Rapport de projet :
OCR Word Search Solver

BERNARDEAU Lucas
MUSEUX Michael
BOUARBI Ayemane
GAULT Matyas

Table des matières

1 Introduction

Le projet que nous entreprenons vise à créer un logiciel capable de résoudre des grilles de mots cachés en utilisant la reconnaissance optique de caractères, ou OCR (Optical Character Recognition). Ce type de logiciel, appelé OCR Word Search Solver, est conçu pour analyser une image contenant une grille de mots, identifier les lettres qui la composent, puis trouver et afficher les mots de manière automatisée. L'OCR Word Search Solver s'inscrit dans un champ plus large de traitement d'images et de reconnaissance de texte, une technologie particulièrement prisée pour sa capacité à convertir des images en informations exploitables par des machines. Ce projet offre un défi technique intéressant, combinant la reconnaissance de caractères et l'analyse d'images avec une interface graphique.

Le développement d'une application d'OCR appliquée aux grilles de mots cachés présente plusieurs enjeux. D'une part, la capture et le traitement de l'image nécessitent des étapes de prétraitement qui consistent à transformer l'image en niveaux de gris, puis en noir et blanc, afin de simplifier l'analyse. Ensuite, l'application doit détecter la structure de la grille, identifier les cases, reconnaître les lettres présentes, et organiser ces données pour permettre la résolution automatique de la grille. La capacité de l'application à identifier chaque caractère avec précision repose en grande partie sur un système d'apprentissage automatique, notamment un réseau de neurones que nous allons intégrer au logiciel. Ce réseau apprendra à détecter les formes spécifiques des lettres présentes dans les grilles, améliorant ainsi la précision et la fiabilité de la solution.

Notre projet comprend également le développement d'une interface graphique qui permet à l'utilisateur de charger facilement une image de grille, de visualiser et ajuster cette image, puis de consulter la grille résolue. L'interface prévoit également une fonctionnalité de sauvegarde de la grille résolue, permettant à l'utilisateur de conserver le résultat obtenu pour un usage ultérieur. Cette interface a été pensée pour être intuitive et facile d'utilisation, rendant notre logiciel accessible même aux utilisateurs peu familiers avec les technologies de traitement d'image.

Enfin, ce rapport de projet décrit en détail les différentes étapes de notre démarche, des aspects techniques aux choix d'implémentation. Nous présentons d'abord l'organisation de notre équipe ainsi que la stratégie d'exécution du projet. Ensuite, nous explorons les principales composantes du logiciel, notamment l'environnement de développement, les options de compilation, les normes de codage, et la gestion du dépôt Git. Une attention particulière est également portée à la répartition des charges entre les membres de l'équipe, aux défis techniques rencontrés et à l'avancement de chaque étape.

Dans ce contexte, l'OCR Word Search Solver représente non seulement une opportunité pour notre équipe de mettre en pratique nos compétences en programmation et en apprentissage automatique, mais il constitue également un projet innovant qui pourrait être appliqué à d'autres domaines de la reconnaissance de texte. À travers cette application, nous explorons non seulement les bases de la reconnaissance de caractères, mais aussi des aspects avancés du traitement d'image et de l'apprentissage supervisé, avec l'objectif d'offrir une solution complète et performante pour la résolution de grilles de mots cachés.

2 L'entreprise

2.1 Perceptio

Perceptio est une entreprise innovante spécialisée dans les technologies de reconnaissance d'image, fondée en 2024 au sein de l'école d'ingénieurs EPITA. Cette société repose sur un noyau solide de quatre fondateurs talentueux : Michael Museux, Lucas Bernardeau, Ayemane Bouarbi, et Matyas Gault, qui ont uni leurs compétences et leur passion pour l'intelligence artificielle au service de projets de grande envergure. Le rapport présent vous permettra de mieux connaître ces quatre experts et leur rôle respectif au sein de Perceptio.

L'objectif de Perceptio est de concevoir des logiciels intégrant l'intelligence artificielle de manière innovante et performante pour répondre à des problématiques concrètes rencontrées dans divers secteurs. Nos solutions exploitent les dernières avancées en apprentissage profond et en vision par ordinateur pour offrir des outils performants dans des domaines aussi variés que la sécurité, le commerce, la santé et bien d'autres encore. Nous mettons un point d'honneur à créer des produits à la fois intuitifs et performants, capables de percevoir, analyser et interpréter l'information visuelle pour en extraire une valeur ajoutée pour nos utilisateurs.

La culture de Perceptio repose sur des valeurs fondamentales de coopération, de soutien mutuel et d'amitié, des éléments moteurs pour notre équipe. Ces principes guident notre travail quotidien et renforcent notre ambition de créer des logiciels performants et respectueux des besoins des utilisateurs. Le choix du nom Perceptio s'est imposé naturellement à l'unanimité, symbolisant notre engagement commun à créer des solutions de perception visuelle qui repoussent les limites de la technologie et offrent de nouvelles perspectives.

Passons maintenant à la présentation de l'équipe dynamique et passionnée qui est au cœur de ce projet innovant.

2.2 Les membres de l'équipe

Lucas BERNARDEAU

Agé de 19 ans, il nourrit depuis le lycée une passion pour l'informatique, qu'il a explorée en développant divers programmes et scripts en autonomie. À ce jour, il maîtrise plusieurs langages, dont le Python, C, JavaScript, CSS et LaTeX, et a développé une expertise qui s'étend de la conception de bots à la programmation avancée. Ce projet représente pour lui une opportunité d'affiner ses compétences et de perfectionner sa connaissance des langages qu'il pratique déjà.

Par ailleurs, il est animé par un fort intérêt pour la robotique et les systèmes d'intelligence artificielle, domaines dans lesquels il souhaite se spécialiser afin de réaliser des projets personnels. Cette expérience lui permettra de consolider son expertise et de poser les bases nécessaires à la concrétisation de ses objectifs professionnels.

Il a choisi de collaborer avec ses camarades, qu'il considère motivés et animés par des valeurs communes, telles que le sens du travail, l'anticipation et la volonté de donner le meilleur de soi.

MUSEUX Michael

Michaël Museux, âgé de 18 ans, est un élève de deuxième année à l'EPITA, qui partage sa passion pour l'informatique avec ses camarades. En effet, il a touché à Scratch, un langage ludique basé sur des blocs, pour la première fois lorsqu'il n'avait que 10 ans. Depuis, il a développé un grand intérêt pour tous types de langages de programmation, comme le C#, Python, HTML, C, OCamL et même le LaTeX.

L'année précédente, Michaël avait travaillé avec Ayemane sur un projet de jeu vidéo, plus particulièrement un jeu de rôle. Ce fût sa première expérience d'une gestion de projet avec plusieurs membres, permettant de découvrir la coopération et la collaboration lors d'un projet.

Pour ce projet, Michaël est en charge de créer un programme, qui prend une image d'une grille de lettres avec une liste de mots, et qui peut reconnaitre la position de chacune dans l'image, afin de pouvoir couper l'image en deux, pour séparer la grille et la liste.

BOUARBI Ayemane

Ayemane Bouarbi, étudiant de 19 ans en deuxième année à l'EPITA, nourrit depuis son plus jeune âge une véritable passion pour l'informatique et les nouvelles technologies. Cette passion (qui a débuté très tôt avec Scratch) lui a permis de poser les bases de son savoir-faire et de développer un goût pour la résolution de problème. Au fil des années, il a approfondi ses compétences, acquérant une maîtrise solide de langages de programmation tels que le C, C#, C++, Python, OCaml et HTML.

L'année dernière, Ayemane a participé à un projet de groupe où il a contribué au développement d'un jeu de rôle (RPG), renforçant ainsi son expérience en gestion de projet collectif et en structuration de code. Cette première expérience a été marquante dans son parcours, lui permettant de travailler avec des outils collaboratifs et de comprendre les défis de la création d'un projet.

Dans son projet actuel, qui consiste à développer un système de reconnaissance optique de caractères d'une grille de mot (OCR), Ayemane se charge spécifiquement du prétraitement des images, une étape cruciale qui assure la qualité de la reconnaissance des caractères en optimisant les images pour l'analyse. Sa rigueur et son approche méthodique dans cette phase préliminaire contribuent de manière significative au succès du projet. Avec une passion intacte et une expertise technique en constante progression, Ayemane se révèle un élément moteur de l'équipe.

GAULT Matyas

À seulement 18 ans, il a déjà eu l'occasion de participer à divers projets, tel que la création de jeux vidéo, mais n'a jamais eu affaire à des projets aussi complexes. Cela l'a poussé à se dépasser et à en apprendre davantage sur des sujets qui lui étaient encore peu familiers.

Grâce à ses connaissances en C#, python, HTML et en C, il apporte un réel soutien à ses camarades dans la conception des algorithmes et permet au projet d'avancée au rythme convenu.

Matyas nourri un grand intérêt pour tout casse-tête ou puzzle. Un projet tel que celui-ci va alors l'amener à se perfectionner, et à perfectionner ses algorithmes, tout en développant ses connaissances, afin d'avoir des résultats les plus précis possible.

Dans ce projet, Matyas est en charge de la détection et du découpage des lettres dans la liste de mots, afin de pouvoir stocker chacune d'entre elle sous forme d'image, ce qui facilite leur utilisation dans la résolution de la grille.

3 Le projet

3.1 Informations clés sur le projet

Notre projet actuel consiste en la conception et la réalisation d'un logiciel innovant d'OCR (Optical Character Recognition), conçu spécifiquement pour analyser et résoudre des grilles de mots cachés. Cet outil en cours de développement sera capable d'identifier chaque mot et lettre présents dans une grille, et de produire en sortie une image dans laquelle tous les mots trouvés sont clairement mis en évidence. L'objectif principal de ce projet est de créer un logiciel performant capable de lire une image d'une grille de mots cachés, de la traiter, et d'en afficher une version résolue.

L'application fonctionnera en prenant une image de grille de mots en entrée, qu'elle analysera et résoudra automatiquement pour en restituer la grille complète, en y indiquant visuellement les mots trouvés. Nous envisageons également de doter cette application d'une interface graphique ergonomique et intuitive, permettant à l'utilisateur de charger facilement des images au format standard (PDF, JPG). Cette interface permettra non seulement de visualiser l'image chargée, mais aussi d'apporter des corrections visuelles pour améliorer la qualité de l'analyse (par exemple, ajuster la netteté ou corriger l'alignement ou l'orientation), afin d'optimiser la reconnaissance des caractères.

Dans sa version finale, l'application offrira également la possibilité de sauvegarder la grille résolue. Cette fonctionnalité permettra à l'utilisateur de conserver les résultats pour une consultation ultérieure ou pour un partage éventuel avec d'autres utilisateurs.

De surcroît, notre application intégrera une fonctionnalité d'apprentissage automatique. Celle-ci, bien que distincte de l'interface utilisateur principale, jouera un rôle clé dans l'amélioration des performances de l'OCR. Concrètement, cette section du programme permettra d'entraîner notre réseau de neurones sur des grilles de mots cachés variées, renforçant ainsi sa capacité à reconnaître avec précision différents caractères et mots, même dans des grilles de qualité d'image variable. Les données d'apprentissage pourront être sauvegardées et rechargées pour affiner l'algorithme sans nécessiter un nouvel entraînement complet à chaque utilisation.

Ce projet constitue pour notre équipe une opportunité de développer une solution complète et robuste, combinant des compétences en reconnaissance de caractères, en traitement d'image, et en apprentissage automatique. Nous sommes déterminés à proposer une application performante et conviviale, répondant aux attentes des utilisateurs avides de résoudre des grilles de mots cachés de manière automatisée et efficace.

3.2 Procédure d'exécution du projet

Le traitement effectué par notre application sera conçu pour être efficace, garantissant une bonne manipulation des images traitées. Tout commencera par le chargement d'une image depuis des fichiers locaux via l'interface graphique. Une fois l'image chargée, la première étape consistera à supprimer les couleurs. Nous réaliserons d'abord une conversion en niveaux de gris, qui permettra de simplifier l'image en réduisant le spectre des couleurs, suivie d'une transformation en noir et blanc, puis une suppression du bruit dans l'image. Cette conversion sera essentielle, car elle facilitera les étapes ultérieures de détection et d'analyse en mettant en reliefs les contours et les formes des lettres. Cela permettra aussi d'enlever les éléments superflus.

Comme énoncé précédemment, le prétraitement de l'image jouera un rôle crucial dans l'amélioration de la qualité visuelle, car il inclura des ajustements de contraste et de luminosité, ainsi que l'application de filtres visant à réduire le bruit. Une fois ces ajustements réalisés, l'application procédera à la détection de la position de la grille, une étape clé qui permettra de localiser l'aire où se trouveront les lettres à traiter. Ensuite, notre algorithme identifiera également la position de la liste de mots, ce qui permettra de distinguer les éléments de la grille des mots à rechercher, facilitant ainsi l'organisation des données.

Après avoir localisé la grille, l'application détectera les lettres individuelles. À ce stade, nous extrairons les informations pertinentes sous forme d'images. Cela comprendra non seulement les lettres présentes dans la grille, mais aussi les mots affichés dans la liste, ainsi que les lettres qui formeront ces mots. Cette extraction d'images nous aidera pour la phase de reconnaissance de caractères. Nous analyserons les images récupérées pour identifier les lettres.

Une fois les caractères reconnus, nous reconstruirons la grille sous la forme d'un tableau de caractères, tout en reconstituant la liste de mots sous forme de tableau de chaînes de caractères. Cela permettra une manipulation facile et rapide des données traitées. L'application appliquera ensuite des algorithmes spécifiques pour résoudre la grille, déterminant les lettres manquantes en fonction des mots de la liste. Après avoir exécuté cette résolution, la grille sera affichée à l'utilisateur, permettant de visualiser les résultats du traitement.

Enfin, pour garantir une utilisation optimale, nous proposerons une fonctionnalité de sauvegarde permettant à l'utilisateur de conserver la grille résolue dans un format de fichier tel que PNG ou JPG. Ainsi, notre application ne se limitera pas à traiter une image, mais fournira également une solution pour résoudre des grilles de mots.

