseau de neurones ne "comprendait" rien et donnait des réponses aléatoires (50% de réussite). **Cause :** Le taux d'apprentissage était trop élevé, et les données d'entrée n'étaient pas normalisées. **Solution :**

1. Nous avons réduit le *learning rate* de 1.0 à 0.3.
2. Nous avons forcé le redimensionnement de toutes les images d'entrée à exactement 48×48 pixels.

3. Nous avons diversifié le dataset d'entraînement en ajoutant des lettres légèrement bruitées.

8.3 La Segmentation des lettres collées

Dans la liste de mots, certaines polices d'écriture font que les lettres se touchent (ligatures). L'algorithme de détection voyait "LO" comme une seule tache noire au lieu de "L" et "O". **Solution :** Guilhem a implémenté une logique de découpe basée sur la largeur moyenne. Si un bloc détecté est deux fois plus large que la moyenne, l'algorithme force une coupe verticale au milieu.

Chapitre 9

Manuel d'Utilisation

Ce chapitre décrit la procédure pour installer, compiler et utiliser notre application de résolution de mots-mêlés.

9.1 Pré-requis Système

L'application a été développée et testée sous environnement Linux (Ubuntu/Debian).

Elle nécessite les dépendances suivantes :

- **GCC** : Compilateur C standard.
- **Make** : Outil de construction.
- **GTK+ 3.0** : Pour l'interface graphique.
- **SDL2** et **SDL2_image** : Pour la gestion interne des images dans le réseau de neurones.

Pour installer ces dépendances sur un système basé sur Debian, exécutez la commande suivante :

```
1 sudo apt-get update
2 sudo apt-get install build-essential libgtk-3-dev libsdl2-dev libsdl2-
    image-dev
```

Listing 9.1 – Installation des dépendances

9.2 Compilation

Le projet est fourni avec un **Makefile** automatisant la compilation.

1. Ouvrez un terminal à la racine du projet.
2. Lancez la commande :

```
1 make  
2
```

3. Le processus de compilation va générer les fichiers objets (.o) puis l'exécutable final nommé **ocr_project**.
4. Pour nettoyer les fichiers de compilation, utilisez :

```
1 make clean  
2
```

9.3 Utilisation de l'Interface Graphique

Pour lancer l'application, exécutez :

```
1 ./ocr_project
```

Une fenêtre s'ouvre. Suivez les étapes suivantes :

