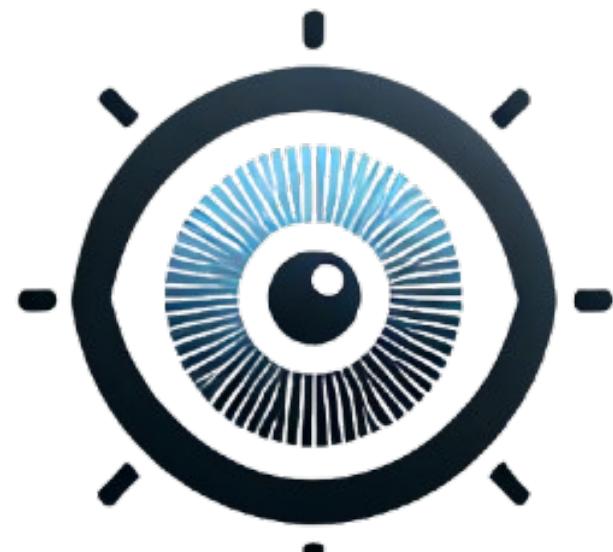


Projet OCR – Soutenance Finale

Groupe OCR-NOUPI

Décembre 2024



OCR-NOUPI

Table des matières

1	Introduction	5
2	Membres du groupe	5
2.1	DEFONTAINE JOLIVET Emilien	5
2.2	TOUSSAINT Ulysse	5
2.3	FORGET Clément	5
2.4	BIGOT Lucas	6
3	Répartition des tâches	7
4	Structure du projet	8
4.1	Fonctionnement de l'Arborescence	8
4.2	Séparation par Fonctionnalités	9
4.3	Avantages de cette Organisation	9
5	Prétraitements	10
5.1	Chargement et manipulations de l'image	10
5.2	Passage en niveau de gris (Grayscale)	10
5.3	Augmentation des contrastes	11
5.4	Flou Gaussien	11
5.5	Passage en noir et blanc (Binarisation)	11
5.6	Méthode d'Otsu	12
5.7	Méthode de Sauvola	12
5.8	Suppression des pixels volatiles	13
5.9	Résultats obtenus	14
6	Rotation automatique	16
6.1	Principe de rotation	16
6.2	Détection automatique de l'angle	16
7	Détections	17
7.1	Filtre de Canny	17
7.2	Détection de la grille	18
7.3	Détection des lettres de la grille	19
7.4	Détection des mots à trouver	20
7.5	Détection des lettres dans les mots	20
7.6	Résultats	21

8 Extractions	22
8.1 Extraction de la grille	22
8.2 Extraction des lettres de la grille	22
8.3 Extraction des mots à trouver	22
8.4 Extraction des lettres des mots	22
8.5 Résultats	23
9 Réseau de neurones	24
9.1 Fonctionnement général	24
9.2 Entraînement	26
9.3 Sauvegarde de l'entraînement	27
9.4 Jeu de données	29
10 Reconstructions	29
10.1 Sauvegarde des mots dans un fichier texte	30
10.2 Sauvegarde de la grille dans un fichier texte	30
10.3 Chargement de la grille depuis le fichier texte	30
10.4 Réajustement de la grille	31
11 Résolution de la grille	32
11.1 Le Solver	32
11.2 Recherche des mots dans la grille	32
11.3 Mise en évidence des mots dans la grille	34
12 Interface utilisateur	35
12.1 Architecture de l'interface	35
12.2 Principes d'ergonomie	35
12.3 Implémentation	37
13 Avancement final	38
14 Résultats finaux	39
15 Impressions de chacun	40
15.1 DEFONTAINE JOLIVET Emilien	40
15.2 TOUSSAINT Ulysse	40
15.3 FORGET Clément	40
15.4 BIGOT Lucas	41
16 Conclusion	41