4 La répartition des charges

Dans le cadre de la réalisation de notre projet, il est impératif de respecter un ensemble de règles et de normes. Pour cela, nous avons identifié plusieurs composantes clés qui devront être mises en œuvre pour garantir le succès et la conformité de notre application.

4.1 Les composantes principales du projet

1. Interface graphique : la création d'une interface graphique est essentielle. Cette interface devra permettre aux utilisateurs d'interagir facilement avec le système, en offrant une expérience utilisateur agréable.
2. Fichier de suppression des couleurs : nous développerons un module spécifique permettant de supprimer les couleurs d'une image. Ce fichier devra être optimisé pour garantir une efficacité maximale dans le traitement des images.
3. Fichier pour rotation manuelle et automatique : il sera nécessaire de concevoir un fichier capable d'effectuer des rotations d'images, tant manuelles qu'automatiques. Cela permettra d'ajuster les images selon les besoins du logiciel.
4. Détection de la position de la grille : un fichier sera dédié à la détection de la position de la grille et de la liste de mots. Ce fichier identifiera les lettres présentes dans la grille ainsi que celles figurant dans la liste de mots.
5. Découpage d'image : nous implémenterons un fichier permettant de découper une image selon des dimensions spécifiées par le logiciel.
6. Algorithme de résolution de grille de mots mêlés : l'implémentation d'un algorithme efficace pour résoudre des grilles de mots mêlés sera un aspect fondamental de notre projet.
7. Compréhension des concepts d'intelligence artificielle : pour démontrer notre compréhension des concepts d'intelligence artificielle, nous créerons une IA capable de résoudre des expressions logiques complexes. Plus précisément, nous mettrons en œuvre une solution pour des expressions du type

   $$\overline{A}.\overline{B} + A.B$$

   Où A et B sont des variables booléennes n'acceptant que les valeurs 0 et 1.
8. Fichier de reconstruction de la grille et de la liste de mots : nous développerons un fichier qui permettra de reconstruire la grille et la liste de mots à partir des données traitées. Ce module inclura également la possibilité de sauvegarder les résultats obtenus pour une consultation ultérieure.

4.2 Pour notre première soutenance

Dans le cadre de cette première soutenance, nous avons défini les objectifs prioritaires sur lesquels nous allons nous concentrer. Ces objectifs comprennent les étapes suivantes :

- Chargement d'une image et suppression des couleurs
- Rotation manuelle de l'image
- Détection de la position :
  - de la grille de mots cachés dans l'image
  - de la liste de mots à trouver
  - des lettres présentes dans la grille
  - des mots contenus dans la liste
  - des lettres dans chacun des mots de la liste
- Découpage de l'image
- Implémentation de l'algorithme de résolution d'une grille de mots cachés
- Réalisation d'une preuve de concept d'un réseau de neurones
- Création d'une interface graphique

4.3 Répartition des rôles

Pour la mise en œuvre de nos objectifs, nous avons soigneusement défini les rôles au sein de notre équipe afin d'optimiser l'efficacité et la synergie entre les membres. Mathias Gault et Michael Museaux ont été désignés pour se charger de la détection des éléments présents dans les images. Leur mission englobe plusieurs aspects essentiels, notamment la localisation de la grille, la détection de la liste de mots, ainsi que l'identification des lettres dans la grille et des mots de la liste. En outre, ils seront responsables du découpage de l'image pour sauvegarder chaque lettre sous forme d'image distincte.

La décision de confier ces tâches à deux personnes s'est fondée sur la complexité du travail à réaliser. En effet, rechercher des éléments sur une image sans disposer d'informations précises, telles que la présence d'une grille pour un mot mêlé, peut s'avérer particulièrement ardu. Par conséquent, il nous est apparu logique de regrouper ces deux activités, car elles sont intrinsèquement liées et nécessitent une coordination étroite. En travaillant ensemble, Mathias et Michael pourront aborder ces défis de manière plus efficace et assurer une meilleure cohérence dans leur approche.

D'autre part, Ayemane Bouarbi se concentrera sur le traitement initial de l'image, notamment le chargement, la suppression des couleurs et la rotation manuelle de l'image. Son intérêt grandissant pour l'analyse et la génération d'images le rend particulièrement apte à relever ces défis techniques. Son rôle est crucial, car il constitue une étape préalable à la détection et à l'analyse des éléments qui seront gérés par Mathias et Michael.

Enfin, Lucas Bernardeau sera responsable de l'implémentation de l'algorithme destiné à résoudre une grille de mots cachés dans le programme en ligne de commande, que nous avons nommé « solver ». En plus de cela, il s'occupera de la réalisation d'une preuve de concept pour notre réseau de neurones, démontrant ainsi son aptitude à apprendre la fonction logique que nous avons définie précédemment. Lucas est également chargé de développer l'interface graphique du projet, ce qui représente un aspect fondamental pour assurer une expérience utilisateur fluide et intuitive. Son intérêt pour la compréhension du fonctionnement des intelligences artificielles, quel que soit le domaine, lui permettra d'apporter une perspective innovante à notre projet.

En résumé, la répartition des rôles au sein de notre équipe repose sur les compétences spécifiques de chaque membre, favorisant ainsi une collaboration efficace et orientée vers l'atteinte de nos objectifs communs. Chaque participant joue un rôle clé dans l'avancement du projet, garantissant que chaque aspect est pris en compte avec rigueur et expertise.

5 Les aspects techniques

5.1 Spécifiés par le cahier des charges

5.1.1 Environnement de développement

Pour la réalisation de notre projet, il est primordial de définir clairement notre environnement de développement. Le code devra être écrit en langage C et doit impérativement être compatible avec les machines mises à disposition par l'école. Cette exigence garantit que tous les étudiants travailleront dans un cadre homogène, ce qui facilite la gestion du projet et les démonstrations. De plus, nous avons l'autorisation d'utiliser tous les logiciels qui sont installés sur ces machines.

5.1.2 Options de compilation et normes de code

Il est essentiel que l'ensemble de notre code compile sans erreurs en utilisant au minimum les options -Wall et -Wextra. Ces options permettent d'activer des avertissements supplémentaires, favorisant ainsi l'identification de problèmes potentiels dès les premières phases du développement. En outre, les lignes de code doivent être limitées à un maximum de 80 caractères afin de garantir une lisibilité optimale. Il est également impératif d'éliminer tout espace inutile en fin de ligne, contribuant ainsi à la propreté du code.

Un autre aspect fondamental est que l'intégralité du code, y compris les noms de variables, les fonctions, ainsi que les commentaires, doit être rédigée en anglais. Cela assure non seulement la compréhension du code par tous les membres de l'équipe, mais facilite également la collaboration avec des développeurs externes ou des évaluateurs qui pourraient ne pas parler français. En respectant ces normes, nous pouvons garantir un niveau de qualité élevé dans notre développement.

5.1.3 Gestion du dépôt git

La gestion de notre dépôt Git est une composante essentielle de notre projet. Une version numérique de notre rapport doit être incluse à la racine du dépôt au format PDF. Il est important de suivre des conventions strictes pour le nommage des fichiers : le rapport de la première soutenance devra être intitulé rapport_1.pdf et devra comporter un minimum de 15 pages. Pour la soutenance finale, le rapport devra être nommé rapport_2.pdf et contenir au moins 40 pages. Il est crucial de respecter ces noms de fichiers, car la récupération des rapports sur le dépôt Git sera automatisée.

En plus du rapport, le dépôt devra contenir l'intégralité du code source du projet, ainsi qu'un fichier Makefile. Ce fichier devra inclure au moins les règles all et clean, permettant ainsi une compilation facile et une gestion simplifiée du projet. Un fichier README est également requis, fournissant des instructions claires sur l'utilisation de l'application. Ce document doit être rédigé dans un format texte qui peut être facilement consulté dans un terminal avec des commandes comme cat ou less, tout en offrant la possibilité d'utiliser également le format Markdown.

5.1.4 Informations sur l'équipe

Il est également nécessaire d'inclure un fichier intitulé AUTHORS, qui doit contenir les logins des membres de l'équipe, présentés sur une ligne par membre selon le format suivant : login (NOM Prénom). Cela assure la transparence et permet de reconnaître la contribution de chaque membre au projet.

5.1.5 Ressources visuelles

Les ressources visuelles jouent un rôle clé dans la présentation de notre projet. Ainsi, le dépôt Git devra contenir quelques images qui serviront à illustrer les différentes parties de notre application. Toutefois, il est recommandé de ne pas surcharger le dépôt avec un trop grand nombre d'images à haute résolution. En général, trois ou quatre images à une résolution adéquate devraient suffire pour atteindre cet objectif. De plus, nous disposerons d'un ensemble d'images nécessaire pour l'apprentissage de notre réseau de neurones.

5.1.6 Exclusions du dépôt

Enfin, il est important de veiller à ce que notre dépôt ne contienne pas de fichiers exécutables, de code source extérieur (comme des bibliothèques ou d'autres projets), ni tout autre type de fichier jugé non nécessaire au projet. En respectant ces directives, nous garantissons un dépôt propre et organisé, facilitant ainsi le travail d'évaluation par les enseignants et la révision par les membres de l'équipe.

5.2 Les bibliothèques et les logiciels utilisés

Dans le cadre de notre projet, nous avons intégré plusieurs technologies et bibliothèques qui ont joué un rôle essentiel dans son développement. Au cœur de notre application se trouvent les bibliothèques SDL (Simple DirectMedia Layer) et SDL_image, qui nous ont permis de gérer la création de fenêtres, l'affichage graphique, et le chargement d'images dans divers formats. Ce choix a été motivé par la large adoption de SDL dans le développement multimédia et sa compatibilité avec différents systèmes d'exploitation, garantissant ainsi une accessibilité optimale.

En complément, nous avons également utilisé GTK (GIMP Toolkit) à travers l'inclusion de <gtk/gtk.h>. GTK est une bibliothèque puissante pour le développement d'interfaces graphiques, qui nous a permis de créer une interface utilisateur intuitive et réactive. L'intégration de GTK a facilité la gestion des événements et des éléments graphiques, nous permettant de concevoir des fenêtres interactives, des boutons, et d'autres composants d'interface utilisateur avec aisance. Ce choix a été particulièrement pertinent pour offrir une expérience utilisateur riche et fluide, essentielle à notre projet.

Pour la gestion des fichiers et la navigation dans les répertoires, nous avons recouru aux bibliothèques standard de C, telles que dirent.h, qui nous ont permis de lire le contenu des répertoires de manière efficace. Cela a facilité le chargement dynamique des images, tout en garantissant une gestion appropriée des erreurs lors de la manipulation des fichiers.

En ce qui concerne notre environnement de développement, nous avons opté pour GIT pour la gestion des versions de notre code. Cet outil nous a permis de suivre les modifications de manière systématique, de collaborer efficacement et de revenir à des versions antérieures si nécessaire. Pour l'édition du code, nous avons choisi Vim, un éditeur puissant qui offre de nombreuses fonctionnalités adaptées aux développeurs. Pour la documentation, Overleaf a été privilégié pour sa convivialité avec LaTeX, facilitant ainsi la mise en forme de nos documents techniques. De plus, la création d'un serveur Discord a été cruciale pour maintenir une communication fluide et coordonnée entre les membres de l'équipe.

6 Synthèse de l'avancement du projet

Nous avons choisi de structurer notre projet en répartissant chacun des aspects dans des fichiers distincts. Cette approche a été adoptée pour faciliter le travail en équipe et simplifier l'implémentation finale. En séparant les différentes fonctionnalités et composants de notre application, nous avons permis à chaque membre de l'équipe de se concentrer sur des tâches spécifiques sans interférer avec le travail des autres. De plus, cette organisation modulaire facilite également le débogage et la maintenance du code, car chaque fichier peut être géré indépendamment.

Voici maintenant une description détaillée de chaque aspect du projet :

6.0.1 Le chargement d'une image et suppression des couleurs

Dans le cadre de ce projet de reconnaissance optique de caractères (OCR), le prétraitement de l'image joue un rôle essentiel pour améliorer la qualité des images avant de passer à l'étape de reconnaissance de texte. Les techniques employées ici — telles que la conversion en niveaux de gris, le flou gaussien, l'augmentation du contraste, et le seuillage adaptatif Sauvola avec l'utilisation d'images intégrales — ont toutes été choisies pour leur efficacité à rendre les caractères plus nets et à réduire les imperfections de l'image. Cette section décrit ces étapes, les bibliothèques utilisées (principalement SDL), et les algorithmes techniques associés, permettant à quiconque de reproduire facilement le projet.

Pour simplifier l'image, nous commençons par la transformer en niveaux de gris, ce qui supprime les informations de couleur, gardant seulement les variations d'intensité lumineuse, qui sont les plus pertinentes pour la reconnaissance de texte. Dans cette implémentation, nous utilisons une formule de pondération standard, basée sur le modèle de luminance, pour calculer l'intensité de gris :

$$\text{Gray} = (\text{Red} \times 0.2126 + \text{Green} \times 0.7152 + \text{Blue} \times 0.0722)$$

Ce calcul permet de donner plus de poids à la composante verte, car l'œil humain est plus sensible à cette couleur. Le résultat est stocké dans chaque pixel de l'image, remplaçant les valeurs de rouge, vert et bleu par cette nouvelle intensité de gris.

Après la conversion en niveaux de gris, une augmentation du contraste est appliquée pour accentuer les différences d'intensité entre les pixels clairs et sombres. Cette opération est nécessaire pour les images présentant des variations faibles de contraste. La méthode utilisée ici consiste à :

- Calculer la moyenne globale d'intensité de tous les pixels de l'image (on utilisera la méthode des images intégrale pour calculer les moyennes plus vite. Les détails de cette méthode sont expliqués dans la partie qui suit).
- Ajuster l'intensité du pixel en fonction de cette moyenne, amplifiant les contrastes selon un facteur défini (ici 10.5).

