EPITA – Projet OCR

Rapport de soutenance

Date : 03/11/2025

Groupe prs B2 G10

Membres du groupe :

Benjamin Raccah	– Module *Loader* et Interface graphique
Georges Savini	– Chef de projet, développement du Solver
Guilhem Petit	– Reconnaissance de caractères (OCR)
Kevin To	– Réseau de neurones (fonction XNOR)

Année 2025

Table des matières

Rôle et mission de chaque membre

Le projet a été réalisé par un groupe de quatre étudiants, chacun ayant pris en charge une partie spécifique du développement, tout en collaborant régulièrement pour assurer la cohérence et l'intégration de l'ensemble. L'objectif global du projet est de créer un **résolveur de mots mêlés à partir d'une image**, capable de détecter automatiquement la grille, de reconnaître les lettres, puis d'y rechercher les mots à trouver.

Benjamin Raccah

Benjamin a conçu et développé le **module de chargement d'images (loader)** ainsi que l'**interface graphique**. Le loader a pour rôle d'effectuer les traitements préliminaires sur les images (redressement, contraste, conversion en noir et blanc), tandis que l'interface permet à l'utilisateur de sélectionner une image, de la visualiser et de lancer le processus de résolution. Son travail a nécessité une bonne maîtrise de la bibliothèque `GTK` et des opérations de manipulation d'images en C.

Georges Savini

Georges a assuré le rôle de **chef de projet**. Il a coordonné l'organisation du groupe, la gestion du dépôt Git et la planification des tâches. Sur le plan technique, il a développé le **solver**, c'est-à-dire le module chargé d'analyser la grille de lettres extraite et de repérer automatiquement les mots à l'intérieur. Son rôle central a permis de garantir la cohérence entre les différentes parties du projet et d'assurer le bon déroulement des livrables.

Guilhem Petit

Guilhem s'est concentré sur la **reconnaissance des caractères** à partir de l'image prétraitée. Il a conçu la partie du programme permettant d'identifier chaque lettre grâce à des techniques d'analyse d'image et de classification basées sur les formes. Ce module constitue un élément essentiel du pipeline de traitement, reliant la sortie du loader à l'entrée du solver.

Kevin To

Kevin a travaillé sur la **preuve de concept du réseau de neurones** en développant une implémentation de la fonction `XNOR`. Ce travail préliminaire a permis de valider la faisabilité d'une approche neuronale pour la reconnaissance de caractères et a servi de base à de futures améliorations du système OCR.

Ainsi, chaque membre a contribué de manière complémentaire, permettant la mise en place d'une solution complète allant du traitement de l'image jusqu'à la détection automatique des mots.

Objectifs, planification et état d'avancement

1 Objectifs du projet

L'objectif principal de notre projet était de concevoir un programme capable d'analyser une grille de mots mêlés à partir d'une simple image, puis d'y rechercher un mot donné. Nous avons donc cherché à combiner plusieurs domaines : le traitement d'image, la reconnaissance de caractères et la recherche algorithmique dans une matrice.

Pour y parvenir, nous avons défini plusieurs sous-objectifs clairs :

- Développer un **loader** chargé de préparer les images pour le traitement (chargement, redressement, amélioration du contraste et conversion en noir et blanc) ;
- Concevoir une **interface graphique** conviviale permettant de charger une image, de la faire pivoter manuellement et de lancer la reconnaissance ;
- Mettre en place un module de **reconnaissance de caractères** reposant sur la fonction logique `XNOR`, afin d'interpréter le contenu de la grille ;
- Implémenter un **solver** capable de rechercher automatiquement un mot dans toutes les directions possibles (horizontales, verticales et diagonales) ;
- Expérimenter une **correction automatique de l'inclinaison** de la grille (deskew), fonctionnalité dont la version finale reste partielle.

2 Planification du projet

Dès le lancement du projet, nous avons établi une feuille de route découpée en plusieurs étapes : conception du pipeline global, développement de chaque module, puis intégration et tests. Notre méthode de travail reposait sur un cycle simple : planification → implémentation → test → intégration. Nous avons régulièrement ajusté nos priorités en fonction des difficultés rencontrées et du temps disponible.

