

Rapport de Projet



EPITA 2028: Projet S3
Groupe CANA

RAMSTEIN Antoine
MULLER Célian
CORNU Noé
MEYER Aurélien

TABLE DES MATIÈRES

1	Introduction	2
2	Présentation du groupe	3
3	Répartition des charges	5
4	Etat d'avancement du projet	7
4.1	Chargement d'une image et suppression des couleurs	7
4.2	Prétraitement complet	10
4.3	Détection de la grille	16
4.4	Détection et Extraction	18
4.5	Algorithme de résolution d'une grille de mots cachés	23
4.6	Concept du réseau de neurones : NXOR	25
4.7	Réseau de neurones	27
4.8	Interface graphique	30
5	Aspect technique	33
5.1	Prétraitement	33
5.2	Détection de la grille et extraction des mots	34
5.3	Concept de réseau de neurones : NXOR	39
5.4	Réseau de neurones	40
5.5	Solver	44
6	Conclusion	46

1 INTRODUCTION

L'objectif de ce projet est de réaliser un logiciel de type OCR (Optical Character Recognition) qui résout une grille de mots cachés. L'application prendra en entrée une image représentant une grille de mots cachés et affichera en sortie la grille résolue. Dans sa version définitive, l'application devra proposer une interface graphique permettant de charger une image dans un format standard, de la visualiser, de corriger certains de ses défauts, d'afficher la grille complètement remplie et résolue, un prétraitement complet, un réseau de neurones complet et fonctionnel, la reconstruction de la grille, de la liste de mots et de la grille, l'affichage de la grille, la sauvegarde du résultat et une interface graphique. Le résultat devra également pouvoir être sauvegardé. L'objectif est de montrer l'état d'avancement de notre projet pour la soutenance finale.

2 PRÉSENTATION DU GROUPE

Notre groupe se compose de 4 membres : Noé CORNU, Célian MULLER, Aurélien MEYER et Antoine RAMSTEIN. Ce projet a pour but de renforcer notre capacité à travailler efficacement en groupe tout en perfectionnant notre aptitude à gérer un projet complexe. Cette expérience nous enseignera des compétences essentielles en gestion de projet, telles que la planification, la répartition des tâches, la communication et la collaboration au sein de l'équipe. Pour le bon déroulement du projet, nous avons décidé de mettre en place des réunions, le plus souvent toutes les semaines, pour être le plus efficace dans l'avancement du projet. En somme, ce projet constitue une opportunité d'apprentissage holistique dans le domaine de la technologie et de l'informatique, nous préparant ainsi à des défis professionnels futurs.

Célian

L'informatique me motive profondément pour son potentiel d'innovation et d'impact dans divers secteurs. J'apprécie particulièrement les défis techniques, comme ce projet de WorldSearchSolver. J'aime l'idée que, grâce aux technologies, nous pouvons améliorer des systèmes, optimiser des processus, et transformer la manière dont nous vivons et travaillons. Dans le cadre de ce projet, je suis particulièrement intéressé par l'apprentissage d'un réseau de neurones et la création d'interface graphique. Cet aspect est très motivant pour moi car il me permet de mettre en pratique mes compétences vu en cours magistraux et d'apprendre à collaborer avec d'autres étudiants, ce qui est essentiel pour mon développement professionnel. Ainsi, ce projet est une source de motivation et d'épanouissement puisqu'il me permet d'apprendre d'avantage des notions qui me serviront dans le monde professionnel.

Noé

Ce projet me permet d'acquérir une expérience précieuse, bien différente de celle de projets classiques pour lesquels de nombreuses ressources existent déjà en ligne. Ici, le caractère unique du World Search Solver rend les recherches de solutions plus difficiles, car peu de projets similaires sont disponibles pour s'en inspirer. Cette absence de repères nécessite de développer une réflexion autonome, de sortir des méthodes établies et de penser de manière créative pour construire un algorithme efficace. Ce processus, bien qu'exigeant, va me per-

mettre d'affiner ma capacité à résoudre des problèmes de manière innovante, renforçant ainsi ma confiance en mes propres capacités. En somme, ce projet m'offre une occasion de grandir techniquement et personnellement, tout en stimulant mon autonomie et ma persévérance face aux défis complexes.

Aurélien

je suis passionné par la cybersécurité et la programmation, avec une ambition pour le milieu militaire. Ce qui me motive particulièrement, c'est l'idée de relever des défis complexes et d'utiliser mes compétences pour sécuriser des systèmes.

Dans ce projet d'OCR, je suis en charge du réseau de neurones, une chose que je n'ai jamais faite, mais qui m'intéresse beaucoup car cela permet d'allier analyse de données et apprentissage automatique. Ces compétences sont cruciales pour comprendre et anticiper le comportement des systèmes informatiques, ce qui est essentiel en cybersécurité.

Travailler sur ce réseau de neurones me permet de renforcer mes compétences en programmation et en analyse, des compétences qui me seront utiles pour une carrière en sécurité informatique. De plus, ce projet est d'autant plus motivant du fait de travailler en groupe, ce qui rend l'expérience plus enrichissante. Nous avons déjà travaillé ensemble par le passé ce qui facilite la communication et l'entraide. Nous pouvons donc aborder des problèmes sous différents angles et apprendre les uns des autres, ce qui nous prépare à travailler efficacement en équipe pour le monde professionnel.

Antoine

Passionné par la programmation, je me spécialise actuellement dans le langage C. Dans le cadre de mes études, je travaille sur un projet de reconnaissance optique de caractères (OCR) pour les mots fléchés. Ce projet me permet de développer des compétences en traitement d'images et en analyse de texte, tout en approfondissant ma maîtrise du C. Mon objectif est de me spécialiser dans des domaines comme l'intelligence artificielle et le développement logiciel, avec toujours la volonté de relever de nouveaux défis techniques.

3 RÉPARTITION DES CHARGES



	Noé	Célian	Aurélien	Antoine
Prétraitement complet		X		
Détection	X			
Découpage	X			
Rotation automatique	X			
Solver				X
Réseau de neurones			X	
Interface graphique				X

Pour cette soutenance, nous avons décidé de répartir les tâches ainsi. En effet, puisque la détection le découpage et la rotation sont intrinsèquement liés, Noé a pris la décision de faire ces 3 tâches. Par ailleurs, Aurélien et Antoine ont des tâches bien différentes et ont donc eu une seule tâche. Comme le prétraitement a été réalisé plus rapidement que prévu, Célian a pu aider Noé, Aurélien et Antoine en liant découpage des mots, réseau de neurones et solver, ce qui nous a fait gagner du temps, pour la suite.

4 ETAT D'AVANCEMENT DU PROJET

4.1 Chargement d'une image et suppression des couleurs

Tâche réalisée par Célian MULLER

Pour charger une image nous utilisons les bibliothèques SDL et SDL-image. SDL fonctionne en utilisant un système de surfaces et de textures. En effet, une image peut être stockée en mémoire (surface) ou être utilisée par la carte graphique via le moteur de rendu (texture). SDL permet, dans un premier temps, de charger une image depuis un fichier et, par la suite, de la modifier. On peut redimensionner l'image pour l'ajuster à une partie de la fenêtre, gérer la transparence d'une image et la faire pivoter ou la transformer. Ainsi, SDL et SDL-image prennent en charge une grande variété de formats d'image (bmp, png, jpeg, gif, ...), et permettent un rendu graphique optimisé via la carte graphique, et facilitent la gestion multi-plateforme (Windows, Linux, macOS).

L'algorithme de suppression de couleur permet de filtrer les différentes couleurs de l'image pour ne rendre que du blanc et du noir. Cette tâche permet donc de faciliter la détection de la grille et des mots qui serviront pour le réseau de neurones. Le filtre permet aussi de rendre l'image plus nette, d'enlever les imperfections de celle-ci, et dans certains cas de supprimer des parties de l'image pour ne laisser passer que les mots et la grille de mot. Dans un premier temps, nous laissons apparaître un filtre noir et blanc sur les images qui en ont besoin.

➡ Find the words below and circle them (across, down, or diagonal):

- | | | |
|--------------------------------|-------------------------------|-----------------------------|
| <input type="radio"/> FLAMINGO | <input type="radio"/> TOUCAN | <input type="radio"/> SWAN |
| <input type="radio"/> KIWI | <input type="radio"/> PELICAN | <input type="radio"/> EAGLE |
| <input type="radio"/> TURKEY | <input type="radio"/> PEACOCK | |
| <input type="radio"/> CARDINAL | <input type="radio"/> PIGEON | |
| <input type="radio"/> PARROT | <input type="radio"/> SEAGULL | |



NAME: _____

DATE: _____

Strange Words

WORD FIND

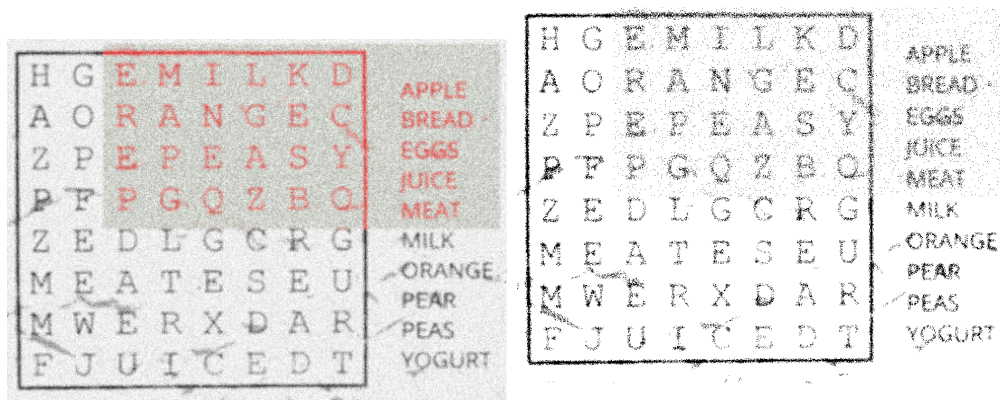


Y I M Z W J C E T A V I T S E R J K M X O H Y
P A L I M P S E S T U X D T T E G C N D M K Y
R B G N O I T A L U B A N N I T N I T E P D P
O O Q I G N I K N U L E P S M E D F V T H E U
P P A N G L O S S I A N Z D C M I T R A A F S
R Y K J P E T R I C H O R N F O U N E L L E I
I H F R I P P E T Q J A N C T N V A T U O N L
O G U F S U S U R R U S X J A A J G X B S E L
C T A T T E R D E M A L I O N M S A O O K S A
E D E M O R D N I L A P U N O O T M Y B E T N
P J R C N X D W E H G N A P S M M R Y M P R I
T S X M L P E D I A N S W H U G E E G O S A M
I G N I T A V R E N E J C L M Y S T Y C I T O
O G E R Y T H R I S M A L I H H I X Z S S E U
N R C F M O H F G N Y N A L T P S C Y I G P S
R G G A I S E N M O T P Y R C S C E S D O H C