Le contraste est ainsi calculé pixel par pixel en augmentant la différence entre les niveaux d'intensité locale et la moyenne globale selon la formule suivante :

$$\text{nouvelle\_intensité} = (\text{intensité} - \text{moyenne\_globale}) \times \text{facteur} + \text{moyenne\_globale}$$

où :

- facteur = 10.5

Cela rend les lettres plus visibles, surtout dans les images dans lesquelles les caractères peuvent se fondre dans le fond.

Le flou gaussien est appliqué pour réduire le bruit et lisser les petites imperfections dans l'image. Une matrice noyau de taille 3 fois 3 est utilisée ici, où chaque pixel est recalculé en fonction d'une moyenne pondérée de ses voisins. La matrice utilisée ici est la suivante :

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

Ce filtre applique des poids plus élevés aux pixels proches du centre, réduisant ainsi l'influence des variations extrêmes dans les zones voisines. Cette étape est réalisée, en multipliant les valeurs d'intensité des pixels par le noyau gaussien et en stockant les résultats dans une copie temporaire de l'image, afin de ne pas interférer avec le calcul des autres pixels voisins.

La méthode des images intégrales est utilisée ici pour accélérer le calcul de la moyenne dans des régions locales autour de chaque pixel (ici, la région locale correspond à un carré de 25 par 25 pixels). Cette technique permet de calculer la somme des intensités dans une zone donnée en temps constant, grâce à un tableau d'images intégrales accumulées.

Image Intégrale : Dans une image intégrale, chaque pixel représente la somme de tous les pixels situés à gauche et au-dessus de lui dans l'image d'origine. En d'autres termes, la valeur d'un pixel (x,y) dans l'image intégrale est égale à la somme de toutes les valeurs de pixels de l'image d'origine dans la région allant de (0,0) à (x,y).

Soit une image d'origine I, et l'image intégrale correspondante S, la valeur de chaque pixel dans S est calculée comme suit :

$$S(x, y) = I(x, y) + S(x - 1, y) + S(x, y - 1) - S(x - 1, y - 1)$$

où :
— $I(x, y)$ est la valeur du pixel (x, y) dans l'image d'origine.
— $S(x, y)$ est la valeur du pixel dans l'image intégrale.
— les termes $S(x - 1, y)$, $S(x, y - 1)$, et $S(x - 1, y - 1)$ compensent les sommes déjà calculées pour éviter les redondances.

Une fois l'image intégrale calculée, il est possible d'obtenir la somme des pixels dans n'importe quel rectangle de l'image d'origine en utilisant uniquement quatre accès à l'image intégrale, peu importe la taille du rectangle. Cela réduit considérablement le temps de calcul, passant d'un temps proportionnel à la taille de la région à un temps constant.

Enfin, le seuillage Sauvola, qui utilise une approche adaptative, est appliqué. Contrairement à un seuillage global qui fixe un seuil identique pour chaque pixel, le seuillage Sauvola ajuste le seuil pour chaque pixel en fonction des variations locales d'intensité, ce qui est utile pour traiter des documents avec des variations d'éclairage. La formule de Sauvola pour le seuil adaptatif est :

$$\text{seuil} = \text{moyenne\_locale} \times \left(1 + k \times \left(\frac{\text{écart\_type\_local}}{R} - 1\right)\right)$$

où :
— k est un paramètre d'ajustement (ici, 0.6).
— R est la valeur maximale d'écart type (ici, 128).

Pour optimiser ce calcul, nous utilisons les images intégrales pour calculer rapidement la moyenne et l'écart type dans chaque région autour du pixel.

Un amincissement simple est appliqué pour éliminer le bruit résiduel. Cette étape consiste à transformer les pixels d'intensité élevée (blancs) en blanc s'ils ont au moins six voisins blancs dans un voisinage de 3×3. En pratique, cette méthode simplifie la structure des caractères et élimine une partie du bruit.

Enfin, un débruitage par analyse des régions de pixels noirs connectés est appliqué afin d'éliminer le bruit restant tout en préservant les du texte. Cette méthode repose sur une détection de regroupements de pixels noirs, permettant de distinguer les détails pertinents (ici les lettres) du bruit.

Tout d'abord, chaque pixel noir de l'image est analysé pour compter le nombre de pixels noirs adjacents dans un voisinage immédiat de huit directions (haut, bas, gauche, droite et les quatre coins). Lorsqu'une région de pixels noirs connectés est identifiée, elle est temporairement marquée pour éviter les comptages multiples et garantir une analyse de chaque région comme une entité unique.

Ensuite, deux seuils prédéfinis — un seuil minimum et un seuil maximum de taille — déterminent les actions à effectuer selon la taille des regroupements :

— Les petites zones noires, souvent des artefacts de bruit, sont transformées en pixels blancs, car elles ne représentent pas de caractères significatifs.
— Les zones de grande taille, qui peuvent être des artefacts indésirables (comme la grille de mots croisé), sont également converties en blanc pour faciliter l'analyse.
— Les régions de taille intermédiaire, qui sont potentiellement des lettres, sont conservées en noir.

Ce processus est appliqué sans altérer les zones déjà analysées, permettant de mieux préserver les caractères pertinents tout en réduisant le bruit et en accentuant le contraste des éléments textuels.

Après ce prétraitement, une image plus nette et uniforme est obtenue, avec des caractères accentués et moins de bruit. Pour tester le bon fonctionnement de chaque étape, l'image traitée est affichée à l'aide de la bibliothèque SDL. Cette image est prête pour les étapes qui vont suivre.

6.0.2 La rotation manuelle de l'image

Nous allons maintenant aborder la rotation manuelle des images. L'organisation de ce dossier de projet est présentée en ci-dessous.

```
.
|--- Makefile
|--- main.c
|--- manual_grid_rotation.c
|--- manual_grid_rotation.h
```

Le Makefile que nous avons créé permet de compiler le fichier main.c en utilisant les deux autres fichiers présents dans le répertoire. Nous avons pris la décision de développer un fichier de rotation en C sans fonction main, afin de faciliter son appel dans d'autres fichiers du projet. Pour permettre le test de nos fonctions, nous avons également créé un fichier d'en-tête (.h) qui contient les prototypes des fonctions, ainsi qu'un fichier main.c qui prend une image en paramètre et exécute le fichier de rotation.

Concentrons-nous maintenant sur le fichier le plus crucial, manual_grid_rotation.c. Comme son nom l'indique, ce fichier est dédié à l'exécution de la rotation manuelle des images. Il est composé de quatre fonctions principales. La première fonction est responsable de l'initialisation de l'image. Bien que cette fonction puisse sembler peu utile, elle a été conçue pour faciliter les tests durant la phase de développement. Nous avons constaté qu'il était fastidieux de répéter sans cesse les mêmes lignes de code, ce qui a conduit à la décision de créer cette fonction, que nous avons finalement choisie de conserver.

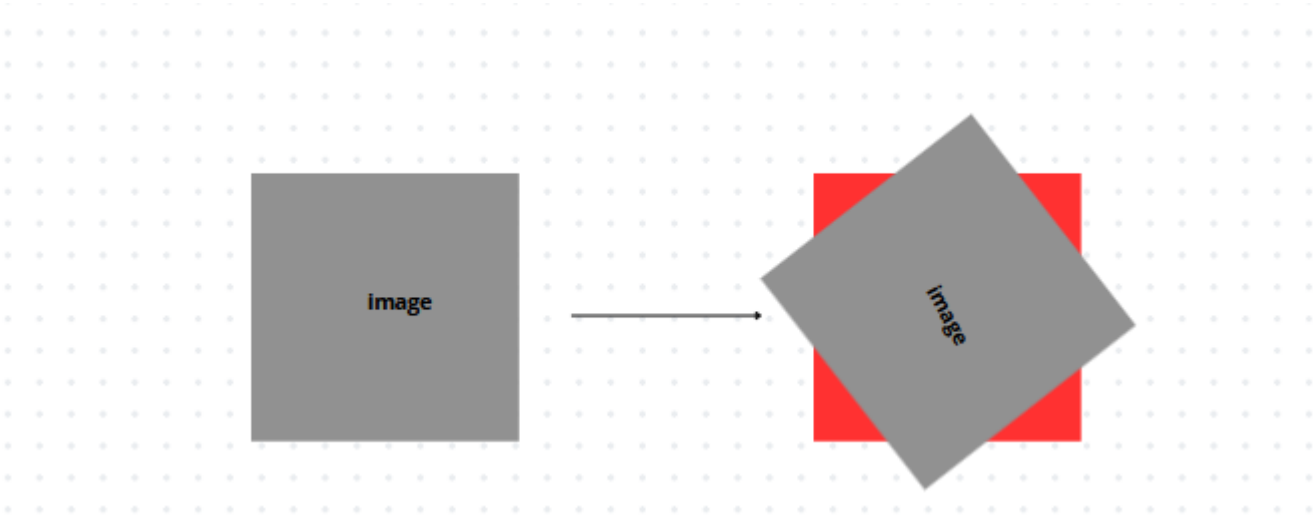
De plus, pour la sauvegarde des images, nous utilisons directement les fonctions fournies par la bibliothèque SDL, ce qui simplifie le processus de chargement et de sauvegarde des images. Cette approche nous permet de nous concentrer sur l'implémentation des fonctionnalités essentielles tout en assurant une gestion efficace des opérations d'entrée/sortie.

La fonction appelée par les autres fichiers est nommée exec_rotation. Initialement, elle représentait la fonction principale du fichier. Cette fonction prend en arguments une image ainsi qu'un angle de rotation, exprimé en degrés. Elle invoque d'abord la fonction nécessaire pour charger l'image, puis appelle la fonction principale de rotation, désignée sous le nom de rotateImage.

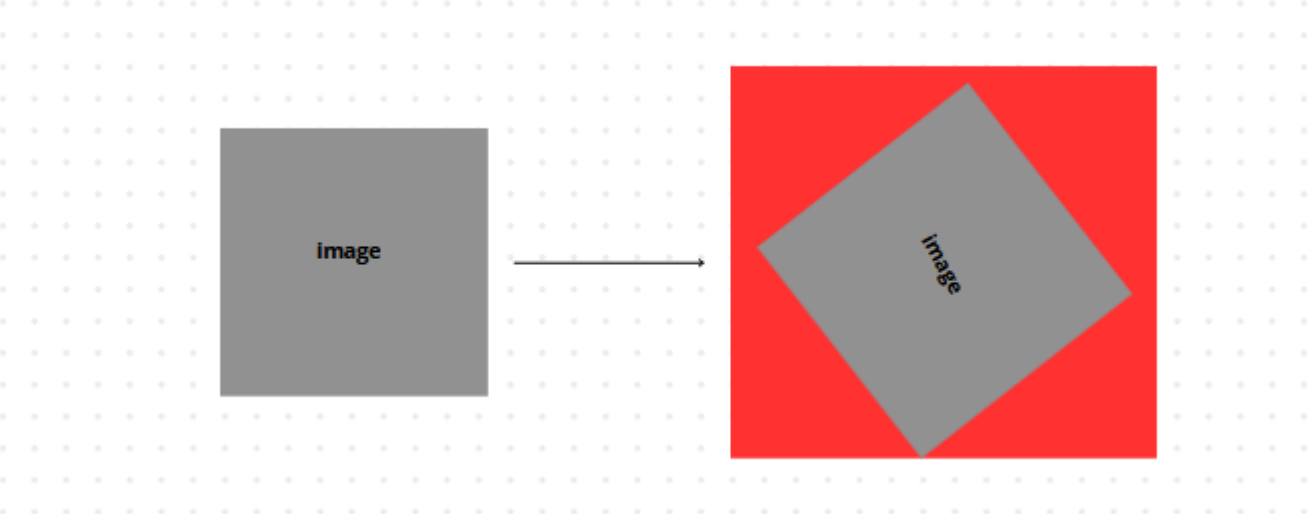
Dans un premier temps, nous calculons les coordonnées de l'image de destination en fonction de l'angle de rotation spécifié. Cela nous permet également de déterminer la taille de l'image résultante. Pour bien appréhender ces principes, nous nous sommes référés à deux sites web : Kongakura [1] et Geeksforgeeks [2].

1. https ://kongakura.fr/article/OpenCV_Python_Tutoriel
2. https ://www.geeksforgeeks.org/turn-an-image-by-90-degree/

Nous calculons ensuite les dimensions de l'image de destination et copions les pixels correspondants depuis l'image d'origine. Toutefois, nous avons rencontré un problème lors de la rotation de 45 degrés, qui entraînait un découpage des images. Pour pallier cette difficulté, nous avons choisi d'ajouter des pixels blancs afin de combler les zones laissées vides en dehors de l'image. Un schéma explicatif illustrant ce processus est présenté en ci-desssous.



Tout d'abord, nous constatons que l'image en niveaux de gris dépasse l'image de destination, bien que ces deux images aient des dimensions identiques. Pour remédier à cette situation, nous avons décidé de créer une image de destination plus grande, afin d'accueillir l'intégralité de l'image source sans perdre de données. Pour faciliter la binarisation ultérieure, nous remplaçons le carré rouge par des pixels blancs, comme illustré ci-dessous.



6.0.3 La rotation automatique

La rotation automatique est une étape clé pour aligner correctement les images de grilles de mots croisés, en particulier dans le cas où l'image n'est pas droite. Cette section explique comment, à partir de l'image binarisée, nous utilisons la transformée de Hough pour détecter l'angle de rotation, puis nous appliquons une rotation pour corriger l'angle de l'image.

Principe de la Rotation Automatique

Une fois l'image binarisée (pixels noirs pour les zones de texte et blancs pour le fond), la rotation automatique consiste à :

— **Identifier les pixels de bord** dans l'image binarisée, c'est-à-dire les pixels noirs ayant au moins un voisin blanc. Ces pixels représentent souvent les contours des caractères ou des lignes.
— **Appliquer la transformée de Hough** pour détecter les lignes dominantes dans l'image. Ces lignes fournissent des indices sur l'orientation générale du texte.
— **Calculer l'angle de rotation** à partir des lignes détectées.
— **Appliquer une rotation** de l'angle calculé pour aligner l'image sur l'un des angles cardinaux : 0°, 90°, 180° ou 270°.