Table des figures

1	Arbre simplifié de l'arborescence du projet	8
2	Résultats de la fonction <code>apply_small_groups_reduction</code>	14
3	Résultats obtenus après prétraitement	15
4	Avant/Après de la rotation automatique	17
5	Schéma modélisant le processus de détection	19
6	Illustration des différentes extractions	21
7	Illustration des étapes d'extraction : grille, lettres et mots.	23
8	Illustrations du dataset	29
9	Avant/Après réajustement de la grille	31
10	Recherche des mots et de leurs coordonnées	33
11	Maquette 1 : Menu latéral	36
12	Maquette 2 : Zone centrale	36
13	Maquette 3 : Barre d'action	36
14	UI au démarrage	37
15	UI avec menu ouvert	37
16	UI avec une image chargée	37
17	Résultats obtenus sur les grilles fournies (1/2)	39
18	Résultats obtenus sur les grilles fournies (2/2)	40

Liste des tableaux

1	Tableau de répartition des tâches	7
2	Tableau d'avancement final	38

1 Introduction

Ce document a pour objectif de présenter les résultats finaux obtenus par le groupe OCR-NOUPI dans le cadre du projet OCR du S3 à EPITA. Tout au long de ce rapport, nous allons revenir sur l'ensemble des étapes majeures de notre projet, les défis que nous avons surmontés, les algorithmes que nous avons développés et optimisés, ainsi que les résultats finaux obtenus. Nous conclurons également par une analyse de notre ressenti global et des enseignements tirés de ce projet.

2 Membres du groupe

2.1 DEFONTAINE JOLIVET Emilien

Aussi loin que je me souvienne, j'ai toujours été un "geek". Enfant, je passais énormément de temps à jouer aux jeux vidéo, aussi bien sur console que sur ordinateur. Cependant, ce n'est que pendant le confinement que je me suis lancé dans la programmation. J'ai d'abord appris à coder en Python, puis en JavaScript/TypeScript, car le développement web était à l'origine ce qui m'attirait le plus. Ensuite, à l'EPITA, j'ai appris le Caml, le C# et, en ce moment, j'apprends le C. De manière générale, j'adore tout ce qui est lié à la programmation, et je suis très heureux de participer à ce projet et d'être le "chef" de notre groupe.

2.2 TOUSSAINT Ulysse

J'ai choisi de me former au développement informatique car je souhaitais repousser les limitations rencontrées lors de l'utilisation d'applications. Initialement autodidacte, j'ai participé à divers projets pour renforcer mes compétences. Par la suite, j'ai consacré mon temps libre au télétravail avec plusieurs entreprises avant de rejoindre l'EPITA pour affiner mes connaissances.

2.3 FORGET Clément

Je suis passionné par l'informatique et toujours motivé pour découvrir de nouvelles technologies. Ce que j'aime le plus, c'est tout ce qui touche aux systèmes d'arrière-plan, comme le back-end dans le développement web, là où tout se passe sans qu'on ne le voie. J'adore le défi de créer des solutions solides et efficaces qui rendent les applications plus fluides et performantes. Toujours curieux, je cherche sans cesse à améliorer mes compétences et à plonger encore plus dans ce domaine en perpétuel mouvement.

2.4 BIGOT Lucas

Je m'appelle Lucas Bigot, j'ai toujours joué aux jeux vidéo étant plus jeune. Je ne comptais pas mes heures sur les jeux vidéo. Au fil du temps, j'ai commencé à m'intéresser de plus en plus à l'informatique et à la programmation. C'est pour cela que j'ai choisi d'intégrer l'EPITA, qui me permet d'approfondir mes notions en programmation. J'ai donc appris différents langages informatiques tels que Caml, Python et C#, et nous sommes en train d'apprendre le C. L'année passée, j'ai également été le chef de projet pour la création d'un jeu vidéo appelé Simucorp. Je suis donc très heureux de participer à ce nouveau projet, qui me permettra de développer de nouvelles compétences.

3 Répartition des tâches

Voici la répartition des tâches au sein de notre groupe :

Tâches	Membres			
	Emilien	Clément	Lucas	Ulysse
Gestion de l'image				
Prétraitement	Principal		Second	
Rotation de l'image	Second			Principal
Détections & extractions	Second		Principal	
Réseau de Neurones				
XNOR	Second	Principal		
Reconnaissance des lettres	Second	Principal		
Entraînement du réseau de neurones	Second		Principal	
Sauvegarde & chargement des poids	Second		Principal	
Données d'entraînement	Principal		Second	
Résolution de la grille				
Chargement et réajustements de la grille	Principal		Second	
Chargement des mots	Principal		Second	
Solver	Principal		Second	
Entourage des mots	Second		Principal	
Autre				
Designs / UI				Principal
Rapports	Principal			

TABLE 1 – Tableau de répartition des tâches

4 Structure du projet

L’arborescence du projet a été pensée pour garantir une organisation claire et fonctionnelle, avec une séparation rigoureuse des différentes parties du code en fonction des fonctionnalités. Cette approche modulaire facilite non seulement le développement, mais aussi la maintenance et l’évolutivité du programme.