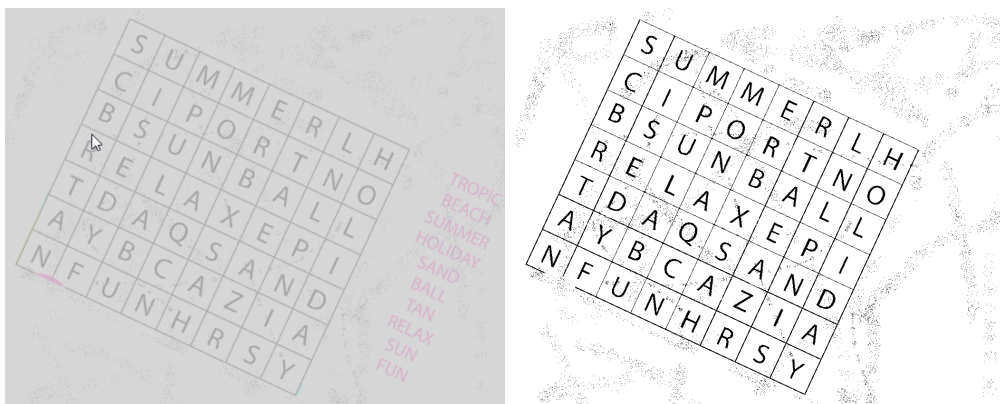
TINTINNABULATION	DEFENESTRATE	TERMAGANT
DISCOMBOBULATED	PANGLOSSIAN	SUSURRUS
OMPHALOSKEPSIS	ERYTHRISMAL	ESTIVATE
PROPRIOCEPTION	PALINDROME	SPANGHEW
TATTERDEMATION	ENERVATING	FRIPPET
PUSILLANIMOUS	PALIMPSEST	SYZYGY
CRYPTOMNESIA	SPELUNKING	TMESIS

M S W A T E R M E L O N	APPLE
Y T B N E P E W R M A E	LEMON
R R L W P A P A Y A N A	BANANA
R A N L E M O N A N E P	LIME
E W L E A P R I A B P R	ORANGE
B B I L B B W B R L A Y	WATERMELON
K E M P M A W L R A R B	GRAPE
C R E P R N R E R R G R	KIWI
A R Y A Y A O A N L A M	STRAWBERRY
L Y Y A R N E R K I W I	PAPAYA
B E B A A A N A A P R T	BLUEBERRY
Y R R E B P S A R N N W	BLACKBERRY
Y R R E B E U L B L G I	RASPBERRY
T Y P A T E A E P A C E	

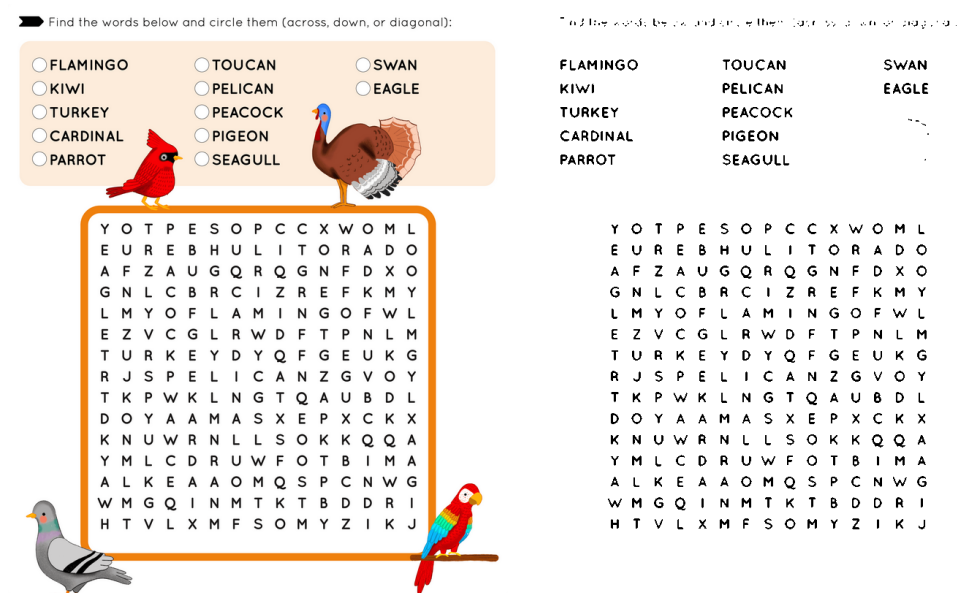
Il suffit de parcourir chaque pixel de l'image en utilisant la librairie SDL, et regarder si par rapport à une valeur seuil (choisi après différents essais) le pixel est inférieur (deviendra blanc) ou supérieur (deviendra noir).



Pour ce genre d'image, le filtre noir et blanc à lui seul ne suffisait pas. Il a fallu mettre en place un algorithme qui détecte des nuages de pixels noirs espacés et les transforme en pixels blancs pour rendre l'image la plus nette possible pour ainsi faciliter la détection de la grille et des lettres pour le réseau de neurones.



Pour faciliter la détection de grille, il est possible pour certaines images de laisser apparaître que la grille et les mots.



En utilisant un algorithme qui détecte des nuages de pixels noirs, il est possible de les filtrer et de ne laisser que la grille et les mots ce qui rend plus facile la détection de la grille par la suite. L'algorithme parcourt les groupes de pixels noirs pour déterminer leur taille. Si un groupe dépasse la taille définie en paramètre, il est remplacé par des pixels blancs.

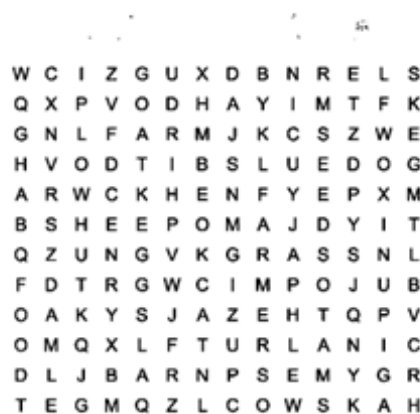
Comme la tâche de supprimer des couleurs et de charger une image était relativement simple, je me suis permis d'avancer et de commencer la suppression de bruit pour rendre l'image la plus nette possible, ce qui a fortement aidé pour la détection de la grille et des mots. Les images du niveau 2 étaient les plus dures à rendre nettes puisqu'elles comportaient beaucoup d'impuretés, ce qui rendait les images complexes à voir, dans un premier temps, et à analyser dans un second temps. De plus, concernant le chargement des images, j'imaginais cette tâche simple comparée à la suppression des couleurs, mais je n'avais pas assez bien estimé ce temps. En effet, il a fallu se familiariser avec le langage C dans un premier temps (syntaxe et Makefile), puis installer correctement les bibliothèques SDL et SDL-image sur nos ordinateurs personnels, ce qui fut une tâche délicate.

4.2 Prétraitement complet

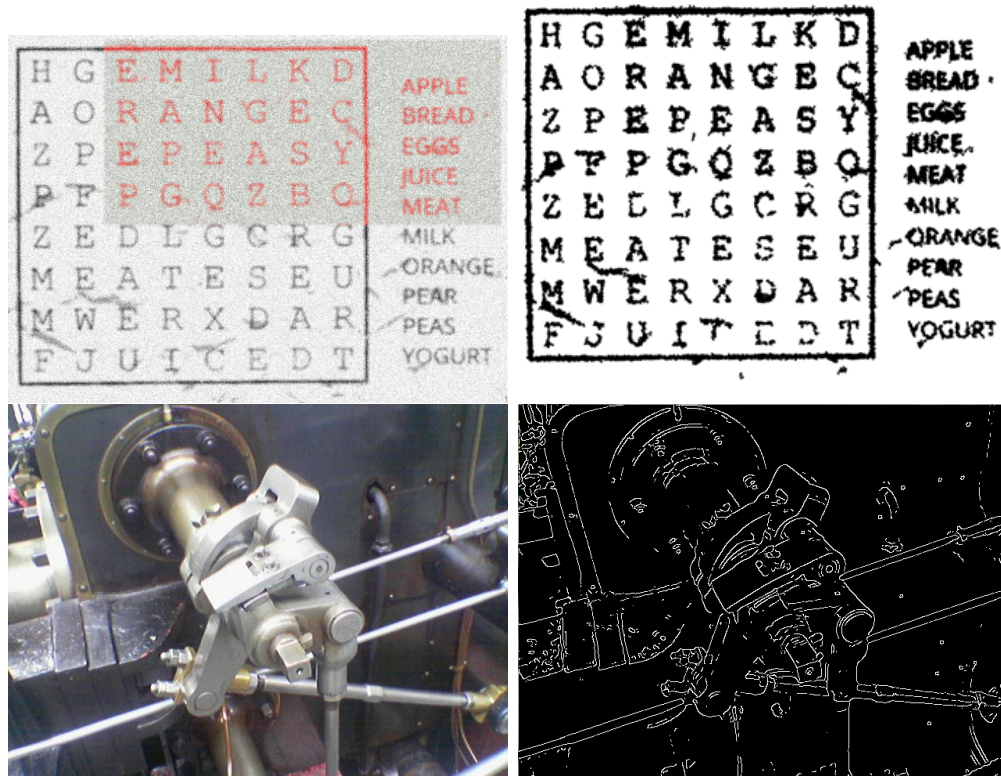
Tâche réalisée par Célian MULLER

Le but de cette partie est de rendre l'image la plus facile à détecter que ce soit la grille de

lettres ou la liste de mots. Il a fallu donc s'y prendre par différents moyens pour enlever le bruit de pixels noirs pour les différentes images.

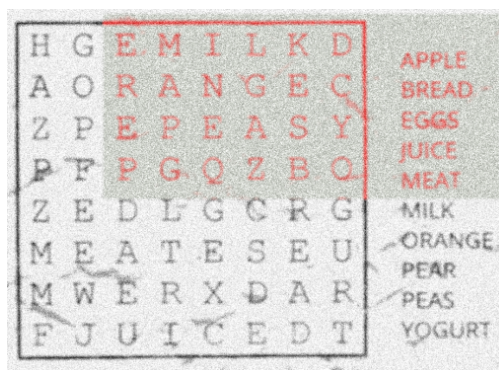
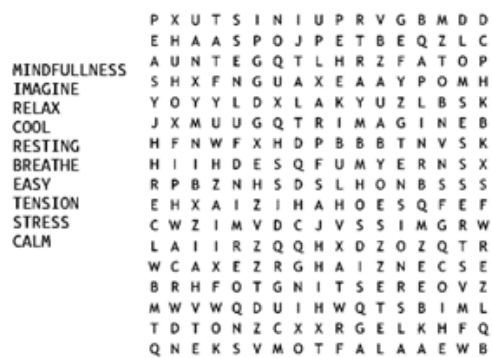


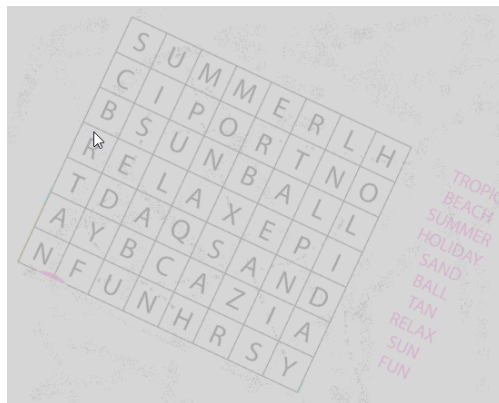
Ici, nous avons enlevé les animaux par différents algorithmes. Pour cela, il faut détecter les contours ou la luminosité afin de différencier les dessins des lettres, par des traits de pixels noirs avec des allures irrégulières. En effet, les dessins sont souvent constitués de zones contiguës. Si la région détectée est plus grande qu'un certain seuil, elle est considérée comme un dessin et est supprimée. D'autres algorithmes ont été mis en place pour supprimer le bruit plus parasite.