Identification des Pixels de Bord

Un **pixel de bord** est défini comme un pixel noir ($I(x, y) = 0$) ayant au moins un pixel blanc ($I(x', y') = 255$) dans son voisinage 3×3. Cette étape consiste à parcourir l'image binarisée et marquer les pixels qui remplissent cette condition. Ces pixels sont ensuite utilisés comme points d'entrée pour la transformée de Hough.

La Transformée de Hough

La transformée de Hough est une méthode de détection des lignes droites dans une image. Elle fonctionne en transformant chaque point de l'espace cartésien (ici il s'agit de l'image) en une représentation dans un espace de paramètres, où chaque ligne peut être représentée par une équation :

$$r = x \cos\theta + y \sin\theta \tag{1}$$

— r est la distance perpendiculaire de la ligne par rapport à l'origine.
— θ est l'angle entre la ligne perpendiculaire et l'axe horizontal.

Étapes de la Transformée de Hough :

1. **Création de l'Accumulateur :** Un tableau 2D est créé, où chaque cellule (r, θ) compte le nombre de points (pixels de bord) qui peuvent correspondre à une ligne avec ces paramètres.
2. **Itération sur les Pixels de Bord :** Pour chaque pixel (x, y), on calcule les valeurs possibles de r pour un ensemble discret d'angles θ (généralement entre 0° et 180°).
3. **Identification des Pics dans l'Accumulateur :** Les cellules ayant les valeurs maximales dans l'accumulateur correspondent aux lignes dominantes de l'image.

Les angles θ associés aux pics sont ensuite extraits comme les orientations probables des lignes dans l'image.

Calcul de l'Angle de Rotation

Dans le contexte des grilles de mots croisés, les lignes détectées sont souvent alignées horizontalement ou verticalement. Par conséquent :

— Si les lignes dominantes détectées ont un angle proche de 0°, 90°, 180°, ou 270°, l'image est déjà alignée sur un angle cardinal.
— Si un angle intermédiaire est détecté (par exemple, 15° ou 120°), nous déterminons la rotation nécessaire pour ramener l'image à l'angle cardinal le plus proche. Cela simplifie la correction tout en respectant les caractéristiques des grilles.

Application de la Rotation et Alignement de l'image

Une fois l'angle θ déterminé, une rotation est appliquée à l'image binarisée en utilisant une fonction la rotation manuelle développée précédemment.

En raison de l'utilisation de la transformée de Hough, cette méthode aligne systématiquement l'image sur l'un des angles cardinaux (0°, 90°, 180°, ou 270°). Cette limitation est acceptable car elle est gérée dans les étapes ultérieures, qui ajustent l'orientation finale de l'image pour garantir qu'elle est correctement alignée parmi ces quatre directions, même en cas de rotation initiale incorrecte (par exemple, à 180°).

6.0.4 La détection de la position

Le processus commence par aplatir l'image, ce qui consiste à réduire l'image verticalement et horizontalement dans un tableau. Ce tableau a pour dimensions la largeur (pour les aplatissements verticaux) et la hauteur (pour les aplatissements horizontaux) de l'image. Chaque colonne (ou ligne) est analysée : si plus de 98 % de ses pixels sont blancs, la case correspondante dans le tableau est définie comme blanche ; sinon, elle est marquée comme noire. Ensuite, tous les pixels noirs entièrement entourés de pixels blancs sont convertis en pixels blancs.

Après l'aplatissement, le processus consiste à identifier les premiers et derniers pixels noirs dans le tableau. Cette identification est cruciale pour ignorer les colonnes et lignes blanches situées sur les bords de l'image, garantissant ainsi une analyse précise.

Dans la suite du processus, on cherche à déterminer la colonne (ou la ligne) blanche la plus large dans l'intervalle défini par les pixels noirs identifiés précédemment. Cela implique d'identifier la suite la plus longue de pixels blancs consécutifs, tout en calculant également la taille moyenne des colonnes ou lignes blanches.

Une fois ces informations collectées, il est possible de déterminer la position de la liste de mots par rapport à la grille. En examinant les résultats des analyses précédentes pour les aplatissements verticaux et horizontaux, on identifie la colonne ou la ligne ayant le plus grand écart par rapport à sa moyenne de taille respective. Pour établir si la liste de mots se trouve à gauche ou à droite (ou au-dessus ou en dessous) de la grille, il suffit d'observer dans quelle moitié du tableau se situe la colonne ou la ligne blanche la plus large. Si celle-ci se trouve dans la première moitié, cela indique que la liste est à gauche (ou au-dessus) de la grille ; sinon, elle se situe à droite (ou en dessous).

Enfin, le résultat est renvoyé sous forme de coordonnées, en fonction des analyses effectuées aux étapes précédentes.

À partir des coordonnées récupérées, nous entamons une recherche approfondie des lettres, tant dans la grille que dans la liste de mots. Ce processus débute par l'identification du nombre de lignes contenant des lettres, ainsi que par la détermination de leurs coordonnées, spécifiquement le coin supérieur gauche $(x_{\text{haut}}, y_{\text{haut}})$ et le coin inférieur droit $(x_{\text{bas}}, y_{\text{bas}})$ de chaque segment pertinent. Cette recherche est effectuée dans la zone délimitée par les coordonnées précédemment acquises, garantissant ainsi que nous nous concentrons uniquement sur les portions d'intérêt de l'image.

Pour procéder à cette recherche, nous analysons chaque ligne de manière systématique. Cela implique de comparer les pixels ligne par ligne à la recherche de la présence de pixels noirs, qui indiquent la présence de lettres. Lorsque nous détectons un pixel noir $I(x, y) = 0$, nous marquons cela comme le début d'un bloc contenant des lettres. En revanche, si nous rencontrons une ligne entièrement blanche, c'est-à-dire $I(x, y) = 255$, cela signale la fin d'un bloc. Nous pouvons formaliser cette détection par l'expression suivante :

$$\text{Début du bloc} \Rightarrow I(x, y) = 0 \quad \text{et} \quad \text{Fin du bloc} \Rightarrow I(x, y) = 255$$

Cette méthode nous permet de renvoyer un tableau contenant les coordonnées $(x_{\text{haut}}, y_{\text{haut}})$ et $(x_{\text{bas}}, y_{\text{bas}})$ pour chaque bloc de lettres identifié.

Après avoir isolé les coordonnées des lignes à analyser, nous effectuons une recherche supplémentaire pour déterminer le nombre de lettres présentes dans chaque ligne. Cette étape est cruciale pour obtenir une compréhension précise de la distribution des lettres dans la grille. Pour cela, nous appliquons une méthode similaire à celle utilisée pour la détection des lignes, mais cette fois-ci en effectuant une recherche verticale afin de repérer les colonnes blanches.

Nous cherchons donc à déterminer les colonnes blanches à l'aide de la condition suivante :

$$\text{Colonne blanche} \Rightarrow \sum_{y=y_{\text{haut}}}^{y_{\text{bas}}} I(x, y) = 255 \times (y_{\text{bas}} - y_{\text{haut}} + 1)$$

Ainsi, chaque colonne est analysée de la même manière que les lignes, en scrutant les pixels pour identifier les segments de lettres. L'objectif est de recueillir les coordonnées des lettres en fonction de leur position dans la grille. En procédant ainsi, nous garantissons que toutes les lettres présentes dans la zone délimitée sont prises en compte, facilitant ainsi une compréhension globale de la disposition des lettres dans la grille.

6.0.5 Le découpage de l'image

Abordons maintenant notre fichier dédié à la découpe d'images. La structure de notre dossier est présenté en ici.

```
.
|--- Makefile
|--- crop_image.c
|--- crop_image.h
|--- main.c
```

Notre Makefile nous permet de compiler le fichier main.c, qui contient l'appel au fichier crop_image.c. Ce dernier ne possède pas de fonction principale, ce qui facilite son appel depuis d'autres fichiers, tout en évitant d'utiliser execvp, ce qui pourrait entraîner une consommation excessive de ressources. Le fichier d'en-tête .h contient les prototypes de fonctions, tandis que le fichier .c regroupe les définitions de ces fonctions.

Détaillons maintenant ces fonctions. Parmi elles, nous avons une fonction permettant de sauvegarder l'image à l'aide de la bibliothèque SDL.

La seconde fonction, CropImage, prend en paramètres l'image source, le chemin de destination, ainsi qu'une structure SDL appelée SDL_Rect, qui facilite la découpe. Nous vérifions d'abord que les coordonnées fournies dans cette structure sont valides, afin de nous assurer qu'elles se situent à l'intérieur des limites de l'image, évitant ainsi de découper en dehors de celle-ci.

Nous utilisons ensuite SDL_CreateRGBSurface pour allouer la mémoire nécessaire à la découpe et inscrire la taille requise. Par la suite, nous faisons appel à SDL_BlitSurface, une fonction SDL qui effectue la copie directe des pixels correspondants dans l'image de destination créée.

Enfin, la dernière fonction, exec_crop, est chargée de créer la structure et d'appeler la fonction de découpe en convertissant les coordonnées passées en paramètres (x1, y1, x2 et y2) en entiers. Cette fonction prend en paramètres le chemin de l'image source, celui de l'image de destination afin de la créer au bon emplacement, ainsi que les coordonnées du coin supérieur gauche et du coin inférieur droit pour effectuer la découpe.

6.0.6 L'implémentation de l'algorithme de résolution d'une grille de mots cachés

Abordons à présent le fonctionnement de notre solveur, un composant essentiel de notre projet. La structure de l'architecture du solveur est présenté en ici.

```
.
|--- Makefile
|--- main.c
|--- solver.c
|--- solver.h
```

Le fichier main.c sert d'interface principale pour le programme, et il appelle les fonctions définies dans solver.c, ce dernier ne contenant pas de fonction main. Cette approche a été choisie pour faciliter l'intégration du solveur dans d'autres parties de notre projet sans nécessiter des appels complexes ou des dépendances inutiles.

Une des principales fonctionnalités de notre solveur réside dans la capacité à charger une grille à partir d'un fichier texte. Pour ce faire, nous avons développé une fonction qui lit la grille fournie en tant que paramètre et la convertit en un tableau que notre programme peut manipuler. Ce tableau permet une gestion efficace de la grille, facilitant ainsi les opérations de recherche par la suite.

Après avoir chargé la grille, nous avons conçu une fonction de recherche de mots qui considère toutes les directions possibles. En effet, un mot dans une grille peut être orienté de huit manières différentes : horizontalement, verticalement, ainsi qu'en diagonale, dans les deux sens. Cette approche exhaustive nous permet de maximiser les chances de retrouver un mot dans n'importe quelle position au sein de la grille.

La fonction principale, que nous avons nommée exec_solver, joue un rôle central dans ce processus. Elle prend en entrée le chemin vers le fichier texte contenant la grille ainsi que le mot à rechercher. À partir de ces données, la fonction procède à la recherche du mot dans le tableau généré précédemment. Si le mot est trouvé, exec_solver renvoie les coordonnées de début et de fin du mot, correspondant respectivement à la position de la première lettre et de la dernière lettre dans le tableau. Cela permet non seulement d'identifier le mot recherché, mais aussi de fournir des informations précises quant à son emplacement dans la grille.

En revanche, si le mot n'est pas présent dans la grille, la fonction génère un message indiquant son absence. Cette fonctionnalité est cruciale, car elle offre une rétroaction immédiate à l'utilisateur, lui permettant de savoir si le mot recherché existe ou non dans la grille.

6.0.7 La preuve de concept d'un réseau de neurones

Nous allons maintenant nous pencher sur notre preuve de concept, dont l'architecture est présentée ci-dessous.

```
.
|--- ia_proof.c
|--- ia_proof.h
```

Cette preuve de concept est actuellement accessible uniquement via notre interface graphique, un aspect que nous prévoyons d'améliorer lors de notre prochaine présentation. L'objectif de notre preuve de concept est de répondre à l'expression booléenne suivante :

$$\overline{A}.\overline{B} + A.B$$

où A et B sont des variables booléennes pouvant prendre les valeurs 0 ou 1. Pour développer ce concept, nous avons consulté les ressources suivantes : TensorFlow Playground [3] et le livre Neural Networks and Deep Learning [4]. Ces ressources nous ont permis de mieux comprendre le fonctionnement des réseaux de neurones.