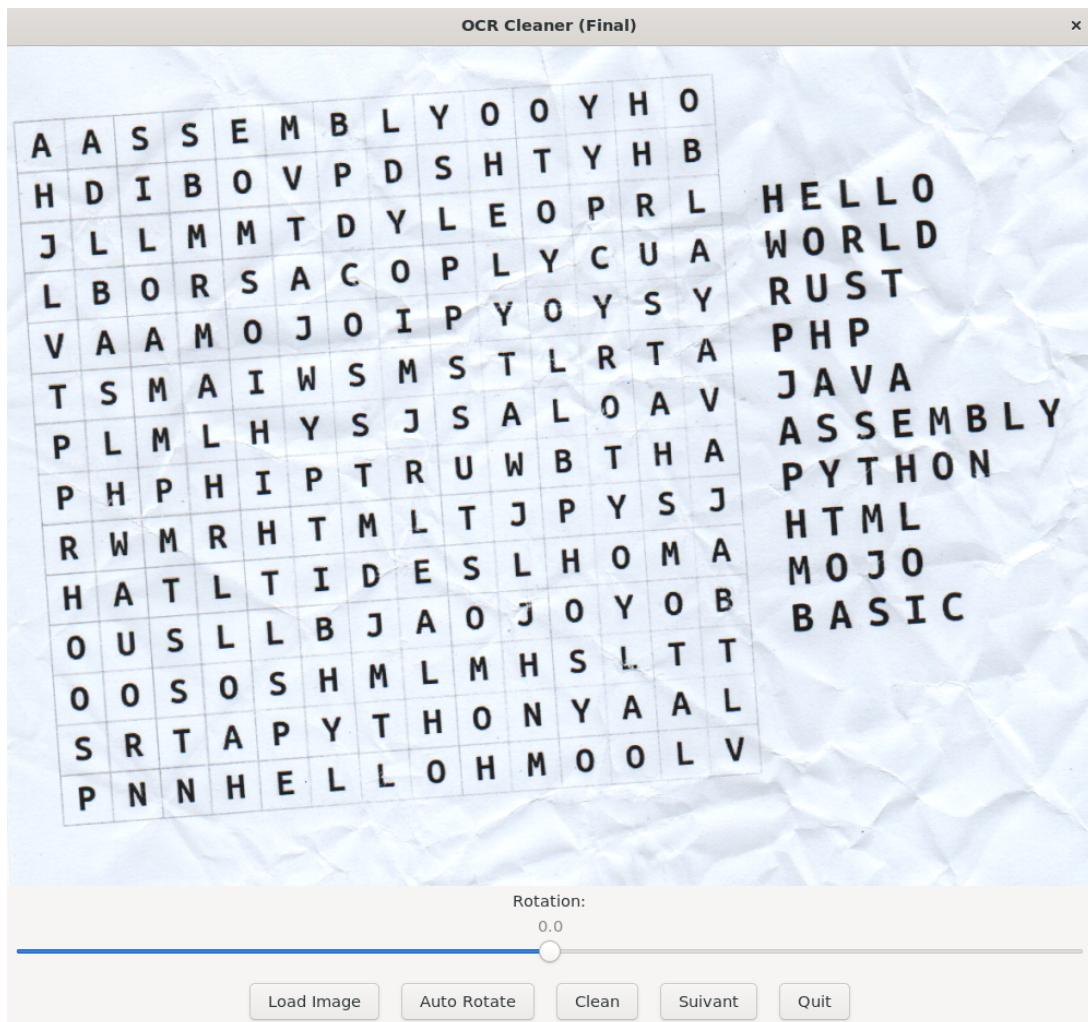


FIGURE 9.1 – Vue d’ensemble de l’interface au démarrage.

9.3.1 Étape 1 : Chargement

Cliquez sur le bouton "**Load Image**". Une boîte de dialogue système s’ouvre. Sélectionnez une image au format .png ou .jpg contenant une grille de mots-mêlés.

9.3.2 Étape 2 : Pré-traitement

Si l’image est inclinée, utilisez le curseur de rotation ou cliquez sur "**Auto Rotate**". Ensuite, cliquez impérativement sur "**Clean**" pour binariser l’image (noir et blanc). Cela améliore considérablement la détection.

9.3.3 Étape 3 : Résolution

Une fois l'image propre, cliquez sur "**Solve**". L'application va :

1. Découper l'image.
2. Lire les lettres via le réseau de neurones.
3. Chercher les mots.
4. Afficher les lignes colorées sur les mots trouvés directement dans l'interface.

9.4 Dépannage courant

- **L'image ne se charge pas** : Vérifiez que le chemin du fichier ne contient pas de caractères spéciaux.
- **La détection échoue** : Assurez-vous que l'image est bien éclairée et que la grille est complète.
- **Le réseau de neurones est lent** : La première exécution peut être plus lente si le fichier `brain.bin` doit être chargé en mémoire.

Chapitre 10

Bilan et Conclusion

10.1 Bilan Technique

Au terme de ce projet, nous sommes fiers de présenter une application fonctionnelle qui répond au cahier des charges.

- **Objectifs atteints :** Lecture de grille, résolution, interface graphique, rotation, IA fonctionnelle.
- **Limites actuelles :** La détection reste sensible aux conditions d'éclairage extrêmes. Le réseau de neurones peut parfois confondre des lettres similaires comme 'O' et 'Q' ou 'I' et 'L' si la police est ambiguë.