Le développement a débuté par la mise en place du module de chargement d'images et de conversion en noir et blanc, nécessaires à la reconnaissance de caractères. Nous avons ensuite travaillé sur l'interface graphique, permettant d'effectuer des rotations manuelles et d'afficher le résultat du traitement en temps réel. Une fois cette partie stabilisée, nous

avons concentré nos efforts sur la reconnaissance de caractères et sur le solver, afin de compléter la chaîne de traitement du projet.

Dans l'ensemble, la plupart des fonctionnalités ont été finalisées dans les délais prévus. La seule partie restée partielle concerne la rotation automatique, qui nécessiterait une meilleure calibration de l'algorithme d'analyse d'angle.

3 État d'avancement et description des modules

À la fin du développement, l'ensemble du pipeline est fonctionnel, depuis le chargement de l'image jusqu'à la détection automatique des mots dans la grille.

Le module `loader`

Le loader assure le chargement et le prétraitement des images. Nous avons utilisé la bibliothèque `GdkPixbuf` pour sa compatibilité native avec les formats PNG et JPEG. Le module applique plusieurs filtres successifs :

- `enhance_contrast()` pour accentuer la lisibilité des lettres ;
- `blur_image()` pour lisser les contours et améliorer la détection de l'orientation ;
- `to_black_and_white()` pour convertir l'image via un seuillage adaptatif calculé automatiquement.

Une fonction `deskew_image()` a également été implémentée afin de corriger l'inclinaison de la grille. Elle repose sur une estimation de l'angle dominant, mais son comportement reste encore à affiner pour certaines images.

Détection des lettres et des cellules

Le programme analyse l'image pour identifier automatiquement les lettres présentes dans la grille de mots mêlés. Il distingue les deux zones principales : la liste de mots et la grille, puis encadre chaque lettre ou cellule à l'aide de rectangles colorés. Chaque lettre est préparée sous forme d'image individuelle, prête pour la reconnaissance via le module OCR.

Cette étape est cruciale car elle relie le prétraitement du loader à la reconnaissance des caractères et permet au solver de fonctionner correctement.

L'interface graphique

Nous avons développé une interface graphique en `C` avec la bibliothèque `GTK`. Elle permet à l'utilisateur de sélectionner une image, de la visualiser, de la faire pivoter manuellement grâce à un curseur, puis de lancer le traitement complet. La rotation s'effectue grâce à une fonction personnalisée, `rotate_pixbuf_any_angle()`, qui applique une transformation géométrique sur chaque pixel. Cette approche offre une rotation fluide et

précise, tout en garantissant un affichage en temps réel. L'interface constitue ainsi un outil ergonomique et intuitif, facilitant l'expérimentation et les tests du projet.

La reconnaissance de caractères

Le cœur du système OCR repose sur une méthode de comparaison binaire fondée sur la fonction logique `XNOR`. Nous avons choisi cette approche simple mais efficace pour comparer des matrices de pixels représentant des lettres. Chaque caractère est représenté sous forme binaire (noir = 0, blanc = 1), et la fonction `XNOR` mesure la similarité entre la lettre à reconnaître et les modèles enregistrés. Cette méthode permet d'obtenir de bons résultats sur des images bien contrastées, sans nécessiter de bibliothèque d'apprentissage automatique.

Le solver de mots mêlés

Le solver constitue la dernière étape du pipeline. Il reçoit en entrée une matrice de lettres (issue de la reconnaissance) et un mot à rechercher. L'algorithme parcourt la grille dans les huit directions possibles et renvoie les coordonnées de départ et d'arrivée du mot lorsqu'il est trouvé. Pour simplifier la manipulation, toutes les lettres sont converties en majuscules avant la recherche. L'ensemble est optimisé pour fonctionner rapidement, même sur des grilles de grande taille.