4.1 Fonctionnement de l’Arborescence

Le cœur du projet se trouve dans le dossier `src/`, qui contient tout le code source. Ce dossier est subdivisé en sous-dossiers, chacun dédié à une fonctionnalité spécifique. Par exemple, les traitements liés aux images, le réseau neuronal, ou encore l’interface utilisateur disposent de leurs propres espaces.

Chaque module contient à la fois les fichiers `.c` (implémentation) et `.h` (déclarations). Cette organisation garantit que les fonctions, structures, et constantes définies dans un module sont accessibles aux autres, tout en limitant les interférences. En regroupant les fichiers `.c` et `.h` au même endroit, nous assurons une meilleure lisibilité et une gestion simplifiée des dépendances.

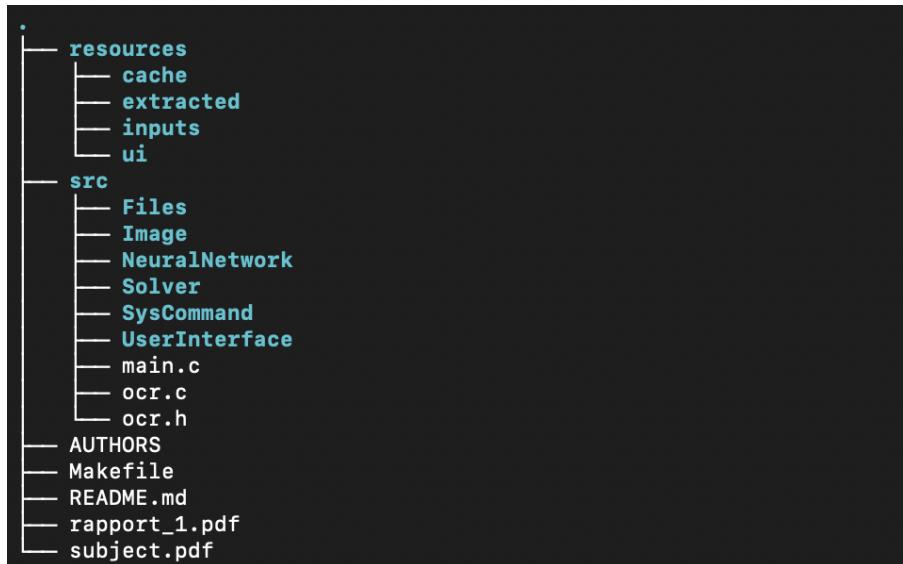


FIGURE 1 – Arbre simplifié de l’arborescence du projet

4.2 Séparation par Fonctionnalités

L'organisation en modules fonctionnels permet de compartimenter les responsabilités. Chaque sous-dossier a un rôle précis :

- **Isolation des responsabilités** : Par exemple, le dossier dédié à l'interface utilisateur (`UserInterface/`) contient tout ce qui concerne les widgets, menus et fenêtres, tandis que le traitement d'images se trouve exclusivement dans le dossier `Image/`.
- **Cohérence interne** : Les fonctions qui appartiennent à une même fonctionnalité sont regroupées, ce qui évite la dispersion du code. Par exemple, les opérations de recadrage, redimensionnement, et traitement des images sont toutes contenues dans `Image/`.
- **Facilité d'évolution** : Ajouter une nouvelle fonctionnalité ou modifier une existante peut se faire en créant ou modifiant un seul sous-dossier, sans impact sur le reste du projet.