Pour illustrer ce concept, nous avons créé un réseau de neurones dont l'architecture est démontrée ici.

```
.
|--- neuron.c
|--- neuron.h
```

Nous avons conçu deux structures principales. La première structure, nommée neuron, est définie en ci-dessous.

```
// Structure of a neuron with its weights, bias, and output
typedef struct {
    double *weights;     // Pointer to an array of weights for the
       connections to this neuron
    int num_weights;     // Number of weights associated with this neuron (
       should equal the number of inputs)
    double bias;         // Bias term for the neuron, used to shift the
       activation function
    double output;       // Output value of the neuron after applying the
       activation function
    double delta;        // Error gradient for backpropagation, representing
        the sensitivity of the neuron's output to the error
} Neuron;
```

Code 1 – Définition de la structure d'un neurone

3. source : https ://playground.tensorflow.org
4. source : http ://neuralnetworksanddeeplearning.com/

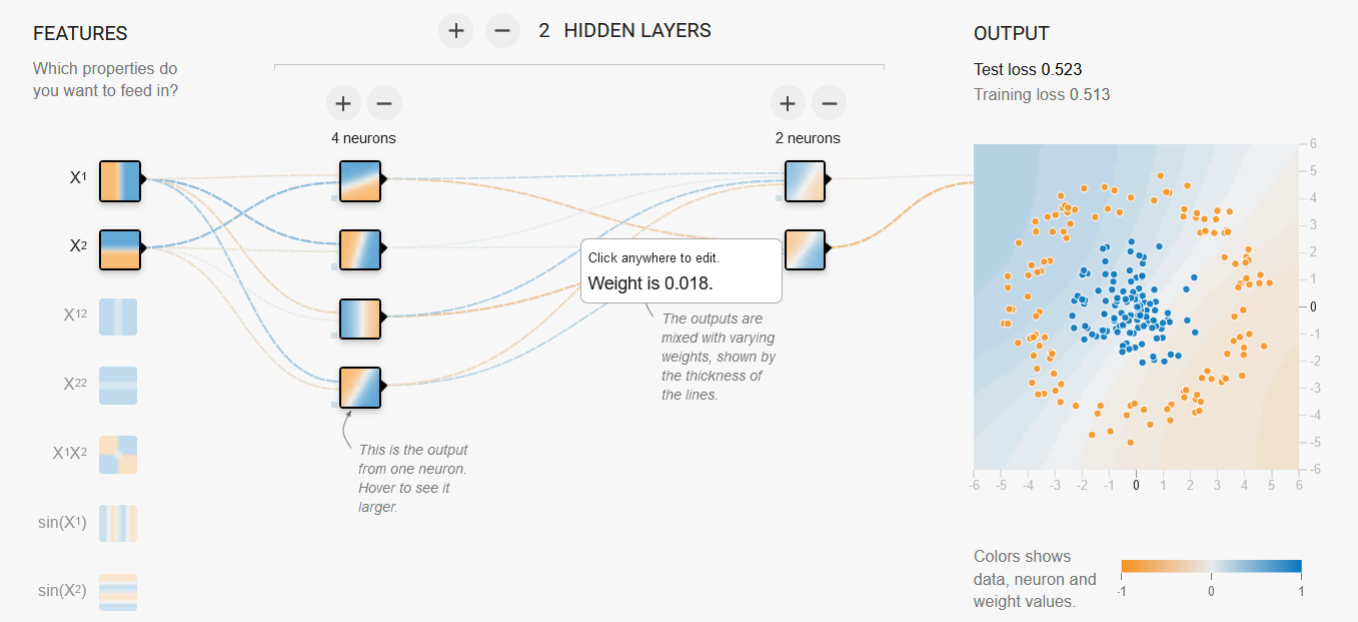
Cette structure inclut les poids de chaque neurone ainsi que le biais, l'erreur, et le delta, qui représente l'erreur de prédiction. La deuxième structure, layer, représente un réseau de neurones et contient plusieurs neurones, chacun étant composé de poids et d'informations. On peut l'envisager comme le "cerveau" d'une couche donnée, contenant les informations apprises par cette partie du réseau.

Nous avons également développé deux fonctions : l'une pour créer une couche (layer) et l'autre pour initialiser tous les paramètres de chaque neurone de manière aléatoire.

Revenons maintenant à notre preuve de concept concernant la fonction NXOR. Dans notre fonction principale, nous offrons deux options : apprendre ou prédire. Commençons par la prédiction. En fournissant deux valeurs pour A et B, le réseau prédit le résultat en fonction de ce qu'il a appris précédemment.

La partie la plus complexe concerne l'apprentissage. Pour commencer, nous avons mis en place une fonction pour charger le modèle si un réseau a déjà été créé pour résoudre ce problème. Dans le cas contraire, il est initialisé avec les fonctions définies dans neuron.c. De plus, une fonction permet de sauvegarder ces données dans un fichier texte. Nous avons également implémenté une fonction sigmoïde qui oriente le résultat du neurone vers 1 ou 0, sujet que nous aborderons plus en détail ultérieurement.

L'apprentissage se fait par un processus appelé "forward propagation". Comme illustré en ci-dessous provenant de TensorFlow, nous prenons en entrée deux paramètres : A et B, à chaque couche ou niveau de notre réseau.



Dans un réseau de neurones, nous prenons en entrée deux paramètres : A et B. À chaque couche ou niveau de notre réseau, chaque neurone effectue la somme pondérée de ses entrées, multipliées par leurs poids respectifs, puis ajoute un biais. Mathématiquement, cela peut être exprimé comme :

$$z = w_1 \cdot A + w_2 \cdot B + b$$

où :
— z est le résultat de la somme pondérée,
— w_1 et w_2 sont les poids associés aux entrées A et B,
— b est le biais.

L'objectif est de trouver une combinaison de poids telle que la sortie du neurone puisse correspondre à l'expression booléenne suivante :

$$f(A, B) = \overline{A} \cdot \overline{B} + A \cdot B$$

où $\overline{A}$ et $\overline{B}$ représentent les valeurs logiques NOT de A et B, respectivement. Cette fonction booléenne correspond à la fonction NXOR, qui renvoie 1 si les deux entrées sont identiques, et 0 sinon.

Chaque neurone calcule ainsi ce qu'il considère comme la meilleure combinaison pour produire le résultat attendu.

À ce stade, la fonction sigmoïde entre en jeu pour transformer la sortie en une valeur comprise entre 0 et 1. La fonction sigmoïde est définie par :

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

Cette transformation nous aide à déterminer si la combinaison de poids tend vers 1 (vrai) ou vers 0 (faux), influençant directement la décision prise par le neurone. Ainsi, nous obtenons une première sortie qui est généralement très aléatoire.

Ensuite, nous appliquons la rétropropagation, une méthode qui permet au réseau d'apprendre de ses erreurs. En fournissant au réseau le résultat attendu, nous calculons l'erreur E entre cette sortie et la sortie prédite :

$$E = \frac{1}{2}(y - \hat{y})^2$$

où :
— y est la sortie attendue (cible),
— $\hat{y}$ est la sortie prédite par le neurone.

Sur cette base, le réseau ajuste les poids de ses neurones selon la règle de mise à jour suivante :

$$w_i \leftarrow w_i + \eta \cdot \delta \cdot x_i$$

où :
— w_i est le poids à mettre à jour,
— η est le taux d'apprentissage,
— δ est l'erreur (gradient de l'erreur par rapport à la sortie),
— x_i est l'entrée correspondante.

Ce processus d'apprentissage est essentiel pour améliorer la performance du réseau sur des tâches complexes, comme la résolution de la fonction NXOR.

6.0.8 L'intelligence artificielle : reconnaissance des lettres

Dans cette section, nous abordons le développement d'un système d'intelligence artificielle dédié à la reconnaissance de lettres. Nous avons choisi de conserver une structure algorithmique similaire à celle utilisée pour la résolution du problème du `nxor`. Autrement dit, notre modèle repose sur deux composantes essentielles : une fonction d'apprentissage et une fonction de rétropropagation (*backpropagation*) pour l'optimisation des paramètres en fonction des erreurs.

Pour extraire les lettres à partir d'une image source, nous utilisons deux fonctions qui convertissent un rectangle défini par ses coordonnées $(x_{\text{haut-gauche}}, y_{\text{haut-gauche}})$ et $(x_{\text{bas-droit}}, y_{\text{bas-droit}})$ en une matrice binaire de taille fixe. Chaque élément de la matrice est une valeur 0 ou 1, correspondant respectivement à l'absence ou à la présence de pixels appartenant à la lettre.

La transformation peut être décrite comme suit : soit R le rectangle de coordonnées (x_1, y_1) et (x_2, y_2), et soit $f(x, y)$ la fonction binaire représentant l'intensité du pixel à la position (x, y). Nous générons une matrice normalisée $M_{i,j}$ telle que :

$$M_{i,j} = \begin{cases} 1, & \text{si } f(x_i, y_j) > T \\ 0, & \text{sinon} \end{cases}$$

où T est un seuil prédéfini.

Les données utilisées pour l'apprentissage sont stockées dans un fichier nommé `dataset_labels.txt`, qui contient les informations nécessaires à l'entraînement du modèle. Ce fichier est structuré de la manière suivante :

— La première ligne indique le nombre total d'images dans la base de données, noté N.
— Chaque ligne suivante décrit une image sous la forme :

$$\underbrace{00000011111\ldots1111000}_{\text{Représentation binaire}};\texttt{label};$$

où la représentation binaire correspond aux pixels de la lettre après transformation, et `label` est la classe associée à la lettre, par exemple 3 pour la lettre C.

Pour faciliter la génération du fichier `dataset_labels.txt`, nous avons développé un programme en langage C intitulé `creation_dataset.c`. Ce programme parcourt un dossier contenant des images, extrait pour chaque image sa représentation binaire et déduit son étiquette à partir du nom de fichier. Le fichier final contient ainsi l'ensemble des données nécessaires à l'entraînement du modèle.

L'algorithme peut être décrit comme suit :

1. Lire chaque image dans le dossier.
2. Convertir chaque image en une séquence binaire de $32 \times 32 = 1024$ bits.
3. Extraire le label de la lettre depuis le nom de fichier (par exemple, `lettre_1.png` correspond au label 1 pour la lettre A).
4. Ajouter la séquence binaire et le label dans le fichier `dataset_labels.txt`.

Dans le code de notre intelligence artificielle, une fonction dédiée lit le fichier `dataset_labels.txt` ligne par ligne. Elle transforme les données lues en deux tableaux :

— Un tableau de tableaux contenant les représentations binaires des images (X).
— Un tableau contenant les labels correspondants (Y).

Ces tableaux sont ensuite utilisés comme entrées pour le réseau de neurones lors de l'entraînement et de la prédiction.

Nous avons conservé la même structure neuronale que celle utilisée pour le problème du `nxor`. Cela inclut une fonction principale permettant à la fois l'entraînement et la prédiction.

L'apprentissage de notre réseau de neurones suit un processus rigoureux destiné à ajuster les poids de manière optimale pour prédire les lettres à partir de leurs représentations binaires.

Pour rendre l'apprentissage interactif et transparent, une barre de progression a été ajoutée afin de permettre à l'utilisateur de suivre l'état d'avancement du processus. Lorsque l'apprentissage est lancé, les étapes suivantes sont effectuées :

1. Chargement des poids :
   — Si un fichier de modèle nommé `modelletter.txt` existe, les poids y sont chargés.
   — Si ce fichier n'existe pas, les poids sont initialisés aléatoirement.
2. Chargement des données : les représentations binaires des images et leurs labels sont extraits à l'aide des fonctions mentionnées précédemment.

L'apprentissage s'effectue sur 100 époques, avec un taux d'apprentissage (ou *learning rate*) fixé à $\eta = 0.001$. Ce processus vise à minimiser la fonction de coût $J(\theta)$ en ajustant les poids θ pour réduire l'erreur entre les prédictions du modèle et les labels réels.

Lors de la phase de **propagation avant** (*forward propagation*), nous utilisons une architecture de réseau de neurones composée de plusieurs couches, où la fonction d'activation principale est la ReLU (*Rectified Linear Unit*). La fonction ReLU, définie par :

$$\text{ReLU}(x) = \max(0, x),$$

permet d'introduire de la non-linéarité dans le modèle tout en conservant une structure computationnelle simple et efficace. Cette fonction est particulièrement adaptée car elle renvoie des valeurs claires et distinctes, notamment 1 ou 0, pour représenter les biais au sein des neurones. Ces propriétés facilitent l'interprétation des activations intermédiaires et favorisent une convergence rapide lors de l'entraînement.

Une fois les activations propagées à travers toutes les couches intermédiaires, nous appliquons une fonction **softmax** sur la dernière couche. La fonction softmax est définie comme suit pour une sortie $\mathbf{z} = [z_1, z_2, \ldots, z_k]$ associée à k classes :

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}.$$

Elle transforme les sorties en une distribution de probabilité, où chaque valeur représente la probabilité que l'entrée appartienne à une classe donnée. Cette étape est cruciale pour les tâches de classification, car elle permet d'obtenir des prédictions probabilistes exploitables.

Pendant la phase de **rétropropagation** (*backpropagation*), nous utilisons la dérivée de la fonction ReLU pour calculer le gradient de la fonction de perte par rapport aux paramètres du modèle. La dérivée de ReLU est donnée par :

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{si } x > 0, \\ 0 & \text{sinon.} \end{cases}$$

Cette dérivée simple mais efficace permet d'identifier les poids à ajuster pour minimiser l'erreur sur l'ensemble des prédictions. Plus précisément, nous calculons les gradients de manière itérative à travers les couches, en utilisant la règle de la chaîne pour propager l'erreur de sortie jusqu'aux couches d'entrée. Les poids sont alors mis à jour à l'aide d'un algorithme de descente de gradient, selon la formule suivante :

$$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla_w \mathcal{L},$$

où $w^{(t)}$ représente le poids à l'itération t, η est le taux d'apprentissage, et $\nabla_w \mathcal{L}$ est le gradient de la fonction de perte $\mathcal{L}$ par rapport au poids w.

Ce processus, qui combine la propagation avant et la rétropropagation, est répété sur l'ensemble du jeu de données pour chaque époque. Une époque correspond à un passage complet sur toutes les images du jeu de données. Dans notre cas, ce jeu de données contient plus de 4000 images. Pour chaque image, le modèle effectue un cycle complet de propagation avant et de rétropropagation, ce qui lui permet d'apprendre de manière progressive en ajustant les poids pour minimiser l'erreur sur les prédictions.

En itérant sur plusieurs époques, le modèle converge vers une solution optimisée, où les poids du réseau sont ajustés pour produire des prédictions précises. Ce processus d'entraînement est essentiel pour garantir que le réseau est capable de généraliser à des données inconnues, ce qui constitue l'objectif final de tout système d'apprentissage supervisé.

Ainsi, grâce à cette combinaison de fonctions d'activation (ReLU et softmax), à l'optimisation des gradients par rétropropagation, et à l'entraînement sur un large jeu de données, notre modèle est en mesure d'apprendre efficacement à partir des exemples fournis et de réduire progressivement l'erreur au fil des itérations.