4 Bilan de l'avancement

Nous considérons que la grande majorité des objectifs initiaux du projet ont été atteints. Toutes les étapes clés — chargement, interface, conversion, reconnaissance et recherche — sont opérationnelles et intégrées entre elles.

La détection automatique des lettres fonctionne de manière satisfaisante sur les images simples et bien contrastées, mais elle n'est pas encore pleinement fonctionnelle : elle détecte principalement les lettres du premier niveau de la grille et peut en ignorer certaines dans les zones plus denses ou mal séparées. Cette limitation n'empêche pas le bon déroulement du reste du pipeline, mais constitue une piste d'amélioration prioritaire pour la suite du développement.

Enfin, la rotation automatique reste partiellement aboutie et nécessite un meilleur calibrage de l'algorithme d'analyse d'angle. Malgré ces points perfectibles, le programme est stable, modulaire et répond aux attentes du sujet, en produisant des résultats cohérents sur la plupart des jeux de tests.

Choix technologiques, difficultés et solutions

1 Choix technologiques

Dès le début du projet, nous avons fait le choix de développer l'ensemble du programme en **langage C**. Ce choix s'explique à la fois par les contraintes pédagogiques du module et par notre volonté de travailler à un niveau bas pour mieux comprendre les mécanismes de traitement d'image et de manipulation mémoire. Le C nous a permis d'obtenir un contrôle précis sur les structures de données et les performances, tout en respectant une architecture modulaire.

Pour l'interface graphique, nous avons retenu la bibliothèque **GTK**, adaptée au développement d'interfaces légères et compatibles avec les systèmes Unix. Elle offre une intégration fluide avec le C et dispose de fonctions avancées pour la gestion des événements, des widgets et des images via le module `GdkPixbuf`. Ce choix nous a permis d'offrir une interface claire et fonctionnelle, tout en restant proche de l'environnement système.

Concernant la gestion des images, nous avons utilisé la librairie `GdkPixbuf` pour le chargement, la rotation et la conversion des fichiers au format `.png` ou `.jpeg`. Les opérations de transformation (conversion en niveaux de gris, binarisation, rotation d'angle arbitraire) ont été implémentées manuellement, afin de garder une maîtrise totale sur le processus de traitement.

Pour la partie reconnaissance de caractères, nous avons privilégié une approche simple mais efficace, fondée sur la comparaison pixel à pixel via la fonction `XNOR`. Plutôt que d'entraîner un réseau de neurones complet, nous avons choisi de commencer par une preuve de concept permettant de comparer directement la similarité entre une lettre de la grille et un modèle stocké. Cette méthode a constitué une première étape vers une reconnaissance automatique plus avancée.

Enfin, la recherche de mots dans la grille a été réalisée à l'aide d'un **algorithme déterministe** parcourant la matrice dans les huit directions possibles. Ce choix a été motivé par sa simplicité, sa fiabilité et sa rapidité, adaptées à la taille réduite des grilles testées.

2 Difficultés rencontrées

Au cours du développement, plusieurs difficultés techniques se sont présentées :

- **Manipulation d'images et gestion mémoire :** le traitement pixel par pixel en C nécessite une rigueur particulière, notamment pour éviter les dépassements de mémoire et garantir la cohérence des pointeurs sur les buffers d'image.
- **Rotation d'images :** implémenter une rotation d'angle quelconque sans perte de qualité s'est révélé complexe. Nous avons dû interpoler la position des pixels et recentrer l'image pour éviter les décalages visuels.
- **Seuil de binarisation :** la conversion en noir et blanc dépend fortement du contraste et de la luminosité de l'image source. Nous avons donc testé plusieurs méthodes de calcul de seuil avant d'opter pour un *seuil adaptatif* calculé sur l'ensemble de l'image.
- **Intégration des modules :** bien que chaque partie fonctionne isolément, l'interconnexion entre le loader, l'OCR et le solver a nécessité plusieurs ajustements pour harmoniser les formats de données.
- **Détection des lettres et cellules :** La reconnaissance des zones et des lettres a posé plusieurs problèmes :
  - les lignes de la grille étaient souvent détectées en double,
  - certaines zones de mots étaient mal délimitées,
  - des lettres collées étaient parfois confondues,
  - les seuils fixes donnaient des résultats incohérents selon les images.
- **Lecture et insertion d'un fichier texte dans une matrice :** La lecture du contenu d'un fichier texte pour construire la grille de mots s'est révélée plus complexe que prévu. Il fallait récupérer chaque caractère du fichier, détecter les changements de ligne et ignorer les retours à la ligne pour placer correctement les lettres dans une matrice bidimensionnelle. De plus, la gestion de la fin de fichier et des tailles de lignes variables nécessitait une manipulation rigoureuse des indices et de la mémoire.