Exemple d'image résultante de l'application du filtre de Canny

Pour arriver à ce résultat, il a fallu faire de nombreuses recherches pour comprendre comment améliorer les filtres déjà appliqués au paravent (voir Chargement d'une image et suppression des couleurs). L'algorithme de Canny ou de Sobel ont pu nous aider à comprendre comment les améliorer. Ils permettent d'appliquer un filtre Gaussien pour réduire le bruit et appliquent une suppression non maximale pour affiner les contours. Ils permettent ainsi d'être plus précis, robuste et efficace dans la suppression d'effet indésirable pour la détection de la grille. Après plusieurs tentatives, voici le résultat final du prétraitement complet des images.





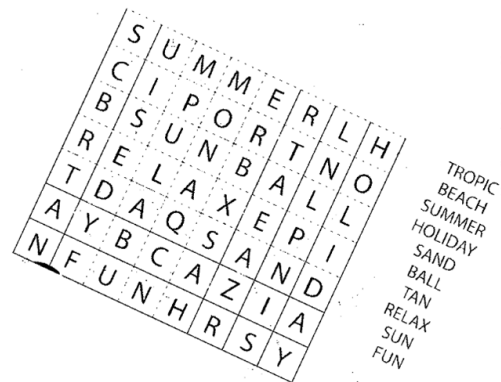
NAME: _____ DATE: _____

Strange Words WORD FIND



YIMZWJCEITAVITSERJEMXOHY
 PALIMPSESTUXDITTEGCHNDMKEY
 RBGNOITALUBANNITNITEPDP
 OQOIGNIKNULEPSMEDFVTHEU
 PPANGLOSSIANZDCMITRAAFS
 RYKJPETRICHORNFOUNELLEI
 IHFRIPPETQJANCTNVATUONL
 OGUFSSUSURRUSXJAAJGXBSSEL
 CTATTERDEMALIONMSAOKESA
 EDEMORDNILAPUNOOTMYBETN
 FJRCNXDWEHGNAPSMRMYPRI
 ISXMLPEDIANSWHUGEEGOSAM
 IGNITAVRENEJCLMYSTYCITO
 OGERYTHRISMALIHIXZSSSEU
 NRCFMOHFGNYNALTIPSCYIGPS
 RGGAISENMOTPYRCSCESDOHC

TINTINNABULATION	DEFENESTRATE	TERMAGANT
DISCOMBOLATED	PANGLOSSIAN	SUSURRUS
OMPHALOSKEPSIS	ERYTHRISMAL	ESTIVATE
PROPRIOCEPTION	PALINDROME	SPANGHEW
TATTERDEMALION	ENERVATING	FRIPPET
PUSILLANIMOUS	PALIMPSEST	SYZGY
CRYPTOMNESIA	SPELLUNKING	TMESS



NAME: _____ DATE: _____

YIMZWJCEITAVITSERJEMXOHY
 PALIMPSESTUXDITTEGCHNDMKEY
 RBGNOITALUBANNITNITEPDP
 OQOIGNIKNULEPSMEDFVTHEU
 PPANGLOSSIANZDCMITRAAFS
 RYKJPETRICHORNFOUNELLEI
 IHFRIPPETQJANCTNVATUONL
 OGUFSSUSURRUSXJAAJGXBSSEL
 CTATTERDEMALIONMSAOKESA
 EDEMORDNILAPUNOOTMYBETN
 FJRCNXDWEHGNAPSMRMYPRI
 ISXMLPEDIANSWHUGEEGOSAM
 IGNITAVRENEJCLMYSTYCITO
 OGERYTHRISMALIHIXZSSSEU
 NRCFMOHFGNYNALTIPSCYIGPS
 RGGAISENMOTPYRCSCESDOHC

TINTINNABULATION	DEFENESTRATE	TERMAGANT
DISCOMBOLATED	PANGLOSSIAN	SUSURRUS
OMPHALOSKEPSIS	ERYTHRISMAL	ESTIVATE
PROPRIOCEPTION	PALINDROME	SPANGHEW
TATTERDEMALION	ENERVATING	FRIPPET
PUSILLANIMOUS	PALIMPSEST	SYZGY
CRYPTOMNESIA	SPELLUNKING	TMESS

► Find the words below and circle them (across, down, or diagonal):

- | | | |
|--------------------------------|-------------------------------|-----------------------------|
| <input type="radio"/> FLAMINGO | <input type="radio"/> TOUCAN | <input type="radio"/> SWAN |
| <input type="radio"/> KIWI | <input type="radio"/> PELICAN | <input type="radio"/> EAGLE |
| <input type="radio"/> TURKEY | <input type="radio"/> PEACOCK | |
| <input type="radio"/> CARDINAL | <input type="radio"/> PIGEON | |
| <input type="radio"/> PARROT | <input type="radio"/> SEAGULL | |



Y O T P E S O P C C X W O M L
E U R E B H U L I T O R A D O
A F Z A U G Q R Q G N F D X O
G N L C B R C I Z R E F K M Y
L M Y O F L A M I N G O F W L
E Z V C G L R W D F T P N L M
T U R K E Y D Y Q F G E U K G
R J S P E L I C A N Z G V O Y
T K P W K L N G T Q A U B D L
D O Y A A M A S X E P X C K X
K N U W R N L L S O K K Q Q A
Y M L C D R U W F O T B I M A
A L K E A A O M Q S P C N W G
W M G Q I N M T K T B D D R I
H T V L X M F S O M Y Z I K J

► Find the words below and circle them (across, down, or diagonal):

- | | | |
|--------------------------------|-------------------------------|-----------------------------|
| <input type="radio"/> FLAMINGO | <input type="radio"/> TOUCAN | <input type="radio"/> SWAN |
| <input type="radio"/> KIWI | <input type="radio"/> PELICAN | <input type="radio"/> EAGLE |
| <input type="radio"/> TURKEY | <input type="radio"/> PEACOCK | |
| <input type="radio"/> CARDINAL | <input type="radio"/> PIGEON | |
| <input type="radio"/> PARROT | <input type="radio"/> SEAGULL | |

Y O T P E S O P C C X W O M L
E U R E B H U L I T O R A D O
A F Z A U G Q R Q G N F D X O
G N L C B R C I Z R E F K M Y
L M Y O F L A M I N G O F W L
E Z V C G L R W D F T P N L M
T U R K E Y D Y Q F G E U K G
R J S P E L I C A N Z G V O Y
T K P W K L N G T Q A U B D L
D O Y A A M A S X E P X C K X
K N U W R N L L S O K K Q Q A
Y M L C D R U W F O T B I M A
A L K E A A O M Q S P C N W G
W M G Q I N M T K T B D D R I
H T V L X M F S O M Y Z I K J



FARM

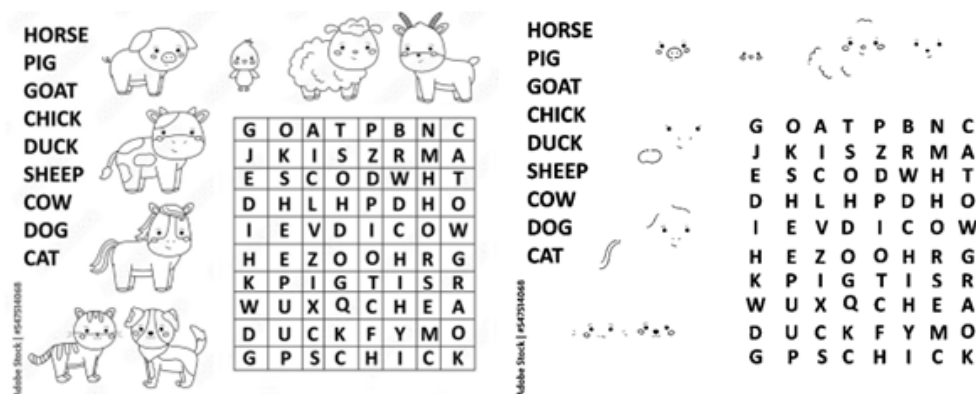


W C I Z G U X D B N R E L S
Q X P V O D H A Y I M T F K
G N L F A R M J K C S Z W E
H V O D T I B S L U E D O G
A R W C K H E N F Y E P X M
B S H E E P O M A J D Y I T
Q Z U N G V K G R A S S N L
F D T R G W C I M P O J U B
O A K Y S J A Z E H T Q P V
O M Q X L F T U R L A N I C
D L J B A R N P S E M Y G R
T E G M Q Z L C O W S K A H



W C I Z G U X D B N R E L S
Q X P V O D H A Y I M T F K
G N L F A R M J K C S Z W E
H V O D T I B S L U E D O G
A R W C K H E N F Y E P X M
B S H E E P O M A J D Y I T
Q Z U N G V K G R A S S N L
F D T R G W C I M P O J U B
O A K Y S J A Z E H T Q P V
O M Q X L F T U R L A N I C
D L J B A R N P S E M Y G R
T E G M Q Z L C O W S K A H





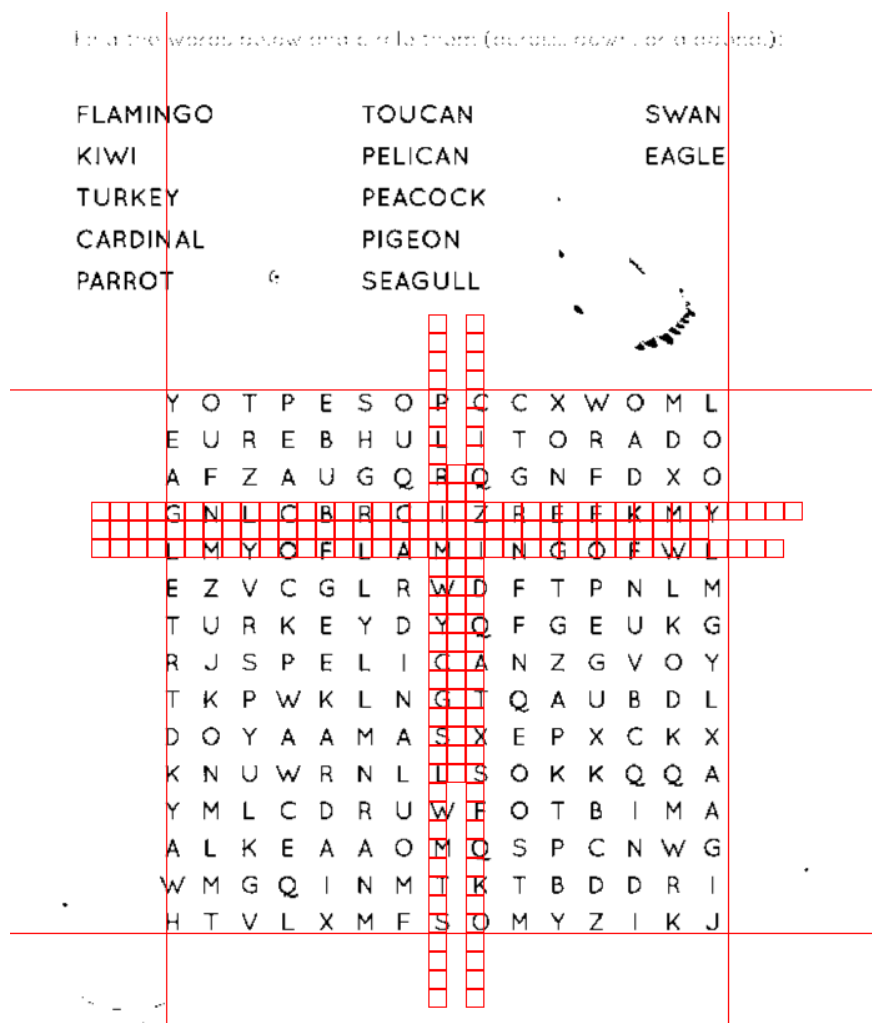
Comme le prétraitement complet n'était pas attendu pour la première soutenance, j'ai donc pu m'accorder plus de temps sur des cas plus particulier, ce qui m'a permis d'être plus efficace et de proposer des aides pour la détection de la grille et des mots. En somme, le prétraitement complet de l'image n'a pas été insurmontable, car j'avais pris de l'avance pour la première soutenance.