Lors de l'apprentissage, nous affichons dans le terminal une représentation visuelle de la lettre en pixels noirs et blancs, suivie de la prédiction effectuée par le modèle. Cet affichage permet de suivre en temps réel les performances du réseau de neurones et d'identifier visuellement les éventuelles erreurs. Voici un extrait typique de l'affichage dans le terminal :

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000111111111000000000000000
00000001111111111100000000000000
00000011111111111110000000000000
00000111111111111111000000000000
00000111110000011111000000000000
00001111100000001111000000000000
00001111100000001111000000000000
00000111110000000000000000000000
00000111111100000000000000000000
00000011111111100000000000000000
00000011111111111000000000000000
00000000111111111110000000000000
00000000000011111111000000000000
00000000000000001111000000000000
00001110000000001111000000000000
00001111000000001111000000000000
00001111100000011111000000000000
00001111111001111111000000000000
00000111111111111111000000000000
00000011111111111110000000000000
00000011111111111000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000

The predicted letter is S
```

Dans cet exemple, l'image en entrée correspond à une lettre « S » représentée par une matrice de caractères « 1 » (pixels noirs) et de caractères « 0 » (pixels blancs). Sous cette représentation, le modèle affiche sa prédiction, ici « S ».

Ce type d'affichage est particulièrement utile pour :
— Vérifier la cohérence entre l'image entrée et la prédiction du modèle.
— Identifier les erreurs de classification, par exemple si le modèle prédit une lettre différente de celle en entrée.
— Suivre l'évolution des performances à travers les différentes époques.

Un tel retour visuel offre une manière intuitive de valider que le modèle comprend correctement les données en entrée, même pour des utilisateurs sans connaissance approfondie des matrices numériques sous-jacentes.

Afin d'élargir les fonctionnalités du modèle, nous avons ajouté la possibilité de prédire des images de dimensions inférieures ou égales à 32×32 pixels. Cette nouvelle fonctionnalité permet d'introduire davantage de flexibilité, notamment pour des applications avec des données à faible résolution. L'image est affichée directement dans le terminal sous une représentation en noir et blanc, suivie du résultat de la prédiction produit par l'intelligence artificielle.

Voici un exemple typique de l'affichage dans le terminal pour une image :

```
00000111111000000000000000000000
00011111111110000000000000000000
00111111111111000000000000000000
00111111111111000000000000000000
01111000000111100000000000000000
11110000000011110000000000000000
11110000000011110000000000000000
11110000000011110000000000000000
11110000000011110000000000000000
11110000000011110000000000000000
11110000000011110000000000000000
01111000000111100000000000000000
00111111111111100000000000000000
00011111111111000000000000000000
00001111111110000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000