3 Solutions apportées

Pour surmonter ces difficultés, nous avons adopté plusieurs stratégies :

- **Modularisation du code :** nous avons séparé le projet en modules indépendants (`loader`, `ocr`, `solver`, `gui`) afin de faciliter les tests unitaires et le débogage.
- **Utilisation de structures adaptées :** des structures de données simples mais robustes ont été utilisées pour représenter les matrices de caractères et les images binaires.
- **Mise en place de fonctions utilitaires :** des fonctions communes, comme la normalisation de taille ou la conversion en majuscules, ont été mutualisées pour

éviter la redondance et garantir la cohérence.

— **Tests progressifs :** chaque module a été validé individuellement avant d'être intégré, ce qui a permis de détecter rapidement les erreurs d'interfaçage.

— **Amélioration du rendu visuel :** des ajustements dans les filtres de contraste et de flou ont permis d'améliorer significativement la lisibilité des grilles traitées.

— **Amélioration de la détection des lettres :** Pour résoudre les problèmes de détection, nous avons :
  — fusionné les lignes trop proches pour éviter les doublons,
  — ajouté des marges automatiques autour des zones détectées,
  — ajusté les seuils et l'espacement minimal entre lettres,
  — utilisé des seuils adaptatifs basés sur la densité moyenne de l'image.

  Ces améliorations ont rendu la détection des lettres stable et précise, préparant efficacement les données pour le module OCR.

— **Lecture du fichier texte et construction de la matrice :** Nous avons choisi de lire le fichier texte ligne par ligne à l'aide de `fgets()`. Chaque ligne est ensuite copiée dans la matrice après exclusion du caractère de retour à la ligne (`'\n'`). Cette méthode garantit une correspondance correcte entre le contenu du fichier et la matrice de lettres, facilitant ensuite la recherche des mots par le solver.

En dépit des difficultés rencontrées, ces solutions nous ont permis d'obtenir un programme stable, modulaire et opérationnel. Chaque étape de développement a renforcé notre compréhension du traitement d'image et de la reconnaissance de formes en langage C, tout en consolidant notre travail d'équipe et notre organisation de projet.

Illustrations du projet

```c
int CreaMatrice(const char *Fichier , char matrice[100][100])
{
    FILE *f = fopen(Fichier, "r");
    if(f == NULL)
    {
        printf("impossible d ouvrire le fichier");
        return 0;
    }
    int ligne = 0;
    char ligneM[MAX];

    while (fgets(ligneM, sizeof(ligneM), f))
    {
        ligneM[strcspn(ligneM,"\n")] = '\0'; //supp \n
        if(strlen(ligneM)==0)
        {
            continue;
        }
        strcpy(matrice[ligne], ligneM);
        ligne++;
    }

    fclose(f);

    return ligne;

}
```

FIGURE .1 – fonction qui crée la matrice de lettres

```c
static GdkPixbuf *rotate_pixbuf_any_angle(GdkPixbuf *src, double angle_deg)
{
    int src_w = gdk_pixbuf_get_width(src);
    int src_h = gdk_pixbuf_get_height(src);
    int channels = gdk_pixbuf_get_n_channels(src);
    int rowstride = gdk_pixbuf_get_rowstride(src);
    guchar *src_pixels = gdk_pixbuf_get_pixels(src);

    double angle = angle_deg * M_PI / 180.0;
    double cos_t = cos(angle);
    double sin_t = sin(angle);

    int dest_w = src_w;
    int dest_h = src_h;
    int dest_rowstride = dest_w * channels;
    guchar *dest_pixels = g_malloc0(dest_rowstride * dest_h);

    double cx = (src_w - 1) / 2.0;
    double cy = (src_h - 1) / 2.0;

    for (int y = 0; y < dest_h; y++)
    {
        for (int x = 0; x < dest_w; x++)
        {
            double dx = x - cx;
            double dy = y - cy;
            double sx = cos_t * dx + sin_t * dy + cx;
            double sy = -sin_t * dx + cos_t * dy + cy;
```