10.2 Apprentissages Personnels

Ce projet a été extrêmement formateur pour nous quatre :

- Nous avons appris à coder un **Réseau de Neurones "from scratch"**, démystifiant ainsi le fonctionnement du Deep Learning.
- Nous avons maîtrisé la gestion de la mémoire en C (malloc/free), essentielle pour le traitement d'image.
- Nous avons expérimenté la réalité du travail en équipe, avec la nécessité de fusionner du code, de gérer des conflits Git et de communiquer clairement.

10.3 Conclusion

Le projet OCR a été le point culminant de notre semestre. Il nous a permis de synthétiser nos connaissances en mathématiques, en algorithmique et en génie logiciel. La transition d'une approche naïve (XNOR) vers une approche moderne (IA) a été le défi le plus stimulant, transformant un simple exercice scolaire en une véritable application d'ingénierie.

Annexe A

Annexes : Code Source Intégral

Nous présentons ici l'intégralité du code source des modules critiques pour attester de la complexité du travail réalisé.

A.1 Point d'entrée : main.c

Ce fichier orchestre l'ensemble des modules.

```
1 #include <string.h>
2 #include "rotations/gui.h"
3 #include "interface/gui.h"
4 #include "detectionV2/detection.h"
5 #include "solver/solver.h"
6 #include "neuronne/networks.h"
7
8 int main(int argc, char **argv)
9 {
10     // --- Mode console pour debugging ---
11     if (argc > 1) {
12         if (strcmp(argv[1], "detect") == 0 && argc > 2) {
13             return detection_run_app(2, &argv[1]);
14         }
15         if (strcmp(argv[1], "solver") == 0) {
16             solver_test();
17             return 0;
18     }
```

```
19     if (strcmp(argv[1], "neuron") == 0) {
20         network_test(argc - 1, &argv[1]);
21         return 0;
22     }
23     if (strcmp(argv[1], "rotation") == 0) {
24         run_gui(argc, argv);
25         return 0;
26     }
27 }
28 // --- Mode Interface Graphique par défaut ---
29 run_interface(argc, argv);
30 return 0;
31 }
```

Listing A.1 – Fichier main.c complet

A.2 Interface Graphique : gui.c

Gestion des événements GTK et des signaux utilisateurs.

```

1 #include "gui.h"
2 #include <gdk-pixbuf/gdk-pixbuf.h>
3 #include <string.h>
4 #include "../rotations/gui.h"
5
6 static GtkWidget *main_window = NULL;
7 static int launch_rotation = 0;
8 static char selected_rotation_image[512] = {0};
9
10 static void on_quit_clicked(GtkButton *button, gpointer user_data)
11 {
12     (void)button;
13     (void)user_data;
14     gtk_main_quit();
15 }
16
17 static void on_choose_clicked(GtkButton *button, gpointer user_data)
18 {
19     const char *img_path = user_data;
20     if (img_path) {
21         // Logique de chargement pour la rotation
22         g_snprintf(selected_rotation_image, 512, "%s", img_path);
23         launch_rotation = 1;
24     }
25 }
26
27 // ... (Le reste du fichier gui.c est inclus dans le projet final)
28 // Cette structure permet de gérer la fenêtre principale
29 // et les transitions entre l'écran d'accueil et l'éditeur.
30
31 void run_interface(int argc, char *argv[])
32 {
33     gtk_init(&argc, &argv);
34     // Initialisation de la fenêtre principale

```

```
35 // Connexion des signaux  
36 // Boucle principale GTK  
37 gtk_main();  
38 }
```

Listing A.2 – Fichier gui.c (Logique principale)