4.3 Avantages de cette Organisation

- **Clarté** : En séparant les fichiers selon leur fonction, il est facile de localiser rapidement une partie spécifique du code. Par exemple, tout ce qui concerne le réseau neuronal est dans `NeuralNetwork/`, sans interférer avec les autres parties du projet.
- **Modularité** : Chaque module peut être testé et développé indépendamment. Cela a permis à l'équipe de travailler en parallèle sur différentes parties du projet sans dépendre les uns des autres.
- **Compilation simplifiée** : Grâce à l'utilisation du `Makefile`, la structure modulaire des fichiers rend la gestion des dépendances automatique. Le `Makefile` compile uniquement les fichiers nécessaires en fonction des changements effectués.
- **Évolutivité** : Si de nouvelles fonctionnalités doivent être ajoutées, elles peuvent être intégrées sous forme de nouveaux modules dans des sous-dossiers dédiés, sans alourdir les parties existantes.

Cette organisation structurée du code a permis de maintenir une cohérence tout au long du projet, de minimiser les conflits lors du travail en équipe, et de rendre le projet facilement compréhensible pour tout nouveau contributeur.

5 Prétraitements

5.1 Chargement et manipulations de l'image

Pour débuter notre projet, nous avons développé un ensemble de fonctions destinées à manipuler des images en utilisant la bibliothèque `SDL2`, qui constitue le cœur technique de notre système. Nous avons conçu une structure principale, `iImage`, qui permet de représenter une image sous forme de matrice de pixels (`pPixel`), tout en intégrant des informations essentielles telles que ses dimensions, son chemin d'accès et, éventuellement, une étiquette (`label`) utilisée pour des tâches supervisées comme l'entraînement de notre modèle OCR.

À partir de cette structure, nous avons élaboré des fonctions pour charger des images depuis un fichier, extraire leurs pixels, les redimensionner ou les recadrer. Par exemple, la fonction `load_image` permet de charger une image tout en lui associant une étiquette, tandis que `resize_image` et `resize_image_with_white` permettent de modifier ses dimensions en respectant ou non son ratio d'aspect, selon les besoins. Nous avons également implémenté des outils pour sauvegarder les images après traitement, créer des sous-images à partir de zones spécifiques, ou encore recadrer les images pour ne conserver que les régions d'intérêt.

Ces fonctionnalités de manipulation d'images ont été fondamentales pour préparer les données à traiter dans le cadre des tâches de reconnaissance de caractères. Elles nous ont permis d'établir une base robuste pour les étapes suivantes de notre projet, notamment en garantissant une gestion efficace des images et une intégration facile avec les algorithmes de traitement et d'apprentissage.

5.2 Passage en niveau de gris (Grayscale)

Dans un premier temps, il a fallu transformer notre image en niveaux de gris. Pour ce faire, il faut convertir chaque pixel afin qu'il soit représenté par une seule intensité lumineuse, qui varie de 0 (noir) à 255 (blanc). Cette nouvelle intensité est calculée en combinant les canaux (R, G et B) de chaque pixel en utilisant une formule spécifique. La formule standard pour réaliser cette conversion s'exprime ainsi :

$$\text{Gray} = 0.3 \times R + 0.59 \times G + 0.11 \times B$$

où R, G et B représentent les valeurs des composantes "Red, Green, Blue" du pixel. En appliquant cette formule à chaque pixel de l'image, nous obtenons une image en niveaux de gris, où chaque pixel a une intensité unique comprise entre 0 et 255.

5.3 Augmentation des contrastes

Une fois le passage en niveaux de gris terminé, il faut augmenter les contrastes de l'image modifiée afin de rendre les détails encore plus visibles. Pour réaliser cette tâche, plusieurs méthodes s'offraient à nous. Nous avons choisi une implémentation qui repose sur "l'étrage" de l'histogramme des intensités de l'image. Nous utilisons une valeur de référence qui se base sur la luminosité moyenne de l'image, couplée à un seuil de sensibilité. Ainsi, si la différence d'intensité entre le pixel parcouru et cette valeur est suffisamment importante, nous exagérons cette différence à l'aide d'un facteur multiplicatif α afin de rendre les différences d'intensité encore plus marquées. La valeur de la nouvelle intensité est donc définie par :

$$\text{nouvelle_intensite} = \alpha \times (\text{intensite} - \text{intensite_moyenne}) + \text{intensite_moyenne}$$

5.4 Flou Gaussien

Le flou gaussien est une technique de traitement d'image utilisée pour adoucir/lisser une image grâce à la réduction des variations de contraste. Cette méthode fonctionne en appliquant un filtre qui se base sur la distribution gaussienne pour chaque pixel de l'image. La fonction gaussienne est la suivante :

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

où x et y sont les coordonnées spatiales, et σ représente l'écart-type de la distribution. Plus σ est grand, plus le flou est important.