FIGURE .2 – fonction pour rotationner l'image dans l'interface 1/2

```c
            double sx = cos_t * dx + sin_t * dy + cx;
            double sy = -sin_t * dx + cos_t * dy + cy;

            int isx = (int)floor(sx);
            int isy = (int)floor(sy);

            for (int c = 0; c < channels; c++)
            {
                guchar val = 255;
                if (isx >= 0 && isy >= 0 && isx < src_w && isy < src_h)
                    val = src_pixels[isy * rowstride + isx * channels + c];

                dest_pixels[y * dest_rowstride + x * channels + c] = val;
            }
        }
    }

    return gdk_pixbuf_new_from_data(
        dest_pixels, GDK_COLORSPACE_RGB,
        channels == 4, 8,
        dest_w, dest_h,
        dest_rowstride,
        free_pixbuf_data, NULL);
}
```

FIGURE .3 – fonction pour rotationner l'image dans l'interface 2/2

Bibliographie

[1] Documentation officielle GTK : *The GTK Project*. Disponible sur : https://docs.gtk.org/

[2] Documentation de la librairie GdkPixbuf : *Image loading and manipulation library*. Disponible sur : https://developer-old.gnome.org/gdk-pixbuf/

[3] Tutoriel OpenClassrooms – *Introduction au traitement d'image en C*. Disponible sur : https://openclassrooms.com/

[4] Article sur la détection et la correction d'inclinaison (deskew) : *Image Deskewing Algorithm – Stack Overflow discussion*. Disponible sur : https://stackoverflow.com/questions/579654/

[5] Article Wikipédia – *Reconnaissance optique de caractères (OCR)*. Disponible sur : https://fr.wikipedia.org/wiki/Reconnaissance_optique_de_caract%C3%A8res

[6] Documentation sur la fonction logique XNOR : *XNOR Gate – Logic and Truth Table*. Disponible sur : https://www.electronics-tutorials.ws/logic/xnor-gate.html

Conclusion

Ce projet nous a permis de mettre en pratique de nombreuses compétences acquises au cours de notre formation, tout en découvrant les enjeux techniques liés à la reconnaissance de caractères et au traitement d'images. En partant d'une simple image de mots mêlés, nous avons construit une chaîne de traitement complète, allant du chargement et de la préparation de l'image jusqu'à la recherche automatique de mots dans la grille.

Le choix du langage C et de la bibliothèque GTK nous a confrontés à des problématiques concrètes de gestion mémoire, d'optimisation et d'interfaçage entre les modules. Ces contraintes nous ont poussés à adopter une approche rigoureuse, en structurant notre code de manière modulaire et en testant chaque partie de façon indépendante avant intégration.

Malgré quelques points encore perfectibles, notamment la rotation automatique des grilles, nous sommes satisfaits du résultat obtenu : le programme est fonctionnel, stable et répond aux objectifs que nous nous étions fixés. Ce travail nous a également permis de mieux comprendre les bases des systèmes de reconnaissance de formes et d'envisager de futures améliorations, comme l'intégration d'un véritable réseau de neurones pour la détection des lettres.

Au-delà de l'aspect technique, ce projet a surtout renforcé notre capacité à travailler en équipe, à planifier efficacement et à surmonter les difficultés rencontrées ensemble. Nous avons ainsi acquis une expérience complète, mêlant conception logicielle, résolution de problèmes et collaboration, qui nous sera utile dans nos futurs projets.