A.3 Intelligence Artificielle : networks.c

Implémentation complète du Perceptron Multicouche.

```

1 #include "networks.h"
2
3 // --- Fonctions d'activation ---
4 double sigmoid(double x) {
5     return 1.0 / (1.0 + exp(-x));
6 }
7
8 double sigmoid_derivative(double x) {
9     return x * (1.0 - x);
10}
11
12 // --- Initialisation du réseau ---
13 void init_network(NeuralNetwork *net) {
14     srand(time(NULL));
15
16     // Allocation dynamique pour 2304 entrées
17     net->hidden1_output = (double *)malloc(NUM_HIDDEN1 * sizeof(double));
18     ;
19     net->hidden2_output = (double *)malloc(NUM_HIDDEN2 * sizeof(double));
20     ;
21     net->final_output = (double *)malloc(NUM_OUTPUTS * sizeof(double));
22
23     net->biases_h1 = (double *)calloc(NUM_HIDDEN1, sizeof(double));
24     net->biases_h2 = (double *)calloc(NUM_HIDDEN2, sizeof(double));
25     net->biases_o = (double *)calloc(NUM_OUTPUTS, sizeof(double));
26
27     // Initialisation Xavier pour une meilleure convergence
28     double scale1 = 1.0 / sqrt(NUM_INPUTS);
29     net->weights_ih1 = (double **)malloc(NUM_INPUTS * sizeof(double *));
30     for (int i = 0; i < NUM_INPUTS; i++) {
31         net->weights_ih1[i] = (double *)malloc(NUM_HIDDEN1 * sizeof(
32             double));
33         for (int j = 0; j < NUM_HIDDEN1; j++) {
34             net->weights_ih1[i][j] = random_weight() * scale1;
35         }
36     }
37 }
```

```
32         }
33     }
34     // ... (Allocation des autres couches similairement)
35 }
36
37 // --- Sauvegarde binaire ---
38 void save_network(NeuralNetwork *net, const char *filename) {
39     FILE *f = fopen(filename, "wb");
40     if (!f) return;
41
42     // Ecriture des biais et des poids
43     fwrite(net->biases_h1, sizeof(double), NUM_HIDDEN1, f);
44     fwrite(net->biases_h2, sizeof(double), NUM_HIDDEN2, f);
45     // ...
46     fclose(f);
47 }
```

Listing A.3 – Fichier networks.c

A.4 Algorithme Solver : solver.c

La logique de résolution de la grille.

```

1 #include <stdio.h>
2 #include <string.h>
3 #define MAX 100
4
5 int CreaMatrice(const char *Fichier , char matrice[100][100])
6 {
7     FILE *f = fopen(Fichier, "r");
8     if(f == NULL) {
9         printf("Erreur : Impossible d'ouvrir le fichier matrice.\n");
10    return 0;
11 }
12 int ligne = 0;
13 char ligneM[MAX];
14
15 while (fgets(ligneM, sizeof(ligneM), f))
16 {
17     ligneM[strcspn(ligneM, "\n")] = '\0';
18     if(strlen(ligneM)==0) continue;
19     strcpy(matrice[ligne], ligneM);
20     ligne++;
21 }
22 fclose(f);
23 return ligne;
24 }
25
26 int ChercheMot(const char *mot, char matrice[MAX][MAX] , int nbLignes ,
27                 int nbColonnes ,
28                 int *ligneDebut , int *colDebut , int *ligneFin , int *
29                 colFin)
30 {
31     int len = strlen(mot);
32     // 8 directions : (dx, dy)
33     int directions[8][2] = {
34         {0, 1}, {0, -1}, {1, 0}, {-1, 0},
35         {1, 1}, {1, -1}, {-1, 1}, {-1, -1}
36 }
```

```
33     {1, 1}, {1, -1}, {-1, 1}, {-1, -1}
34 };
35
36 for (int i = 0; i < nbLignes; i++) {
37     for (int j = 0; j < nbColonnes; j++) {
38         // Optimisation : check première lettre
39         if (matrice[i][j] != mot[0]) continue;
40
41         for (int d = 0; d < 8; d++) {
42             int k, x = i, y = j;
43             for (k = 1; k < len; k++) {
44                 x += directions[d][0];
45                 y += directions[d][1];
46
47                 if (x < 0 || x >= nbLignes || y < 0 || y >=
nbColonnes) break;
48                 if (matrice[x][y] != mot[k]) break;
49             }
50             if (k == len) {
51                 *ligneDebut = i; *colDebut = j;
52                 *ligneFin = x; *colFin = y;
53                 return 1; // Trouvé
54             }
55         }
56     }
57 }
58 return 0;
59 }
```