4.3 Détection de la grille

Tâche réalisée par Noé CORNU

Dans le cadre du projet WorldSearchSolver, j'ai pris en charge la détection de la matrice de lettres ainsi que l'extraction de la liste de mots. La principale difficulté résidait dans l'absence de grille visible dans l'image, ce qui a limité l'utilisation de méthodes standard souvent appliquées aux grilles structurées (par exemple, les solveurs de Sudoku). Après avoir étudié diverses techniques disponibles, j'ai identifié qu'il serait nécessaire de développer une méthode adaptée spécifiquement à notre projet.

En analysant les images fournies, j'ai constaté que la matrice de lettres était systématiquement centrée. En partant du centre de l'image, j'ai développé un algorithme de propagation par zone : il parcourt l'image dans toutes les directions en divisant l'espace en carrés de pixels. Lorsque la densité de pixels noirs diminue en dessous d'un seuil défini, cela marque la fin de la matrice. Cette approche permet de délimiter avec précision la grille de lettres.



Lors de la détection de la grille et de la liste de mots, la complexité des images traitées a présenté des défis variés selon leur niveau. Les images des premiers niveaux sont relativement simples, permettant une détection efficace et rapide avec notre algorithme initial, sans ajustements particuliers.

Les images de niveau 2, en revanche, introduisent une difficulté supplémentaire en raison de leur bruit visuel. Actuellement, ces interférences empêchent une détection fiable, tant pour la grille que pour la liste de mots. Pour y remédier, nous travaillons sur l'intégration de filtres d'image visant à réduire le bruit et améliorer la lisibilité des caractères. Par ailleurs, certaines images présentent une inclinaison qui complique l'analyse, nécessitant des ajuste-

ments de notre approche afin de mieux aligner la grille lors de la détection.

Pour les images de niveau 3, qui comprennent des éléments visuels plus complexes, les filtres que nous avons appliqués jusqu'à présent s'avèrent majoritairement efficaces pour éliminer les artefacts gênants. Cependant, un cas particulier persiste : les listes de mots divisées en plusieurs colonnes. Pour ces images, il a été nécessaire d'adapter l'algorithme de recherche des mots en vérifiant autour de la première colonne détectée la présence d'autres mots à gauche ou à droite. Si une autre colonne de mots est identifiée, nous appliquons le même algorithme de segmentation sur chaque colonne.

Enfin, pour les deux dernières images, la détection de la grille s'est avérée réalisable sans difficulté majeure. La liste de mots, cependant, pose encore quelques problèmes en raison de l'alignement irrégulier des mots ou de la présence d'éléments perturbateurs. Ces derniers peuvent être confondus avec les listes de mots et nécessitent un affinement de notre approche pour distinguer clairement les mots des autres formes visuelles.

4.4 Détection et Extraction

Tâche réalisée par Noé CORNU

La détection de la grille de lettres est l'une des étapes fondamentales du projet WorldSearch-Solver. L'objectif principal était de localiser précisément la position de la grille dans une image, qu'elle soit visible (avec des traits délimitant les cellules) ou invisible (sans traits), tout en assurant la flexibilité face à des cas complexes tels que des grilles inclinées ou des images bruyantes. Cette étape était essentielle pour découper chaque lettre de la grille de manière isolée et structurer les données en vue de leur utilisation ultérieure dans un réseau neuronal.

Les principaux défis identifiés incluaient :

1. La variabilité des formats d'images (grilles visibles ou invisibles).
2. La rotation éventuelle des grilles dans certaines images.
3. Les bruits visuels et les imperfections qui perturbent la détection.

4. Les dimensions irrégulières des cellules dans certains cas.

Pour relever ces défis, une série d'algorithmes spécifiques a été conçue, testée et optimisée.

1. Gestion des grilles visibles

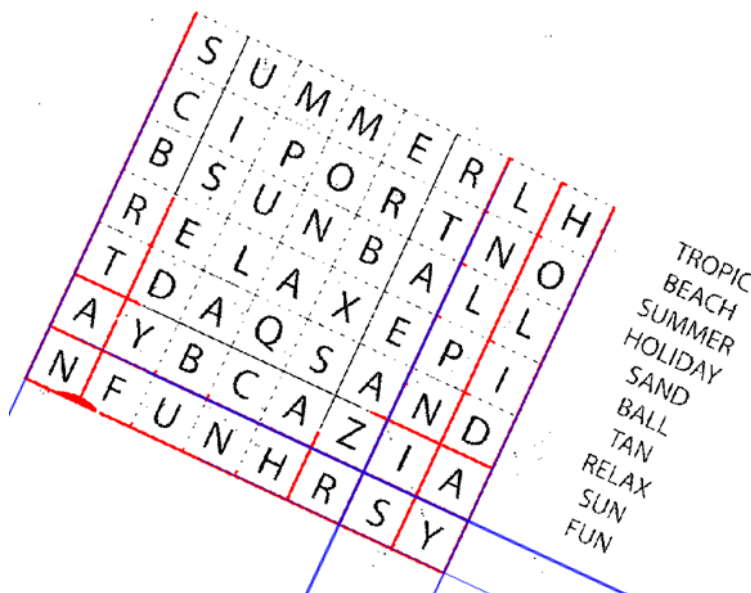
Lorsqu'une grille visible était présente dans l'image, elle constituait une difficulté initiale pour l'algorithme de détection. Afin de simplifier le processus, une fonction spécifique a été développée pour rendre ces traits invisibles. Cette fonction agit en supprimant les pixels noirs des lignes de la grille tout en conservant ceux appartenant aux lettres. Une fois les traits éliminés, l'image devient équivalente à celle d'une grille invisible, permettant ainsi d'appliquer l'algorithme général de détection sans modification supplémentaire. Cette approche homogène améliore la robustesse du processus tout en réduisant la complexité algorithmique.

2. Détection de la grille par propagation

L'algorithme principal repose sur une propagation par zones à partir du centre de l'image. Ce choix est motivé par la structure typique des images, où la grille de lettres est souvent centrée. L'image est divisée en blocs de pixels carrés. Pour chaque bloc, la densité de pixels noirs est calculée. Si cette densité dépasse un seuil prédéfini, le bloc est considéré comme faisant partie de la grille. La propagation continue dans les quatre directions (haut, bas, gauche, droite) jusqu'à ce que des blocs contenant majoritairement des pixels blancs soient rencontrés. Ces blocs marquent les bords de la grille.

3. Gestion des grilles inclinées

Un des cas qui faisait échouer l'algorithme initial concernait les grilles inclinées. Pour résoudre ce problème, un algorithme de rotation automatique a été conçu. L'algorithme identifie les traits de la grille (horizontaux ou verticaux) en analysant les alignements des pixels noirs. Une fois les traits détectés, l'angle d'inclinaison est calculé à l'aide de techniques géométriques. L'image est ensuite redressée pour aligner la grille avec les axes verticaux et horizontaux. Cette rotation garantit que la détection fonctionne même pour des grilles mal orientées.



4. Uniformisation des tailles

Pour gérer la diversité des tailles de grilles dans les différentes images, toutes les images sont redimensionnées à une taille standard avant d'appliquer l'algorithme. Cela permet de simplifier les calculs tout en garantissant des résultats homogènes, quel que soit le format initial de l'image.

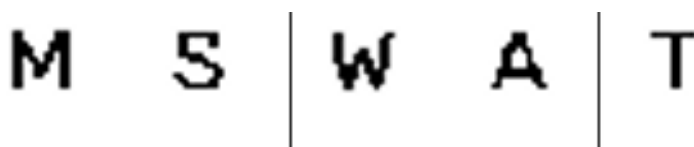
5. Détermination des dimensions de la grille

Une fois la grille détectée, une fonction calcule le nombre de lignes et de colonnes en parcourant l'image ligne par ligne et colonne par colonne pour identifier les transitions régulières de pixels noirs. Ces transitions correspondent aux séparations entre les lettres. La taille totale de la grille est ensuite divisée par le nombre de lignes et de colonnes détectées pour estimer la taille de chaque cellule.

L'algorithme est capable de gérer des grilles irrégulières, comme celles présentant des cellules de tailles variables (ex. image 2 du niveau 4), sans nécessiter d'ajustements spécifiques. Performances : La détection a été optimisée pour s'exécuter en moins de 0,25 seconde, offrant une expérience perçue comme instantanée par l'utilisateur.

Une fois la grille détectée et délimitée, chaque cellule contenant une lettre est isolée pour être sauvegardée individuellement. Ce processus revêt une importance cruciale pour la

préparation des données destinées à l'entraînement et aux tests du réseau neuronal. Le processus d'extraction se décompose en deux phases principales.

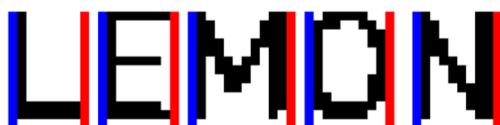


Lettres de la grille

La première phase concerne l'extraction des lettres directement présentes dans la grille. Chaque lettre est soigneusement extraite et sauvegardée sous forme d'image au format PNG, choisi pour sa qualité d'image et sa compatibilité optimale avec les outils d'apprentissage automatique. Une nomenclature explicite a été développée pour nommer ces images : chaque fichier est intitulé cell-X-Y.png, où X et Y représentent précisément les coordonnées de la cellule au sein de la grille.

Lettres des mots

La seconde phase porte sur le traitement de la liste de mots située à l'extérieur de la grille. Cette liste fait l'objet d'une analyse approfondie où chaque mot est segmenté en lettres individuelles. Ces lettres sont ensuite sauvegardées selon une nomenclature similaire : word-A-letter-B.png, où A indique la position du mot dans la liste et B la position de la lettre au sein du mot.



Cette méthode d'extraction et de nommage présente plusieurs avantages significatifs. La nomenclature claire facilite grandement le débogage, permettant de localiser rapidement une lettre ou un mot spécifique pour vérification. De plus, les images extraites peuvent être directement utilisées pour l'entraînement du réseau neuronal, sans nécessiter de transformations supplémentaires, ce qui optimise considérablement le processus de préparation des données. Le projet a été contraint d'utiliser la bibliothèque SDL, disponible uniquement sur les ordinateurs de l'école. Bien que ses fonctionnalités soient limitées comparées aux bibliothèques modernes, SDL a permis de lire, modifier et sauvegarder efficacement les images.