The predicted letter is O
```

Dans cet extrait, l'image correspond à la lettre « O », représentée sous forme matricielle avec les caractères « 1 » pour les pixels noirs et les caractères « 0 » pour les pixels blancs. Le modèle, après avoir traité l'entrée, affiche la prédiction « O » en dessous de l'image.

Lorsqu'une image est fournie : 1. Le modèle vérifie que les dimensions sont conformes ($h \times w \leq 32 \times 32$). 2. L'image est convertie en une représentation binaire (noir et blanc). 3. Une propagation avant (*forward propagation*) est effectuée pour générer une prédiction. 4. Le terminal affiche l'image sous sa forme binarisée, suivie de la lettre prédite.

Cette extension apporte plusieurs avantages :
— **Compatibilité avec des images de petite taille :** Le modèle est désormais adapté à des applications nécessitant des entrées à faible résolution.
— **Facilité de validation visuelle :** L'affichage dans le terminal permet aux utilisateurs de vérifier rapidement l'exactitude des prédictions.
— **Applications pratiques :** Cette capacité élargie ouvre la voie à des utilisations variées, comme la reconnaissance de caractères manuscrits ou de symboles à faible résolution.

Cette fonctionnalité démontre l'adaptabilité du modèle et sa capacité à gérer des cas d'usage variés, tout en conservant une interface simple et intuitive.

6.0.9 La résolution des images

Dans le cadre de notre projet, la résolution des images suit une série d'étapes bien définies, allant du prétraitement des données (*preprocessing*) jusqu'à l'identification des lettres et la résolution finale des grilles. Voici une description détaillée des différentes phases impliquées :

La première étape consiste à transformer l'image d'entrée dans un format compatible avec notre pipeline d'analyse. Cette étape est réalisée grâce à une série de fonctions que nous avons développées, permettant d'effectuer les opérations suivantes :

1. **Conversion en niveaux de gris :** L'image en couleurs est d'abord convertie en nuances de gris.
2. **Binarisation :** Une fois en niveaux de gris, l'image est convertie en une version binaire (noir et blanc) en appliquant un seuil T. La valeur de chaque pixel est alors définie comme :

$$I_{\text{bin}}(x, y) = \begin{cases} 1 & \text{si } I_{\text{gris}}(x, y) > T, \\ 0 & \text{sinon.} \end{cases}$$

3. **Correction de l'orientation :** Une rotation automatique est réalisée pour aligner l'image selon les axes principaux, en détectant les bords dominants et en appliquant une transformation affine pour redresser l'image.

Ces étapes garantissent que l'image est normalisée et prête pour les phases suivantes.

Une fois le prétraitement terminé, nous faisons appel à une fonction d'analyse spécifique qui extrait les informations structurales de l'image. Cette fonction identifie trois éléments principaux :

1. **La grille de lettres :** Les coordonnées de chaque cellule de la grille sont extraites et stockées pour un traitement ultérieur.
2. **La liste de mots :** Les régions correspondant aux mots à rechercher sont localisées, en identifiant leurs positions dans l'image.
3. **Chaque lettre individuellement :** Chaque caractère contenu dans la grille est segmenté et isolé pour être reconnu ultérieurement.

Ces informations sont cruciales pour les étapes suivantes et permettent de structurer l'analyse de manière hiérarchique.

Pour identifier chaque lettre contenue dans la grille, nous utilisons un modèle d'intelligence artificielle préalablement entraîné. Ce modèle prend en entrée une image binarisée d'une lettre et retourne la lettre correspondante.

Le processus de reconnaissance peut être formalisé comme suit : soit $I_{\text{lettre}}(x, y)$ l'image binaire d'une lettre, le modèle IA applique une fonction de classification f_{IA} telle que :

$$\text{Lettre prédite} = f_{\text{IA}}(I_{\text{lettre}}).$$

La précision de cette étape est directement liée à la qualité du prétraitement et à la performance du modèle d'apprentissage automatique.

Le traitement de la grille de lettres et de la liste de mots est effectué séparément afin de remplir deux fichiers texte distincts :

1. **Fichier 1 :** Contient les lettres reconnues dans la grille, organisées selon leurs coordonnées respectives.
2. **Fichier 2 :** Contient la liste des mots à rechercher avec leurs coordonnées dans l'image.

Ces fichiers servent ensuite d'entrée à notre solveur, implémenté lors de la dernière soutenance. Le solveur utilise les données extraites pour localiser les mots dans la grille, en calculant leurs positions et en traçant les résultats directement sur l'image binarisée.

Une fois les mots trouvés, les coordonnées retournées par le solveur sont utilisées pour dessiner les tracés sur l'image. Cette étape finale consiste à superposer les mots trouvés sur la grille initiale.

La visualisation finale est sauvegardée, montrant clairement les mots identifiés, comme illustré ci-dessous :

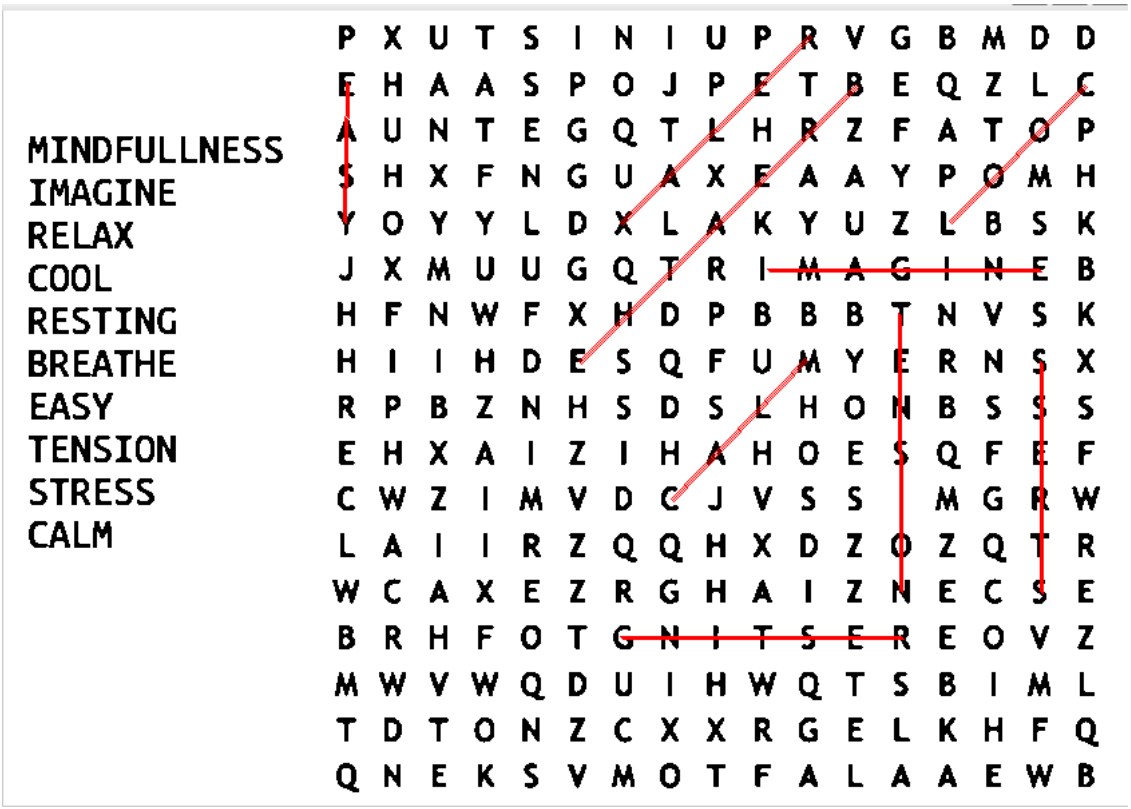


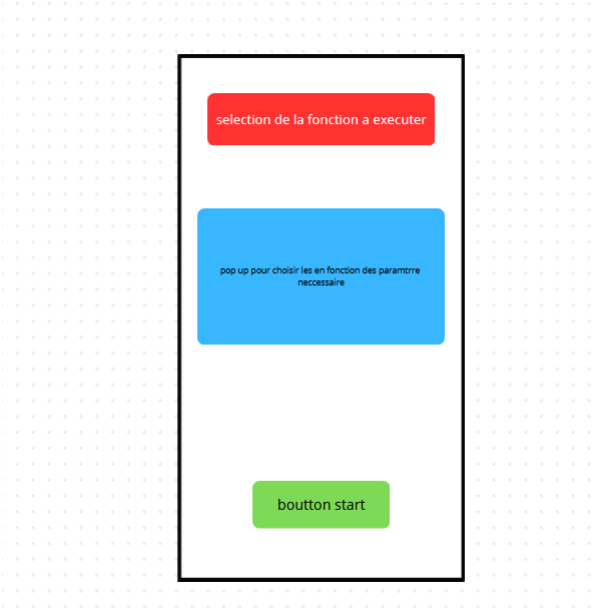
FIGURE 1 – Exemple de grille annotée après résolution.

Ce processus structuré, allant du prétraitement des images jusqu'à la résolution finale des grilles, garantit une analyse efficace et précise. Grâce à l'intégration du solveur et du modèle d'intelligence artificielle, nous sommes capables de résoudre des grilles complexes tout en produisant une visualisation intuitive des résultats.

6.0.10 L'interface graphique

Abordons à présent le développement de l'interface graphique de notre projet. Bien qu'il n'ait pas été nécessaire de créer une interface graphique pour la première soutenance, nous avons pris la décision d'en concevoir une. Cette initiative visait à présenter notre projet de manière plus claire et structurée, tout en regroupant tous nos fichiers et fonctionnalités au sein d'un même environnement. Cela permet de simplifier l'utilisation et d'éviter de naviguer entre divers fichiers dispersés dans le système durant la présentation.

Pour concevoir notre interface graphique, nous avons élaboré un schéma relativement simple, qui illustre la structure et l'organisation des éléments que nous souhaitions intégrer. Vous pouvez voir le schéma prototype de notre interface en ici.



Nous avons opté pour une interface minimaliste, afin d'éviter les complications liées à une implémentation trop complexe, qui aurait pu compromettre la fonctionnalité de l'application.

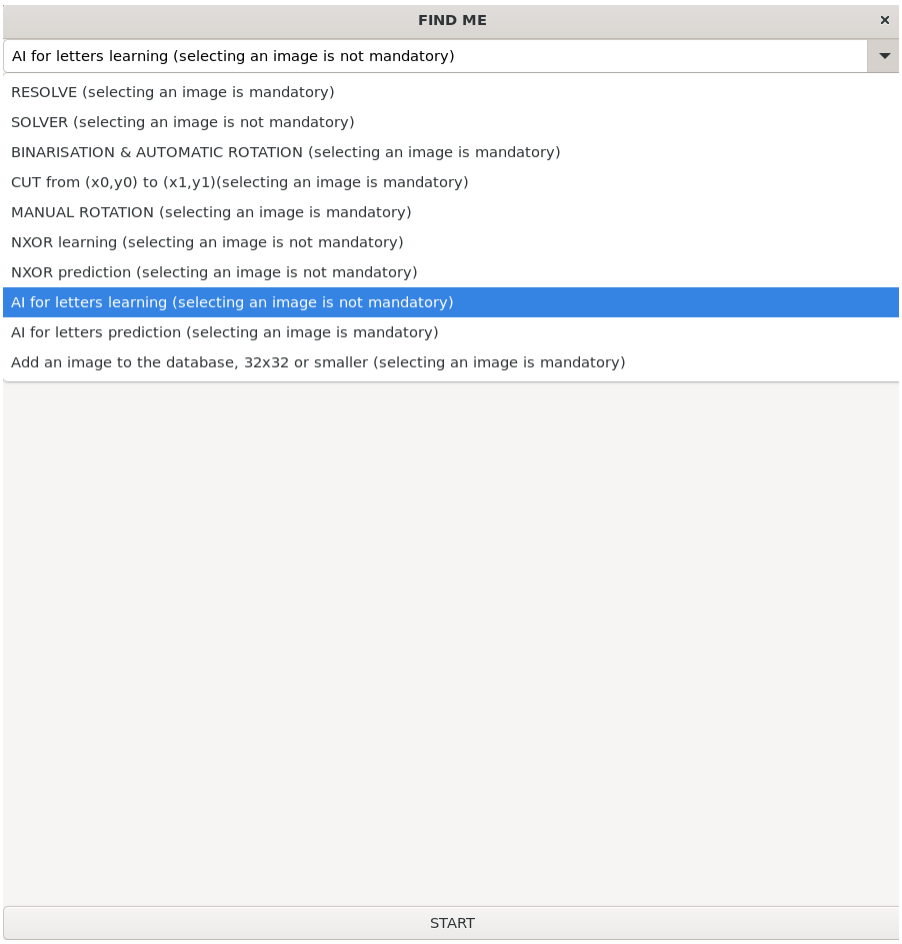
Ce choix s'est avéré judicieux, car nous avons rencontré plusieurs défis lors du développement de l'interface, en particulier en ce qui concerne l'affichage dynamique des boutons. Ces boutons apparaissent et disparaissent en fonction des options sélectionnées dans l'interface, ce qui a nécessité une gestion soignée des événements et des interactions utilisateur.

Pour réaliser notre interface graphique, nous avons utilisé la bibliothèque GTK3, qui est bien adaptée pour le développement d'applications en C. Le processus a débuté par la création de notre fenêtre principale, à laquelle nous avons ajouté tous les boutons nécessaires. Toutefois, nous avons choisi de masquer ces boutons initialement afin de ne les afficher que lorsque les conditions requises étaient remplies.

L'un des éléments clés de notre interface est la Combobox, un élément interactif qui permet à l'utilisateur de faire des sélections parmi plusieurs options. Lorsqu'un utilisateur effectue une sélection dans cette liste déroulante, les boutons pertinents s'affichent dynamiquement en fonction des choix effectués. Cela permet de ne montrer que les options nécessaires à chaque étape, rendant ainsi l'interface plus intuitive et évitant la surcharge d'informations.

Une fois que tous les paramètres requis pour une fonction sont renseignés, le bouton Start devient visible. Ce bouton, lorsqu'il est activé, exécute la fonctionnalité demandée et affiche les résultats à l'utilisateur. Cette approche facilite l'interaction avec le programme et permet d'optimiser l'expérience utilisateur en rendant les processus plus fluides et compréhensibles.

En somme, bien que le développement de l'interface graphique ait présenté son lot de défis, il a également enrichi notre projet en offrant une présentation cohérente et accessible de nos travaux. Cette interface représente une étape importante dans notre projet, car elle permet à tous les utilisateurs, qu'ils soient novices ou expérimentés, de comprendre et d'interagir facilement avec les fonctionnalités que nous avons développées.

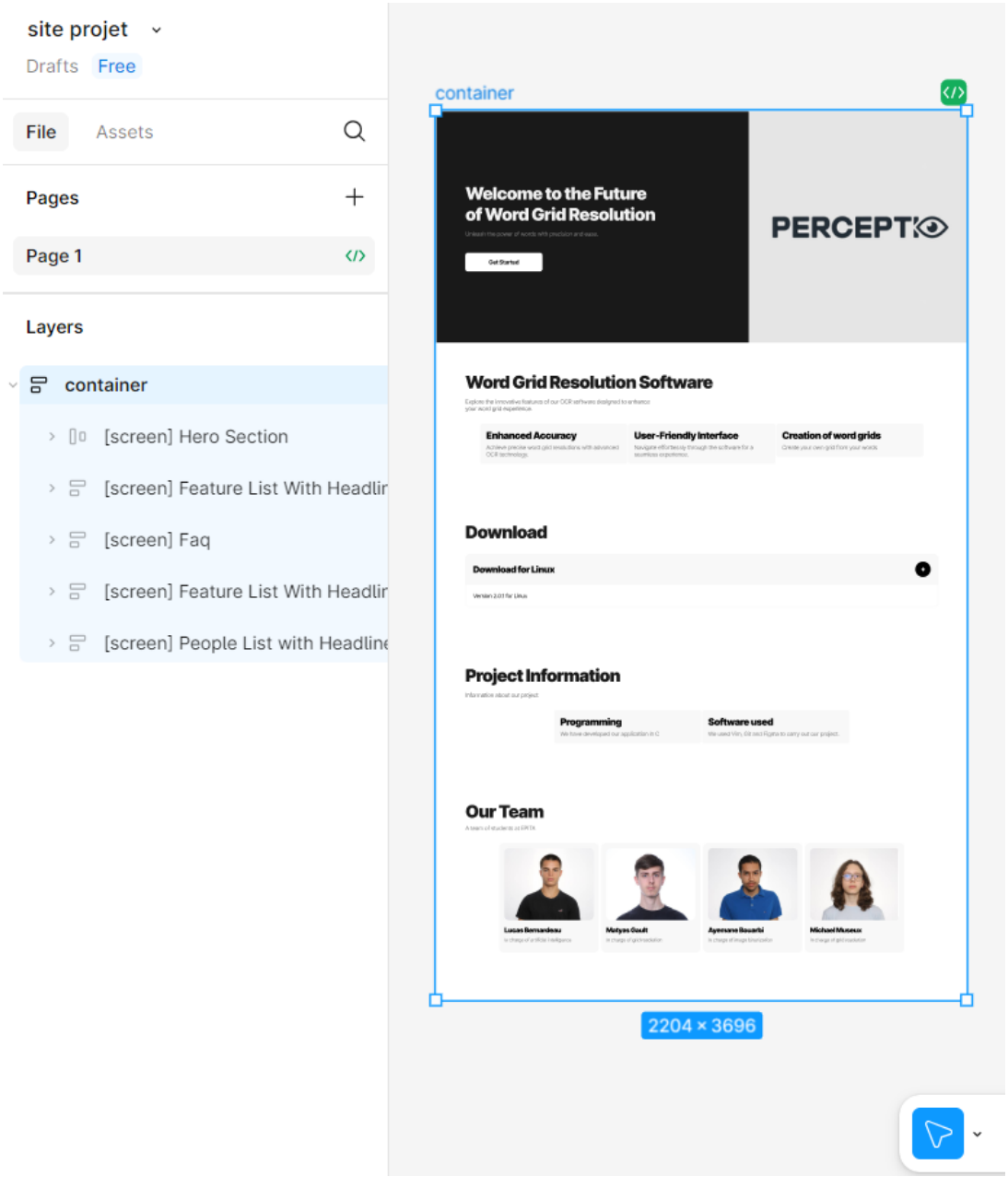


6.0.11 Le site internet

Dans le cadre de notre projet, la création d'un site internet était une étape essentielle pour présenter notre travail et faciliter l'accès à ses fonctionnalités. Ce site a été conçu avec l'objectif d'être à la fois esthétique, fonctionnel et facile à développer dans les délais impartis. Voici une description détaillée de notre démarche, depuis l'imagination jusqu'à la conception.

Conception du design

La première étape de notre processus a consisté à imaginer et élaborer un design répondant à nos besoins. Nous avons utilisé Figma, un outil de conception collaboratif, pour créer une maquette de notre site. Cette étape a impliqué plusieurs discussions approfondies au sein de l'équipe afin de définir une esthétique qui reflète notre identité tout en restant pratique à coder. Nous avons pris soin de trouver un équilibre entre un design attrayant et une structure technique simple, permettant une implémentation efficace.



Réalisation technique et choix des outils

Compte tenu du temps limité dont nous disposions, nous avons choisi de développer notre site en utilisant Bootstrap, un framework CSS reconnu pour sa rapidité de mise en œuvre. Bootstrap offre une collection de composants préconçus et un système de mise en page basé sur une grille, ce qui nous a permis de structurer notre site rapidement et efficacement. Grâce à cet outil, nous avons pu concentrer nos efforts sur le contenu et les fonctionnalités sans sacrifier la qualité visuelle et technique de notre site.

Structure et contenu du site

Le site internet se présente sous la forme d'une landing page organisée en cinq sections distinctes, conçues pour guider l'utilisateur à travers les informations essentielles de manière claire et fluide :

1. Section d'accueil : La première section vise à capter l'attention des visiteurs dès leur arrivée. Elle contient notre logo et un bouton bien visible qui redirige vers l'espace de téléchargement. L'objectif est de maximiser l'impact visuel tout en offrant un point d'accès immédiat aux fonctionnalités principales.
2. Présentation générale du projet : Cette deuxième section fournit une vue d'ensemble de notre projet. Elle décrit notre produit comme un OCR (Optical Character Recognition) accessible via une interface graphique conviviale. Cette partie est essentielle pour aider les visiteurs à comprendre rapidement la nature et l'utilité de notre solution.
3. Espace de téléchargement : La troisième section est dédiée au téléchargement de notre application. Actuellement, elle propose un bouton permettant d'accéder à un fichier exécutable. À l'avenir, nous prévoyons de développer un installateur (.exe) pour simplifier encore davantage l'installation et l'utilisation par les utilisateurs finaux.
4. Informations techniques : La quatrième section s'adresse aux utilisateurs intéressés par les détails techniques du projet. Nous y présentons le langage de programmation principal utilisé pour le développement de notre OCR, qui est le C. Cette partie met en avant la solidité et la performance de notre solution sur le plan technique.
5. Présentation de l'équipe : Enfin, la dernière section met en avant les membres de l'équipe de développement. Elle détaille les rôles et responsabilités de chacun, illustrant la diversité des compétences mobilisées pour mener à bien ce projet.

Ce site internet constitue une vitrine essentielle pour notre projet. Sa conception soignée, combinée à l'utilisation d'un framework performant, nous a permis de créer un outil fonctionnel et adapté aux besoins de nos utilisateurs. Cependant, nous voyons ce site comme une base évolutive. À l'avenir, nous envisageons d'ajouter des fonctionnalités supplémentaires, notamment des mises à jour régulières et un espace utilisateur personnalisé pour répondre aux attentes d'un public plus large.

6.0.12 Les options supplémentaires

Dans le cadre de l'amélioration de notre logiciel, nous avons ajouté une fonctionnalité essentielle : la possibilité d'ajouter de nouvelles images directement dans la base de données via l'interface utilisateur. Cette option vise à rendre l'utilisation de notre logiciel plus flexible et à permettre une personnalisation accrue des données.

Depuis l'interface graphique de notre application, un bouton a été ajouté pour permettre à l'utilisateur de sélectionner une image. Une fois l'image choisie, elle est automatiquement copiée dans un dossier dédié, conçu pour contenir toutes les images à traiter. Cette étape initiale est rapide et intuitive, garantissant une expérience utilisateur fluide.

Une fois les images placées dans le dossier, une fonction spécifique est activée pour effectuer les vérifications et transformations nécessaires. Voici les étapes principales de ce processus :

1. La fonction parcourt l'ensemble des fichiers présents dans le dossier afin d'identifier les images à traiter.
2. Chaque image est vérifiée pour s'assurer qu'elle respecte les dimensions requises, soit une taille inférieure ou égale à `32x32` pixels. Si l'image ne respecte pas cette contrainte, elle est ignorée.
3. Les images conformes passent ensuite par une seconde fonction qui les convertit en une matrice de `0` et `1`, correspondant à leur structure binaire. Cette conversion est essentielle pour l'intégration dans le modèle de reconnaissance optique.
4. Simultanément, un label est généré pour chaque image. Ce label correspond à la position de la lettre représentée dans l'alphabet, ce qui permet une association directe entre l'image et sa classification.
5. Une fois les données binaires et les labels générés, ils sont ajoutés ligne par ligne dans le fichier `dataset_labels.txt`, qui sert de base de données principale pour notre logiciel.

Cette fonctionnalité offre plusieurs avantages majeurs :

— Les utilisateurs peuvent enrichir la base de données sans avoir besoin de connaissances techniques approfondies.
— Les nouvelles données permettent d'adapter le logiciel à des besoins spécifiques ou à des alphabets personnalisés.
— Le processus automatisé réduit les risques d'erreur humaine et accélère la mise à jour de la base de données.

À l'avenir, nous envisageons d'améliorer cette fonctionnalité en ajoutant la possibilité de redimensionner automatiquement les images non conformes et de gérer des formats supplémentaires. Cette évolution renforcerait encore l'efficacité et la convivialité de notre logiciel.

7 Conclusion

En conclusion, nous avons réalisé des progrès significatifs dans notre projet, et nous sommes très satisfaits de l'avancement global. À ce stade, nous sommes bien dans les délais prévus, ce qui nous permet d'aborder les étapes restantes avec confiance.

Il ne nous reste plus qu'à finaliser quelques éléments clés avant notre prochaine soutenance. Tout d'abord, il est essentiel d'achever le prétraitement complet des images, incluant l'implémentation de la rotation automatique. Cette étape est cruciale pour garantir que les images soient correctement orientées avant d'être traitées par notre réseau de neurones. En intégrant cette fonctionnalité, nous nous assurerons que les données d'entrée soient optimales pour la phase suivante.

Ensuite, le développement d'un réseau de neurones complet et fonctionnel est primordial. Nous devons finaliser la phase d'apprentissage du réseau, ce qui implique de former le modèle sur un ensemble de données représentatif afin qu'il puisse généraliser ses prédictions sur de nouvelles images. Parallèlement, il est impératif que notre réseau soit capable de reconnaître les lettres présentes dans la grille ainsi que dans la liste de mots. Cette fonctionnalité est au cœur de notre projet, car elle déterminera l'efficacité de notre solution dans des scénarios réels.

Enfin, il est nécessaire de travailler sur la reconstruction de l'image résolue. Cette étape finale est essentielle pour présenter les résultats de notre traitement et démontrer l'efficacité de notre approche lors de la soutenance. Une image bien reconstruite non seulement met en valeur notre travail, mais permet également de visualiser clairement les résultats obtenus par notre réseau de neurones.

En somme, les étapes restantes à accomplir sont à la fois techniques et stratégiques. En nous concentrant sur ces points, nous sommes convaincus que nous serons en mesure de présenter un projet robuste et fonctionnel. Nous continuerons à travailler avec diligence pour respecter les délais et garantir un résultat de haute qualité lors de notre soutenance finale.

8 Annexes

8.1 Annexe 1 : Fonctions de chargement et de sauvegarde d'une image