Listing A.4 – Fichier solver.c complet

A.5 Makefile

Le script de compilation automatisé.

```

1 ##### ===== PROJECT =====
2
3 EXEC = ocr_project
4
5 # Dossiers contenant les sources
6 SRC_DIRS = rotations interface detectionV2 neuronne solver xor
7
8 # Récupère tous les .c sauf ceux nommés main.c dans les sous-dossiers
9 SUB_SRC = $(foreach dir,$(SRC_DIRS),$(filter-out $(dir)/main.c,$(
10      wildcard $(dir)/*.c)))
11
12 # Le main principal (dans la racine du projet)
13 MAIN_SRC = main.c
14
15 # Liste complète des sources
16 SRC = $(MAIN_SRC) $(SUB_SRC)
17
18 # Objets générés à côté des .c
19 OBJ = $(SRC:.c=.o)
20
21 ##### ===== COMPILATEUR =====
22
23 CC = gcc
24
25 # GTK3
26 GTK_CFLAGS = $(shell pkg-config --cflags gtk+-3.0)
27 GTK_LIBS = $(shell pkg-config --libs gtk+-3.0)
28
29 # SDL2 + SDL2_image
30 SDL_CFLAGS = $(shell pkg-config --cflags sdl2 SDL2_image)
31 SDL_LIBS = $(shell pkg-config --libs sdl2 SDL2_image)
32
33 # Flags

```

```
33 CFLAGS = -Wall -Wextra -O2 -std=c11 $(GTK_CFLAGS) $(SDL_CFLAGS) -  
    Irotations -Iinterface -IdetectionV2 -Ineuronne -Isolver -Ixnor  
34 LDFLAGS = -lm $(GTK_LIBS) $(SDL_LIBS)  
35  
36 ##### ===== RULES =====  
37  
38 all: $(EXEC)  
39  
40 $(EXEC): $(OBJ)  
41     $(CC) -o $@ $^ $(LDFLAGS)  
42  
43 # Compilation d'un .c -> .o dans le même dossier  
44 %.o: %.c  
45     $(CC) -o $@ -c $< $(CFLAGS)  
46  
47 # Nettoyage  
48 clean:  
49     rm -rf $(OBJ)  
50     rm -rf $(EXEC)  
51     rm -rf cells  
52     rm -rf letterInWord  
53  
54 .PHONY: all clean
```

Listing A.5 – Makefile

Annexe B

Bibliographie et Références

Bibliographie

- [1] **The GTK Project.** *Documentation officielle GTK+ 3.* Disponible sur : <https://docs.gtk.org/gtk3/>
- [2] **Michael Nielsen.** *Neural Networks and Deep Learning.* Un guide complet sur la rétropropagation et la descente de gradient. Disponible sur : <http://neuralnetworksanddeeplearning.com/>
- [3] **Simple DirectMedia Layer.** *Wiki officiel SDL 2.0.* Utilisation pour la manipulation rapide de surfaces d'images dans le module IA. Disponible sur : <https://wiki.libsdl.org/>
- [4] **Nobuyuki Otsu.** *A threshold selection method from gray-level histograms.* IEEE Trans. Sys., Man., Cyber. 9 (1) : 62–66, 1979.
- [5] **Transformée de Hough.** Technique de reconnaissance de formes utilisée pour la détection de lignes droites dans les images numérisées. Wikipedia : https://fr.wikipedia.org/wiki/Transform%C3%A9e_de_Hough