Problèmes rencontrés et solutions

L'absence initiale de guidance méthodologique a représenté un défi majeur. Contrairement aux projets classiques, aucune ressource spécifique n'était disponible, exigeant une réflexion approfondie pour concevoir des solutions originales adaptées aux contraintes spécifiques du projet.

La gestion de cas complexes a nécessité des ajustements techniques pointus. Les images bruitées ont particulièrement sollicité l'expertise de l'équipe, nécessitant des réglages fins des seuils de densité et de propagation pour garantir une détection fiable. Les alignements irréguliers de mots, notamment ceux répartis sur plusieurs colonnes, ont demandé une adaptation algorithmique sophistiquée pour identifier correctement les différentes sections.

Une optimisation rigoureuse des performances a été menée, visant à réduire le temps d'exécution de l'algorithme sous le seuil critique de 0,25 seconde. Cette optimisation a impliqué des ajustements complexes dans le calcul des densités, l'utilisation de structures de données efficaces et une gestion précise des zones de détection.

La validation des résultats a été conduite avec une rigueur méthodologique. Chaque image extraite a fait l'objet d'une vérification manuelle approfondie, garantissant sa qualité et sa compatibilité avec le réseau neuronal. Les tests de performance ont démontré que l'algorithme de détection pouvait gérer une large gamme d'images, des plus simples aux plus complexes, tout en respectant les contraintes de rapidité initialement fixées.

Le travail réalisé sur la détection et l'extraction constitue une base solide pour les étapes ultérieures du projet, notamment la reconnaissance des caractères par le réseau neuronal et la résolution des grilles.

4.5 Algorithme de résolution d'une grille de mots cachés

Tâche réalisée par Antoine RAMSTEIN

L'objectif de cette partie du projet est de développer un programme en C appelé `solver`, capable de résoudre des grilles de mots cachés. Le programme doit lire une grille de caractères depuis un fichier, rechercher un mot donné dans cette grille, et retourner les coordonnées de la première et dernière lettre du mot s'il est trouvé. Cette partie permet de travailler sur la manipulation de fichiers, la gestion des chaînes de caractères et la mise en œuvre de techniques de recherche multi-directionnelle dans une grille.

Cette version complète devrait couvrir tous les aspects de ton travail. Si tu as d'autres détails ou exemples spécifiques à inclure, je peux les intégrer.

1. Entrées du Programme :

- Le programme doit être exécuté en ligne de commande et accepter deux arguments :
 - Nom du fichier contenant la grille de caractères.
 - Mot à rechercher dans la grille.
- La grille est un fichier texte formaté, où chaque ligne représente une rangée de la grille et chaque caractère est une lettre majuscule (A-Z).
- Le mot à rechercher peut être donné en majuscules ou en minuscules, sans distinction de casse.

2. Format de la Grille :

- La grille est composée de lettres majuscules et doit avoir un format rectangulaire avec un minimum de 5 lignes et 5 colonnes.
- Chaque ligne du fichier représente une ligne de la grille, avec des caractères séparés par des espaces ou collés.

3. Recherche du Mot :

- Le programme doit rechercher le mot en suivant huit directions possibles à partir de chaque lettre dans la grille :

- Horizontalement : de gauche à droite et de droite à gauche.
- Verticalement : de haut en bas et de bas en haut.
- Diagonalement :
 - * Haut gauche vers bas droit.
 - * Bas gauche vers haut droit.
 - * Haut droit vers bas gauche.
 - * Bas droit vers haut gauche.
- Si le mot est trouvé, le programme retourne les coordonnées de la première et dernière lettre du mot dans la grille.
- Si le mot n'est pas trouvé, le programme affiche "Not found".

4. Format de la Sortie :

- **Mot trouvé** : Si le mot est trouvé, le programme affiche les coordonnées dans le format $(x_0, y_0) (x_1, y_1)$, où :
 - (x_0, y_0) sont les coordonnées de la première lettre du mot dans la grille.
 - (x_1, y_1) sont les coordonnées de la dernière lettre du mot.
- **Mot non trouvé** : Si le mot n'est pas trouvé dans la grille, le programme affiche simplement "Not found".
- Les coordonnées $(0, 0)$ correspondent au coin supérieur gauche de la grille.

Lorsqu'un mot est détecté, ses coordonnées de départ et d'arrivée sont enregistrées, permettant ainsi de tracer le mot sur une image de la grille lors de la phase de visualisation. Une particularité importante de ce module réside dans l'intégration de l'algorithme de distance de Levenshtein, qui calcule le nombre minimal d'opérations nécessaires pour transformer un mot donné en un autre. Cette fonctionnalité s'avère essentielle dans les cas où des erreurs, telles que des fautes de frappe ou des inexactitudes dues à un module de reconnaissance précédent, rendent impossible la localisation exacte d'un mot. L'algorithme suggère alors des correspondances proches, garantissant une expérience utilisateur plus complète et résiliente.

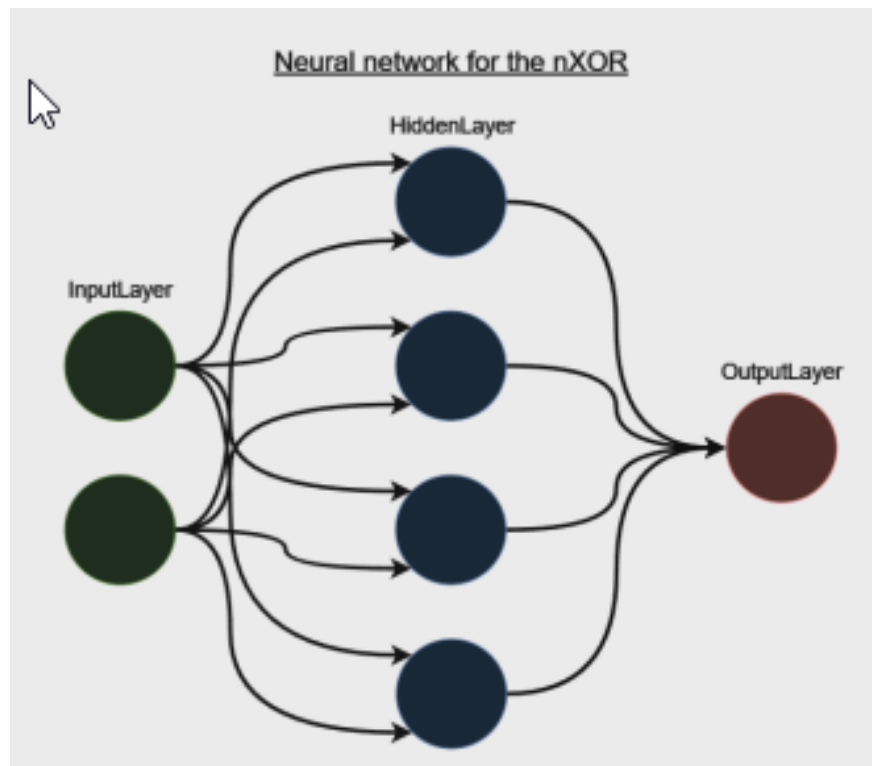
```
(antoine@DESKTOP-2NSP0IL) ~/Documents/epita-prepa-computer-science
$ ./solver test words
Mot trouvé : APPLE (8,3) -> (4,3)
Mot trouvé : LEMON (3,3) -> (3,7)
Mot trouvé : BANANA (5,5) -> (10,5)
Mot trouvé : LIME (4,2) -> (7,2)
Mot trouvé : ORANGE (8,6) -> (13,11)
Mot trouvé : WATERMELON (0,2) -> (0,11)
Mot trouvé : GRAPE (7,10) -> (3,10)
Mot trouvé : KIWI (9,8) -> (9,11)
Mot trouvé : STRAWBERRY (0,1) -> (9,1)
Mot trouvé : PAPAYA (2,4) -> (2,9)
Mot trouvé : BLUEBERRY (12,8) -> (12,0)
Mot trouvé : BLACKBERRY (10,0) -> (1,0)
Mot trouvé : RASPBERRY (11,8) -> (11,0)
Mot non trouvé : TYPE. Suggestion proche : (0,4) -> (3,4), Distance : 1
Image sauvegardée dans 'redrawn_output.bmp'.
```

4.6 Concept du réseau de neurones : NXOR

Tâche réalisée par Aurélien MEYER

La conception du réseau de neurones a tout d'abord commencé par des recherches notamment sur le fonctionnement de celui-ci.

Pour cette première soutenance, mon avancement sur le développement du réseau de neurones devait se traduire par un exemple concret : faire apprendre à l'intelligence artificielle une fonction simple, celle du nXOR. Pour y parvenir, mon réseau de neurones devait ressembler à ceci:



-2 inputs d'entrée chacun pouvant prendre la valeur 0 ou 1

-4 couches cachées

-1 input de sortie pouvant prendre la valeur 0 ou 1

Je devais me munir d'une base de données pour entraîner mon programme. Dans ce cas, cela n'a pas été compliqué étant donné la simplicité de la fonction à reproduire.

A	B	$A \leftrightarrow B$ (XNOR)
0	0	1
0	1	0
1	0	0
1	1	1

Pour chaque paire d'input, je connaissais l'output attendue. J'ai donc pu répéter le processus d'entraînement un nombre assez conséquent de fois jusqu'à que les poids de chaque lien et

les biais de chaque neurone soient parfaitement ajustés.

Mon premier réseau de neurones a donc appris la fonction nXOR. Ainsi, pour sauvegarder son résultat, il me suffit de stocker les poids de chaque lien et les biais de chaque neurone. Le début a été très difficile en raison de la difficulté de concevoir un réseaux de neurones, mais avec du temps et des exemples, j'ai pu comprendre les principes fondamentaux, affiner les paramètres et finalement obtenir des résultats prometteurs.

4.7 Réseau de neurones

Tâche réalisée par Aurélien MEYER

Il a fallu implémenter les neurones, les couches et le réseau neuronal à la manière de la Programmation Orienté Objet du C. En effet, les structures C permettent de créer ces propres types et de les relier à des structures et des attributs. L'implémentation fait de cette manière est une avancée, car à la première soutenance nous n'avions pas encore utilisé les structures de la Programmation Orienté Objet.

```
11  typedef struct Filter {
12      int side_size;
13      double **weight;
14      double bias;
15  } Filter;
16
17  typedef struct Image {
18      int side_size;
19      double **pixels;
20      void* previous_image;
21      Filter filters;
22  } Image;
23
24  typedef struct Neuron {
25      int nbr_weight;
26      double *weight;
27      double bias;
28      double output;
29      double error;
30  } Neuron;

typedef struct Conv_Layer {
    int nbr_images;
    Image *images;
} Conv_Layer;

typedef struct Layer {
    int nbr_neurons;
    Neuron *neurons;
} Layer;

typedef struct NeuralNetwork {
    int nbr_inputs;
    int nbr_layers;
    Layer *layers;
    int nbr_conv_layers;
    Conv_Layer *conv_layers;
} NeuralNetwork;
```

En plus d'ajouter des structures basées sur la programmation orientée objet(POO), nous

avons intégré la convolution dans le réseau de neurones pour en améliorer les performances. Un réseau de neurones de convolution (CNN) est un modèle conçu pour analyser des données en grille, comme des images, en s'appuyant sur quatre étapes clés : convolution, filtre, pooling, et couches entièrement connectées (fully connected).