```
    // Function to load an image
    SDL_Surface* loadImageRotation(const char* filename)
    {
        SDL_Surface* image = IMG_Load(filename);
        if (!image)
        {
            fprintf(stderr, "Error loading image %s: %s\n",
            filename, IMG_GetError());
            exit(1);
        }
        return image;
    }

    // Function to save an image as PNG
    void saveImageRotation(SDL_Surface* surface, const char* filename)
    {
        if (IMG_SavePNG(surface, filename) != 0)
        {
            fprintf(stderr, "Error saving image: %s\n", IMG_GetError());
            exit(1);
        }
    }
```

Code 2 – Fonctions de chargement et de sauvegarde d'une image

8.2 Annexe 2 : Fonction exec_rotation pour effectuer la rotation d'une image

```
int exec_rotation(int argc, char* argv[]) {
    if (argc != 4) {
        fprintf(stderr,
        "Usage: %s <input.jpg/png> <output.png> <angle_degrees>\n", argv
           [0]);
        return 1;
    }

    if (SDL_Init(SDL_INIT_VIDEO) != 0)
    {
        fprintf(stderr, "Error initializing SDL: %s\n", SDL_GetError());
        return 1;
    }
    if (!IMG_Init(IMG_INIT_JPG | IMG_INIT_PNG))
    {
        fprintf(stderr, "Error initializing SDL_image: %s\n",
           IMG_GetError());
        return 1;
    }

    SDL_Surface* img = loadImageRotation(argv[1]);

    // Rotate image angle
    double angle = atof(argv[3]);
    SDL_Surface* rotated = rotateImage(img, angle);

    // Save
    saveImageRotation(rotated, argv[2]);

    // Free
    SDL_FreeSurface(img);
    SDL_FreeSurface(rotated);

    IMG_Quit();
    SDL_Quit();

    return 0;
}
```

Code 3 – Fonction exec_rotation pour effectuer la rotation d'une image

8.3 Annexe 3 : Fonction rotateImage pour effectuer la rotation d'une image

```c
// Function to rotate the image
SDL_Surface* rotateImage(SDL_Surface* src, double angle_degrees) {
    double angle = angle_degrees * PI / 180.0;  // angle to radians
    double cos_angle = cos(angle);  // cosine of the angle
    double sin_angle = sin(angle);  // sine of the angle
    // Calculate dimensions -> image after rotation
    int new_width = (int)(fabs(src->w * cos_angle) + fabs(src->h * sin_angle
       ));
    int new_height = (int)(fabs(src->w * sin_angle) + fabs(src->h *
       cos_angle));
    // Allocate a new surface -> rotated image
    SDL_Surface* rotated = SDL_CreateRGBSurface(0, new_width, new_height,
       32, 0x00FF0000, 0x0000FF00,0x000000FF, 0xFF000000);
    if (!rotated) {
        fprintf(stderr, "Error creating surface: %s\n", SDL_GetError());
        exit(1);
    }
    // Coordinates of the center of the original and resulting image
    int cx = src->w / 2;
    int cy = src->h / 2;
    int new_cx = new_width / 2;
    int new_cy = new_height / 2;
    // Fill the rotated image with pixels from the original image
    Uint32* src_pixels = (Uint32*)src->pixels;
    Uint32* rotated_pixels = (Uint32*)rotated->pixels;
    for (int y = 0; y < new_height; y++) {
        for (int x = 0; x < new_width; x++) {
            int xt = x - new_cx;  // coordinates in the rotated image
            int yt = y - new_cy;
            // corresponding coordinates -> original image
            int src_x = (int)(cos_angle * xt - sin_angle * yt) + cx;
            int src_y = (int)(sin_angle * xt + cos_angle * yt) + cy;
            // Copy pixel if coordinates valid
            if (src_x >= 0 && src_x < src->w && src_y >= 0 && src_y < src->h
               ) {
                rotated_pixels[y * new_width + x] = src_pixels[src_y * src->
                   w + src_x];
            }
            else {
                rotated_pixels[y * new_width + x] = SDL_MapRGB(rotated->
                   format, 255, 255, 255);  // Fill with white
            }
        }
    }
    return rotated;
}
```

Code 4 – Fonction rotateImage pour effectuer la rotation d'une image

8.4 Annexe 4 : Fonction SaveImage pour sauvegarder une surface en PNG

```c
// Function to save the surface as a PNG file
int SaveImage(SDL_Surface *surface, const char *filename) {
    if (IMG_SavePNG(surface, filename) != 0)
    {
        printf("Error saving image: %s\n", SDL_GetError());
        return -1;
    }
    printf("Image saved as %s\n", filename);
    return 0;
}
```

Code 5 – Fonction SaveImage pour sauvegarder une surface en PNG

8.5 Annexe 5 : Fonction CropImage pour couper une image

```c
void CropImage(const char *input_image, const char *output_image, SDL_Rect crop_rect) {
    if (SDL_Init(SDL_INIT_VIDEO) != 0) {
        printf("SDL initialization error: %s\n", SDL_GetError());
        return;
    }
    if (!(IMG_Init(IMG_INIT_PNG) & IMG_INIT_PNG)) {
        printf("Error initializing SDL_image: %s\n", SDL_GetError());
        SDL_Quit();
        return;
    }
    SDL_Surface *image = IMG_Load(input_image);
    if (!image) {
        printf("Error loading image: %s\n", IMG_GetError());
        IMG_Quit();
        SDL_Quit();
        return;
    }
    // Check crop dimensions
    if (crop_rect.x < 0 || crop_rect.y < 0 || crop_rect.x + crop_rect.w > image->w || crop_rect.y + crop_rect.h > image->h) {
        printf("Invalid cropping dimensions.\n");
        SDL_FreeSurface(image);
        IMG_Quit();
        SDL_Quit();
        return;
    }
    // Create a new surface for the cropped image
    SDL_Surface *cropped = SDL_CreateRGBSurface(0, crop_rect.w, crop_rect.h,
        image->format->BitsPerPixel, image->format->Rmask, image->format->Gmask, image->format->Bmask, image->format->Amask);
    if (!cropped) {
        printf("Error creating cropped surface: %s\n", SDL_GetError());
        SDL_FreeSurface(image);
        IMG_Quit();
        SDL_Quit();
        return;
    }
    // Copy the cropped portion from the original image
    SDL_BlitSurface(image, &crop_rect, cropped, NULL);
    SaveImage(cropped, output_image);
    SDL_FreeSurface(cropped);
    SDL_FreeSurface(image);
    IMG_Quit();
    SDL_Quit();
}
```

Code 6 – Fonction CropImage pour rogner une partie d'une image et la sauvegarder

8.6 Annexe 6 : Fonction exec_crop pour appeler CropImage

```c
// Function to execute the cropping operation
int exec_crop(int argc, char *argv[]) {
    if (argc != 7)
    {
        printf("Usage: %s <input_image> <output_image> <x1> <y1> <x2> <y2>\n
           ",\
        argv[0]);
        return 1;
    }

    const char *input_image = argv[1];
    const char *output_image = argv[2];

    // coordinates
    int x1 = atoi(argv[3]);
    int y1 = atoi(argv[4]);
    int x2 = atoi(argv[5]);
    int y2 = atoi(argv[6]);

    if (x1 >= x2 || y1 >= y2)
    {
        printf("Invalid crop coordinates. \
        (x1, y1) must be the top-left corner of (x2, y2).\n");
        return 1;
    }

    // Define crop rectangle
    SDL_Rect crop_rect;
    crop_rect.x = x1;
    crop_rect.y = y1;
    crop_rect.w = x2 - x1;
    crop_rect.h = y2 - y1;

    // Crop, save image
    CropImage(input_image, output_image, crop_rect);

    return 0;
}
```

Code 7 – Fonction exec_crop pour rogner une image

8.7 Annexe 7 : Contenu de solver.h

```c
#ifndef SOLVER_H
#define SOLVER_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_ROWS 100
#define MAX_COLUMNS 100

// Function to read the grid from a text file
int read_grid(char grid[MAX_ROWS][MAX_COLUMNS], const char *filename);
// Function to convert a word to uppercase
void convert_to_uppercase(char *word);
// Function to search for the word horizontally (left to right)
int find_horizontal(char grid[MAX_ROWS][MAX_COLUMNS], int rows, const char *
    word, int *x0, int *y0, int *x1, int *y1, int word_len);
// Function to search for the word horizontally from right to left
int find_horizontal_right_to_left(char grid[MAX_ROWS][MAX_COLUMNS], int rows
    , const char *word, int *x0, int *y0, int *x1, int *y1, int word_len);
// Function to search for the word vertically (top to bottom)
int find_vertical_top_to_bottom(char grid[MAX_ROWS][MAX_COLUMNS], int rows,
    const char *word, int *x0, int *y0, int *x1, int *y1, int word_len);
// Function to search for the word vertically (bottom to top)
int find_vertical_bottom_to_top(char grid[MAX_ROWS][MAX_COLUMNS], int rows,
    const char *word, int *x0, int *y0, int *x1, int *y1, int word_len);
// Function to search for the word diagonally (top-left to bottom-right)
int find_diagonal_top_left_to_bottom_right(char grid[MAX_ROWS][MAX_COLUMNS],
     int rows, const char *word, int *x0, int *y0, int *x1, int *y1, int
    word_len);
// Function to search for the word diagonally (bottom-left to top-right)
int find_diagonal_bottom_left_to_top_right(char grid[MAX_ROWS][MAX_COLUMNS],
     int rows, const char *word, int *x0, int *y0, int *x1, int *y1, int
    word_len);
// Function to search for the word diagonally (bottom-right to top-left)
int find_diagonal_bottom_right_to_top_left(char grid[MAX_ROWS][MAX_COLUMNS],
     int rows, const char *word, int *x0, int *y0, int *x1, int *y1, int
    word_len);
// Function to search for the word diagonally (top-right to bottom-left)
int find_diagonal_top_right_to_bottom_left(char grid[MAX_ROWS][MAX_COLUMNS],
     int rows, const char *word, int *x0, int *y0, int *x1, int *y1, int
    word_len);
// Main function
int exe_solver(int argc, char *argv[], int *x0, int *y0, int *x1, int *y1) ;

#endif // SOLVER_H
```

Code 8 – Contenu de solver.h

8.8 Annexe 8 : Fonction exec_solver

```c
int exe_solver(int argc, char *argv[], int *x0, int *y0, int *x1, int *y1) {
    if (argc != 3) {
        printf("Usage: %s <grid_file> <word>\n", argv[0]);
        return 1; // Return 1 for incorrect usage
    }
    char grid[MAX_ROWS][MAX_COLUMNS];
    int rows = read_grid(grid, argv[1]);
    if (rows == 0) {
        printf("Error reading the grid.\n");
        return 1; // Return 1 for error in reading the grid
    }
    char word[MAX_COLUMNS];
    strcpy(word, argv[2]);
    convert_to_uppercase(word);
    int word_len = strlen(word);
    // Check for the word in all directions
    if (find_horizontal(grid, rows, word, x0, y0, x1, y1, word_len) ||
    find_horizontal_right_to_left(grid, rows, word, x0, y0, x1, y1,
    word_len) ||
    find_vertical_top_to_bottom(grid, rows, word, x0, y0, x1, y1,
    word_len) ||
    find_vertical_bottom_to_top(grid, rows, word, x0, y0, x1, y1,
    word_len) ||
    find_diagonal_top_left_to_bottom_right(grid, rows, word, x0, y0, x1, y1,
    word_len) ||
    find_diagonal_bottom_left_to_top_right(grid, rows, word, x0, y0, x1, y1,
    word_len) ||
    find_diagonal_bottom_right_to_top_left(grid, rows, word, x0, y0, x1, y1,
    word_len) ||
    find_diagonal_top_right_to_bottom_left(grid, rows, word, x0, y0, x1, y1,
    word_len)){
        // Word found
        printf("(%d,%d) (%d,%d)\n", *x0,*y0,*x1,*y1);
        return 0; // Indicate success
    } else {
        printf("Not found\n");
        *x0 = -1;
        *y0 = -1;
        *x1 = -1;
        *y1 = -1;
        return 0; // Return 0 even if not found
    }
}
```

Code 9 – Fonction exe_solver

8.9 Annexe 9 : Structure de nos couches de neuronnes

```

// Structure of a layer -> multiple neurons
typedef struct {
    Neuron *neurons;    // Pointer to an array of Neuron structures,
       representing the neurons in this layer
    int num_neurons;     // Number of neurons in this layer
} Layer;
```

Code 10 – Définition de la structure d'une couche de neurones

8.10 Annexe 10 : Création d'un neurone et d'une couche de neurones

```
// Creation of a neuron (random initialization)
Neuron create_neuron(int num_inputs) {
    Neuron neuron;
    neuron.num_weights = num_inputs;
    neuron.weights = malloc(num_inputs * sizeof(double));
    for (int i = 0; i < num_inputs; i++)
    {
        neuron.weights[i] = ((double)rand() / RAND_MAX) * 2 - 1;
        // Random between -1 and 1
    }
    neuron.bias = ((double)rand() / RAND_MAX) * 2 - 1; // Random bias
    neuron.output = 0;
    neuron.delta = 0;
    return neuron;
}

// Creating a layer with number of neurons
Layer create_layer(int num_inputs, int num_neurons) {
    Layer layer;
    layer.neurons = malloc(num_neurons * sizeof(Neuron));
    layer.num_neurons = num_neurons;
    for (int i = 0; i < num_neurons; i++)
    {
        layer.neurons[i] = create_neuron(num_inputs);
    }
    return layer;
}
```

Code 11 – Création d'un neurone et d'une couche de neurones