Convolution

d'entrée. Un filtre est une petite matrice qui parcourt l'image pour détecter des motifs simples, comme des contours ou des textures, en calculant des produits scalaires entre le filtre et les sous-parties de l'image.

Filtre

Les filtres sont des outils qui permettent de repérer des features (caractéristiques) spécifiques dans l'image, comme des bords ou des formes. Chaque filtre extrait un aspect particulier de l'image, et en combinant plusieurs filtres, le réseau détecte des structures de plus en plus complexes.

Pooling

Une opération de réduction qui diminue la taille des données tout en conservant les informations importantes. Par exemple, le maxpooling divise l'image en zones et conserve uniquement la valeur maximale de chaque zone, ce qui rend le modèle plus rapide et robuste aux variations dans l'image (comme des déformations ou des translations).

Couches entièrement connectées (fully connected)

À la fin du réseau, les données extraites sont aplaties en un vecteur et passées dans des couches classiques où chaque neurone est connecté à tous les autres, comme dans un réseau de neurones classique.

Ces étapes permettent aux CNN de traiter efficacement les images et d'apprendre à reconnaître des motifs complexes pour des applications comme la reconnaissance faciale, la détection d'objets, ou la segmentation d'images.

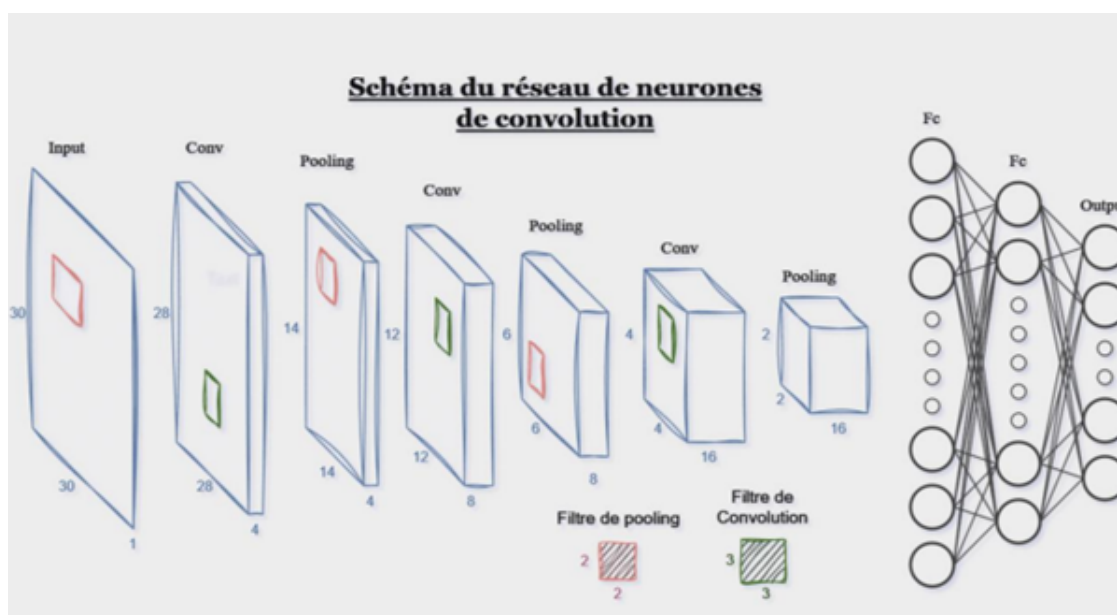
Dans notre projet, nous avons travaillé avec des inputs de 30x30 pixels, soit des images contenant 900 pixels en entrée.

Couches convolutionnelles et max-pooling

Nous avons appliqué une série de 3 couches de convolution, chacune suivie d'une étape de max-pooling. La première couche utilisait 4 filtres, la deuxième 8 filtres, et la troisième 16 filtres, permettant d'extraire des caractéristiques de plus en plus complexes à chaque étape. Après chaque couche de convolution, le max-pooling a progressivement réduit la taille des données, pour passer de 30x30 pixels à des dimensions plus petites, jusqu'à obtenir 16 images de 2x2 pixels (soit un total de 64 pixels)

Couches entièrement connectées

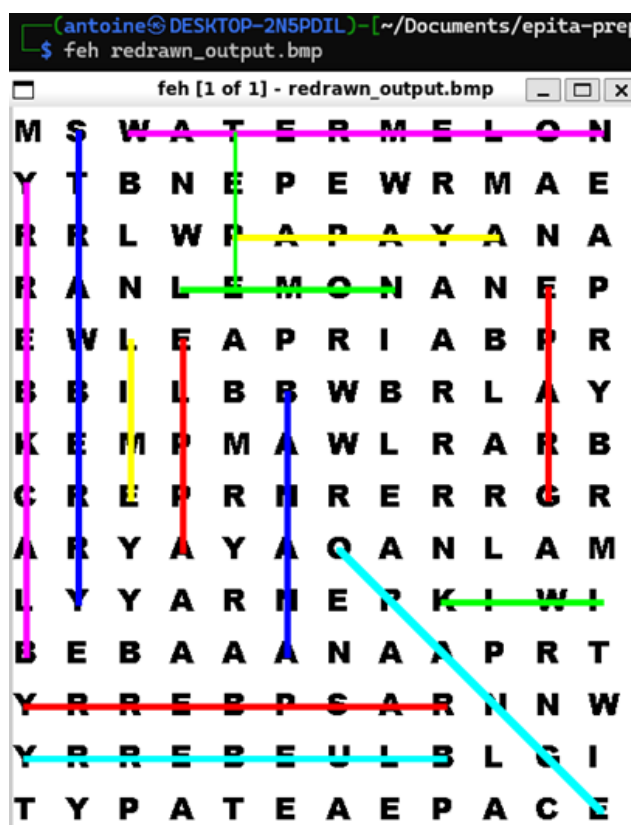
Ces données transformées ont été aplaties pour être passées dans une série de 3 couches entièrement connectées. Les couches contenaient respectivement 64, 32, et 26 neurones, permettant de combiner les informations extraites et d'effectuer une prédiction finale précise. Cette architecture nous a permis de réduire efficacement les dimensions des données tout en conservant leurs caractéristiques clés, aboutissant à un modèle performant et optimisé. Mais aussi une durée d'entraînements plus courte.



4.8 Interface graphique

Tâche réalisée par Antoine RAMSTEIN

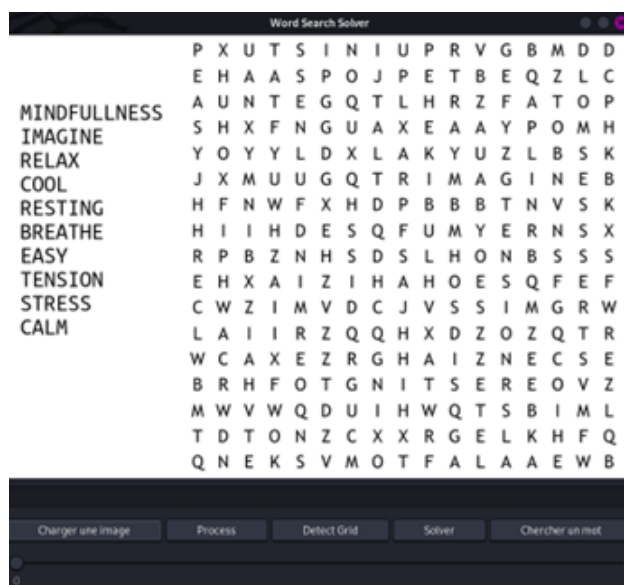
Lorsqu'un utilisateur lance le programme, les mots précédemment trouvés, grâce au solveur, peuvent être chargés et affichés directement sur la grille, évitant toute répétition inutile des calculs. Par ailleurs, le module permet une recherche ciblée, où l'utilisateur peut entrer manuellement un mot pour vérifier s'il est présent dans la grille. Cette fonctionnalité est particulièrement utile pour compléter une résolution partielle ou pour traiter des cas spécifiques où les mots à rechercher ne figurent pas dans la liste initiale.



Grâce à la bibliothèque SDL2, les mots détectés sont affichés directement sur une image de la grille, chaque mot étant mis en évidence par une couleur unique, choisie dynamiquement en fonction de son index dans la liste. Cette visualisation permet non seulement de vérifier facilement les résultats, mais elle contribue également à rendre l'expérience utilisateur plus interactive et agréable.



Cette interface, développée avec la bibliothèque GTK, centralise toutes les fonctionnalités essentielles, facilitant ainsi leur accès même pour des utilisateurs non techniques. Dès le lancement de l'application, l'utilisateur est accueilli par une interface claire et fonctionnelle, où il peut commencer par importer une grille sous forme d'image. Cette étape est rendue possible grâce à un bouton dédié qui permet de sélectionner et de charger une image depuis le système de fichiers. Une fois l'image importée, elle est affichée dans une zone centrale, offrant une vue d'ensemble claire et nette de la grille.



Une des premières fonctionnalités disponibles dans l'interface est la rotation manuelle de la grille, accessible via une barre de défilement. Cette option est particulièrement utile dans les cas où l'image de la grille n'est pas orientée correctement, par exemple si elle a été prise sous un angle ou si elle a subi une rotation involontaire. L'utilisateur peut ajuster précisément l'orientation avant de lancer l'analyse, garantissant ainsi une préparation optimale des données pour les solvers.

Une fois la grille préparée, l'utilisateur peut déclencher l'analyse en appuyant sur un bouton dédié. Les résultats de la détection des mots sont affichés presque instantanément, les mots trouvés étant tracés sur l'image de la grille avec des couleurs distinctes. Cette étape permet de visualiser rapidement quels mots ont été détectés et où ils se trouvent dans la grille. Dans les cas où un mot ne peut être trouvé dans la grille, la robustesse du système se manifeste par l'intégration de l'algorithme de Levenshtein. Lorsque la recherche échoue, l'utilisateur reçoit une suggestion de mots proches affichée dans la console, ce qui lui permet de vérifier si le mot recherché est présent sous une forme légèrement différente. Ce mécanisme de gestion des erreurs est particulièrement utile dans les situations où des incertitudes ou des erreurs se glissent dans les données initiales.

5 ASPECT TECHNIQUE

5.1 Prétraitement

Le prétraitement utilise pour l'instant 3 fonctions : `apply-black-and-white-filter`, `clean-large-black-groups` et `clean-image`. `apply-black-and-white-filter` est une fonction qui renvoie une image sous forme de pixels noirs et blancs.

$$\text{pixel} = 0.3 * \text{red} + 0.59 * \text{green} + 0.11 * \text{blue}$$

En effet, en utilisant cette expression nous pouvons convertir la couleur d'un pixel en un niveau de gris. Une fois cette valeur obtenue, il suffit de la comparer à une valeur de référence, ici 180, pour obtenir du noir ou du blanc.

`clean-large-black-groups` est une fonction permet de "nettoyer" les grands groupes de pixels noirs et de les rendre blancs. En effet, on parcourt chaque pixel pour vérifier s'il fait partie d'un groupe de pixels noirs. Pour chaque groupe de pixels noirs détectés, on utilise une fonction Flood-fill pour compter la taille du groupe et enregistrer ses coordonnées. Si la taille du groupe de pixels noirs dépasse une limite donnée en argument (ici 150), on remplace tous ses pixels par des pixels blancs, sinon on le garde tel quel.

`clean-image` fonctionne sur le même principe, mais pour les pixels trop espacés, permettant une fois supprimés de rendre l'image la plus lisible possible. La valeur arbitraire choisie est 7, s'il y a 7 pixels blancs ou plus autour d'un pixel noir, celui-ci devient blanc.

`remove-drawings` sert à supprimer les zones de dessin ou groupes de pixels noirs d'une image si leur taille dépasse un certain seuil (`size-threshold`, ici 260). Elle utilise une méthode de flood fill (remplissage par propagation) pour identifier et évaluer ces groupes de pixels noirs. Si un dessin est trop grand, il est effacé en remplaçant ses pixels par du blanc. Cela est utile, par exemple, pour nettoyer des images bruitées contenant des annotations ou dessins inutiles.

`dilate-image` réalise une dilatation d'image, une opération classique en traitement d'images. La dilatation étend les zones noires (ou traits sombres) en "coloriant" leurs pixels voisins en

noir. Cela permet d'élargir les lignes ou formes noires présentes dans une image. Dans un premier temps, on identifie chaque pixel noir de l'image temporaire, puis on modifie tous les voisins de ce pixel dans l'image originale pour les rendre noirs. Cette fonction est utilisée dans des opérations morphologiques pour renforcer les traits noirs (par exemple, sur des lettres ou des dessins, ici des lettres notamment pour l'image 2.1).

flood-fill est une implémentation de l'algorithme de flood-fill (remplissage par propagation) pour explorer une composante connectée dans une image. Elle identifie les zones de pixels noirs (ou sombres) connectés entre eux et calcule leur taille. La fonction se rappelle elle-même (récursivement) pour explorer les 8 voisins (haut, bas, gauche, droite, et les 4 diagonales). Chaque voisin est ensuite traité comme un nouveau pixel de départ, et la vérification s'applique pour chacun. Elle parcourt récursivement tous les pixels connectés, marque chaque pixel visité, et incrémente la taille totale de la composante.

5.2 Détection de la grille et extraction des mots

FONCTION : DETECT_LETTER_GRID

1. **Initialisation des bornes** : Fixe les limites de la grille (haut, bas, gauche, droite) aux valeurs maximales/minimales de l'image.
2. **Point de départ et propagation** : Définit des points de départ proches du centre de l'image. Ces points servent de repères pour propager l'algorithme dans les quatre directions (gauche, droite, haut, bas) pour identifier la grille.
3. **Propagation** :
 - Dans chaque direction (gauche, droite, haut, bas), l'algorithme compte le nombre de pixels noirs dans des blocs carrés.
 - Si un bloc contient suffisamment de pixels noirs, il est considéré comme faisant partie de la grille et les bornes sont ajustées.
 - Si un bloc contient trop de pixels blancs (en dessous d'un seuil de noirceur), l'algorithme arrête la propagation dans cette direction, marquant ainsi la limite de la grille.

4. **Fonction de comptage de pixels noirs (`count_black_pixels_in_block`)** : Compte les pixels noirs dans un bloc donné et retourne ce nombre. Les pixels noirs sont identifiés selon une tolérance.

Une fois la grille détectée, je procède au comptage du nombre de lignes et de colonnes pour découper chaque lettre. L'algorithme s'étend ensuite autour de la grille pour repérer la liste de mots, représentée par un bloc de pixels noirs distinct. Un traitement similaire est appliqué pour segmenter chaque mot en lettres individuelles, facilitant ainsi leur extraction et leur utilisation dans le solveur.

FONCTION : PROPAGATE_AND_COUNT

Cette fonction détecte les zones connectées dans une image en propageant la détection à partir d'un pixel de départ.

Variables

- **surface** : Pointeur vers l'image SDL sur laquelle la détection est effectuée.
- **x, y** : Coordonnées actuelles du pixel traité.
- **Height, Width** : Dimensions de l'image pour limiter la propagation.
- **visited** : Tableau 1D marquant les pixels déjà visités (1 pour visité, 0 sinon).
- **component** : Optionnel, tableau 1D qui enregistre les pixels de la composante connectée actuelle (1 si le pixel appartient à la composante, 0 sinon).

Étapes

1. **Vérification des bornes et de l'état du pixel** :
 - Si le pixel est hors de la grille (`is_pixel_in_grid`) ou déjà visité (`visited[y * Width + x]`), la fonction retourne 0 immédiatement.

2. Validation du pixel actif :

- `check_nearby_pixels` vérifie si le pixel ou un pixel adjacent est noir. Si ce n'est pas le cas, la fonction retourne 0.

3. Marquage du pixel :

- Le pixel est marqué comme visité dans le tableau `visited`. Si le tableau `component` est fourni, le pixel est également enregistré comme appartenant à la composante connectée.

4. Propagation aux voisins :

- La fonction explore les pixels adjacents dans quatre directions principales (haut, bas, gauche, droite) définies dans le tableau `directions`.
- Pour chaque direction, elle appelle récursivement `propagate_and_count` pour continuer la détection.

5. Retourne la taille totale de la composante connectée :

- La fonction additionne le nombre de pixels trouvés dans toutes les directions et retourne ce total.

FONCTION : WORDS_EXTRACTION

Cette fonction identifie les coordonnées des mots dans une zone rectangulaire spécifique (liste de mots).

Variables

- `surface` : Image SDL à analyser.
- `list_left`, `list_right`, `list_top`, `list_bottom` : Limites de la zone contenant la liste de mots.
- `white_threshold` : Nombre minimum de pixels noirs par ligne pour identifier un mot.

- **word_count** : Nombre total de mots attendus dans la liste.
- **flag** : Si activé, dessine des lignes horizontales pour visualiser les limites détectées.
- **words_coordinates** : Tableau contenant les limites verticales (**top_bound**, **bottom_bound**) de chaque mot détecté.

Étapes

1. Initialisation :

- Alloue de la mémoire pour stocker les coordonnées des mots (**malloc**).

2. Analyse ligne par ligne :

- Parcourt chaque ligne dans la zone définie par **list_top** et **list_bottom**.
- Compte le nombre de pixels noirs sur la ligne (**count_black_pixels_on_a_line**).

3. Détection des mots :

- Si une ligne dépasse **white_threshold** en pixels noirs :
 - Un nouveau mot commence si ce n'est pas déjà le cas (**is_black** passe à 1).
 - La limite supérieure (**top_bound**) du mot est enregistrée.
- Si une ligne est en dessous du seuil :
 - Si un mot était en cours, il est terminé (**is_black** passe à 0) et sa limite inférieure (**bottom_bound**) est enregistrée.

4. Gestion des cas spéciaux :

- Si un mot s'étend jusqu'à la dernière ligne, ses limites sont ajustées.
- Si moins de mots que prévu sont trouvés, le tableau est redimensionné avec **realloc**.

5. Retour :

- Renvoie le tableau contenant les limites des mots détectés.

FONCTION : LETTERS_EXTRACTION

Cette fonction segmente un mot en lettres individuelles et enregistre chaque lettre sous forme d'image.

Variables

- `surface` : Image SDL contenant la liste de mots.
- `list_left`, `list_right`, `list_top`, `list_bottom` : Limites de la zone contenant le mot à analyser.
- `white_threshold` : Nombre minimum de pixels noirs par colonne pour identifier une lettre.
- `current_word` : Index du mot en cours de traitement.
- `letter_left_bound`, `letter_right_bound` : Coordonnées des limites horizontales d'une lettre.
- `current_letter` : Compteur du nombre de lettres détectées.

Étapes

1. Initialisation :

- `is_black` est utilisé pour suivre si une lettre est en cours d'analyse.
- `letter_left_bound` marque le début d'une lettre.

2. Analyse colonne par colonne :

- Parcourt chaque colonne dans la zone délimitée par `list_left` et `list_right`.
- Compte les pixels noirs dans la colonne (`count_black_pixels_on_a_line`).

3. Détection des lettres :

- Si une colonne dépasse le seuil `white_threshold` :

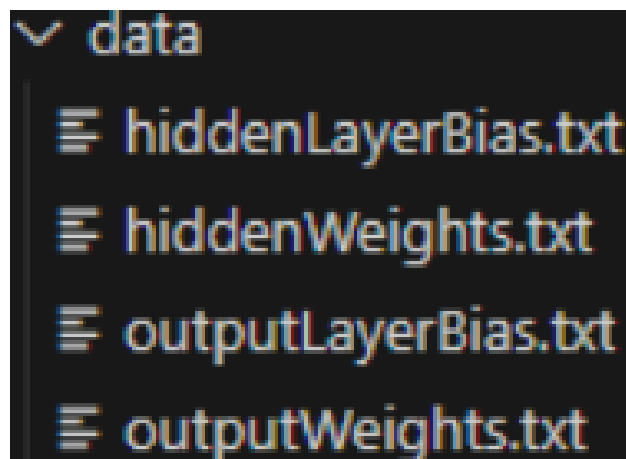
- Une nouvelle lettre commence si ce n'est pas déjà le cas (`is_black` passe à 1).
- `letter_left_bound` est défini.
- Si une colonne est en dessous du seuil :
 - Si une lettre était en cours, elle se termine (`is_black` passe à 0).
 - La limite droite (`letter_right_bound`) est enregistrée, et la lettre est extraite avec `split_word_into_images`.

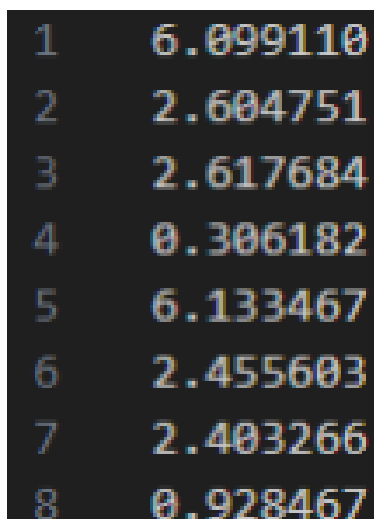
4. Retour :

- La fonction retourne le nombre total de lettres détectées.

5.3 Concept de réseau de neurones : NXOR

L'implémentation du réseau de neurone prend la forme de deux fonctions, l'une pour l'entraînement et l'autre pour son fonctionnement. Pour faciliter l'échange et la sauvegarde des données entre les deux fonctions les données comme les poids et les biais sont sauvegardés dans différents fichiers.





1	6.099110
2	2.604751
3	2.617684
4	0.306182
5	6.133467
6	2.455603
7	2.403266
8	0.928467

Exemple de données sauvegarder dans le fichier hiddenLayerBias

Il est possible de se dire qu'avec un très grand réseau de neurones la sauvegarde de données est volumineuse, mais en réalité un tel fichier texte reste peu volumineux car l'on y enregistre que des caractères. En suivant les instructions et en manipulant correctement les structures de données, l'IA peut maintenant "apprendre" à partir d'exemples.

La partie principale se fait le **passage avant (Forward pass)** qui calcule le résultat du réseau de neurone en fonction des inputs et le **rétropropagation (Backpropagation)** qui applique les changements à effectuer pour corriger les erreurs.

5.4 Réseau de neurones

Toutes les caractéristiques du réseau de neurones sont stockées dans un fichier init.txt, pour faciliter l'entraînement, mais aussi pour récupérer les informations au moment d'utiliser le réseau de neurones au moment de l'OCR.

```

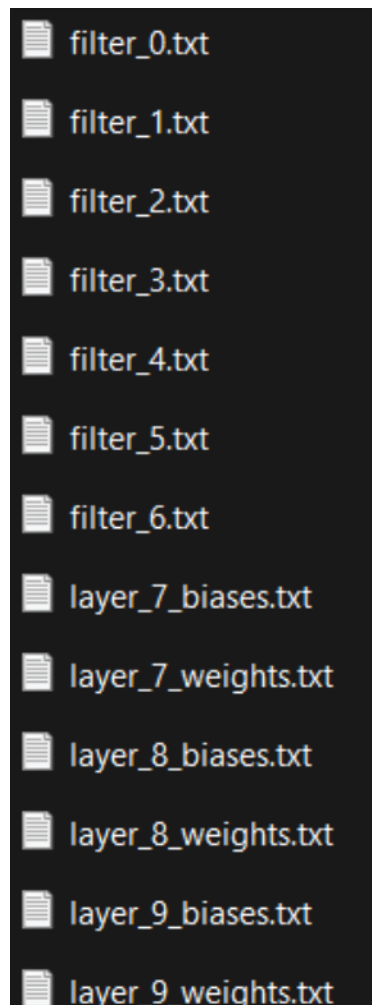
Nombre de layer fc:
3
Nombre de layer de convolution:
7
Nombre de pixels par layer convolution:
900 3136 784 1152 288 256 64
Nombre de neurones par layer:
64 32 26
Nombre de filtre par layer:
0 4 0 8 0 16 0
Learning rate:
0.001
Nombre d'époque:
1000000
Nombre de log par train:
2
Nombre de save:
15|

```

Pour l'entraînement, nous utilisons une database de 13000 images de lettres en png, soit 5000 images par lettre.

Expected Outputs: Q	Obtained Outputs: Q	Probability: 0.941065
Expected Outputs: T	Obtained Outputs: T	Probability: 0.960886
Expected Outputs: F	Obtained Outputs: F	Probability: 0.941410
Expected Outputs: Y	Obtained Outputs: Y	Probability: 0.966417
Expected Outputs: J	Obtained Outputs: J	Probability: 0.983401
Expected Outputs: Z	Obtained Outputs: Z	Probability: 0.985002
Expected Outputs: P	Obtained Outputs: P	Probability: 0.948936
Expected Outputs: A	Obtained Outputs: A	Probability: 0.920102
Expected Outputs: I	Obtained Outputs: I	Probability: 0.948123
Expected Outputs: V	Obtained Outputs: V	Probability: 0.931501
Expected Outputs: D	Obtained Outputs: D	Probability: 0.962131
Expected Outputs: B	Obtained Outputs: B	Probability: 0.783986
Expected Outputs: C	Obtained Outputs: C	Probability: 0.823672
Expected Outputs: S	Obtained Outputs: S	Probability: 0.962410
Expected Outputs: W	Obtained Outputs: W	Probability: 0.967433
Expected Outputs: M	Obtained Outputs: M	Probability: 0.915403
Expected Outputs: K	Obtained Outputs: K	Probability: 0.951955
Expected Outputs: G	Obtained Outputs: G	Probability: 0.960500
Expected Outputs: X	Obtained Outputs: X	Probability: 0.877543
Expected Outputs: U	Obtained Outputs: U	Probability: 0.968203
Expected Outputs: L	Obtained Outputs: L	Probability: 0.964808
Expected Outputs: E	Obtained Outputs: E	Probability: 0.923657
Expected Outputs: H	Obtained Outputs: H	Probability: 0.911624
Expected Outputs: R	Obtained Outputs: R	Probability: 0.964967
Expected Outputs: O	Obtained Outputs: O	Probability: 0.935111
Expected Outputs: N	Obtained Outputs: N	Probability: 0.892114

Les poids, les biais des neurones et les filtres sont sauvegardés dans des fichiers.



Après avoir découpé la grille en sous-images, celles-ci sont stockées dans un dossier. Le programme itère sur chaque image, utilise le réseau de neurones pour reconnaître les lettres, puis écrit les résultats dans un fichier texte. Ce fichier contient la grille complète, prête à être utilisée pour la résolution automatisée.

Problèmes rencontrés

Modification en type struct

L'une des premières difficultés a été la transition vers une implémentation basée sur des structures (structs). Bien que cette approche ait permis une meilleure organisation et une séparation claire des différents composants du réseau, elle a introduit une complexité

supplémentaire. Chaque élément devait être soigneusement encapsulé et lié aux autres parties du réseau, ce qui a nécessité une refonte de certaines parties du code initial et un effort supplémentaire pour garantir que toutes les parties fonctionnent ensemble de manière fluide.

Un réseau trop volumineux

Lors des premières tentatives, la taille du réseau de neurones conçu était trop importante, avec beaucoup trop de paramètres à entraîner. Cela impliquait que l'entraînement aurait nécessité une semaine complète, même avec des ressources raisonnables. Ce problème a mis en lumière l'importance de concevoir des architectures adaptées à la taille des données et aux capacités matérielles disponibles. Cela nous a également obligés à repenser et optimiser l'architecture pour rendre l'entraînement réalisable dans un délai raisonnable.

Changement d'architecture : Fc vers convolution

Au début, le réseau était basé sur des couches entièrement connectées (fully connected, comme pour une architecture de type nXOR). Cependant, ce modèle s'est avéré inadapte pour les données d'entrée structurées comme des images, où les relations spatiales sont importantes. Passer à une architecture de convolution a été un changement crucial, mais cela a introduit de nouveaux défis, notamment la compréhension et la mise en œuvre des concepts de convolution, de filtres, et de pooling. Ce changement a nécessité de réapprendre une partie du fonctionnement des réseaux de neurones et de repenser complètement la structure initiale.

Complexité d'implémentation d'un CNN

La mise en place d'un réseau de neurones de convolution s'est avérée bien plus complexe qu'une architecture classique. Il a fallu gérer plusieurs aspects simultanément : la conception des couches de convolution, la configuration des filtres, le réglage des paramètres de pooling, et enfin l'intégration avec des couches entièrement connectées. De plus, l'aspect algorithmique de la convolution, combiné à l'optimisation des calculs pour réduire la complexité temporelle, a nécessité des efforts importants pour obtenir un code fonctionnel et efficace.

5.5 Solver

La fonction principale de recherche parcourt chaque cellule de la grille. Pour chaque cellule correspondant à la première lettre du mot, le programme explore les huit directions pour vérifier si le mot complet est présent. La fonction `searchWord` utilise une boucle pour parcourir le mot dans la direction spécifiée. Si une lettre ne correspond pas ou si la position sort des limites de la grille, la recherche dans cette direction est abandonnée. Si le mot est trouvé dans une direction, les coordonnées de la dernière lettre sont stockées et retournées. Un tableau de directions est utilisé pour simplifier la gestion des déplacements dans la grille. Chaque direction est représentée par une paire de valeurs (dx, dy) indiquant le déplacement en abscisse et en ordonnée.

```
int directions[8][2] = {  
    {0, 1},  
    {0, -1},  
    {1, 0},  
    {-1, 0},  
    {1, 1},  
    {1, -1},  
    {-1, 1},  
    {-1, -1}  
};
```

Fonctionnement des directions

Chaque direction est testée depuis une cellule jusqu'à ce que le mot soit trouvé ou que toutes les directions aient été explorées.

Enfin, la fonction `isValid` vérifie que les coordonnées (x, y) restent dans les limites de la grille, évitant ainsi les débordements de mémoire ou les accès invalides. Cette fonction est utilisée dans la boucle de recherche pour chaque direction.

La distance de Levenshtein est une distance, au sens mathématique du terme, donnant une mesure de la différence entre deux chaînes de caractères. Elle est égale au nombre minimal d'opérations nécessaires pour transformer une chaîne de caractères en une autre, à l'aide des trois opérations autorisées : remplacement, suppression et ajout. Le programme lit la grille

depuis le fichier spécifié en ligne de commande, et chaque ligne du fichier est lue dans un tableau de caractères à deux dimensions (une matrice). Chaque ligne est lue et stockée en respectant les dimensions maximales de la grille définies par les constantes ROWS et COLS.

Il existe plusieurs axes d'amélioration pour optimiser encore davantage le système. À l'avenir, il serait pertinent de travailler sur l'optimisation des performances des solvers, notamment pour réduire les temps de calcul lorsqu'il s'agit de traiter des grilles de grande taille ou des motifs particulièrement complexes. Une autre piste intéressante serait d'enrichir l'interface utilisateur avec de nouvelles options, comme un mode collaboratif permettant à plusieurs utilisateurs de résoudre une grille simultanément ou un outil d'analyse statistique pour examiner les caractéristiques des mots trouvés. L'utilisation de cet algorithme permet une solution au cas où le réseau de neurones renvoie la mauvaise lettre.

Contraintes et limites

Le programme solver est conçu pour fonctionner avec des grilles de caractères de taille maximale définie par les constantes ROWS et COLS, pour le moment, fixées à 100x100. Les dimensions de la grille doivent respecter cette limite, et toute modification pour des grilles plus grandes nécessiterait des ajustements du code. La recherche de mots dans la grille n'est pas sensible à la casse, ce qui permet d'ignorer la distinction entre majuscules et minuscules lors de la correspondance des lettres. Le programme explore chaque cellule de la grille dans les huit directions (horizontal, vertical et diagonal), ce qui entraîne une complexité de recherche de l'ordre de $O(n * m * 8 * \text{len}(\text{word}))$, où n et m représentent les dimensions de la grille, et $\text{len}(\text{word})$ la longueur du mot à rechercher. Cette approche exhaustive assure une recherche complète mais peut ralentir le programme pour des grilles et des mots de grande taille, surtout si plusieurs occurrences du mot sont possibles.

6 CONCLUSION

Nous avons présenté la répartition des charges, l'état d'avancement du projet ainsi que les aspects techniques de celui-ci. Cette base nous permettra d'avancer efficacement dans le bon déroulement du projet. Pour rappel, nous avons élaboré des algorithmes capables de charger une image, supprimer les couleurs, faire une rotation manuelle et automatique de l'image, détecter la position de grille, mots et lettres, découper les images, résoudre une grille de mots cachés ainsi qu'un concept de réseau de neurones pour un NXOR, le prétraitement complet, un réseau de neurones complet et fonctionnel, la reconstruction de la grille, de la liste de mots et de la grille, l'affichage de la grille, la sauvegarde du résultat et une interface graphique. De plus, nous proposons des solutions en cas d'erreurs ou d'imprévus (distance de Levenshtein, barre de recherche,...) Les objectifs de ce projet vont au-delà de la réalisation du projet lui-même, que ce soit via le développement personnel et professionnel des membres du groupe mais également via l'apprentissage de compétences techniques et humaines qui nous aideront pour de futurs projets.