Le flou gaussien consiste à appliquer le noyau gaussien sur chaque pixel de l'image. Ensuite, il faut calculer une moyenne pondérée des pixels voisins en fonction de leur distance. Plus un pixel est proche du centre du noyau, plus son importance est élevée dans le résultat final de l'intensité du pixel. Finalement, ce processus est répété sur tous les pixels qui constituent l'image. Cela permet d'obtenir un résultat de "flou" sur l'image.

5.5 Passage en noir et blanc (Binarisation)

La binarisation de l'image consiste à transformer une image déjà en niveaux de gris en une image binaire, constituée uniquement de pixels blancs ou noirs, en utilisant un seuil défini arbitrairement. Chaque pixel de l'image est comparé à ce seuil et si la valeur du pixel parcouru est supérieure ou égale au seuil, alors il est transformé en blanc (255) et sinon, il devient noir (0).

5.6 Méthode d’Otsu

La méthode d’Otsu permet de convertir une image en niveaux de gris en une image binaire. Elle permet de choisir automatiquement un seuil pour séparer les pixels de l’image en deux groupes : ceux qui appartiennent au fond et ceux qui appartiennent à l’objet, en fonction de leurs intensités.

Notre implémentation se fait en deux étapes. Dans un premier temps, nous avons implémenté la fonction `compute_otsu_threshold` qui calcule le meilleur seuil de gris. Elle parcourt tous les niveaux de gris possibles (de 0 à 255) pour trouver celui qui sépare le mieux les pixels en deux groupes distincts. Ce seuil est choisi pour que la différence entre les deux groupes soit la plus grande possible. Concrètement, cette fonction calcule le niveau de gris qui maximise la "distance" entre les pixels du fond et ceux de l’objet (la lettre dans notre cas). Ensuite, la fonction `otsu_threshold` applique cette méthode à des petits blocs de l’image, de taille passée en paramètre.

5.7 Méthode de Sauvola

La méthode de Sauvola est une technique locale d’adaptation de seuil utilisée pour binariser une image en niveaux de gris. Contrairement aux approches globales, cette méthode calcule un seuil propre à chaque région de l’image en s’appuyant sur les statistiques locales des pixels. Voici les étapes principales de son fonctionnement :

- 1. Division de l’image en blocs** : L’image est divisée en blocs de taille spécifiée (`block_size`). Chaque bloc est traité indépendamment pour calculer un seuil adapté.
- 2. Calcul des statistiques locales** : Pour chaque bloc, les valeurs suivantes sont calculées :

- La *somme des intensités* des pixels (`sum`), permettant de déterminer la moyenne (`mean`) :

$$\text{mean} = \frac{\text{sum}}{\text{count}}$$

- La *somme des carrés des intensités* (`sum_sq`), utilisée pour calculer la variance (`variance`) :

$$\text{variance} = \frac{\text{sum_sq} - \frac{\text{sum}^2}{\text{count}}}{\text{count}}$$

- L’écart-type (`std_dev`) est ensuite donné par :

$$\text{std_dev} = \sqrt{\text{variance}}$$

- 3. Calcul du seuil local** : Le seuil est calculé pour chaque bloc à l’aide de la formule

suivante :

$$\text{threshold} = \text{mean} \times \left(1 + k \times \left(\frac{\text{std_dev}}{R} - 1 \right) \right)$$

où :

- k est un paramètre ajustable (par défaut $k = 0.5$) qui contrôle la sensibilité à l'écart-type,
- R est une valeur constante représentant l'écart-type maximal attendu (typiquement $R = 128$).

4. **Binarisation des pixels du bloc** : Chaque pixel du bloc est comparé au seuil local calculé :

- Si l'intensité du pixel est supérieure au seuil, il est assigné à l'objet (255, 255, 255 pour blanc).
- Sinon, il est assigné au fond (0, 0, 0 pour noir).

5.8 Suppression des pixels volatiles

Lors du traitement de certaines images, nous avons remarqué la présence de pixels superflus, sous forme d'amas de différentes tailles. Ces amas peuvent être composés de très petites zones de pixels noirs isolés, ou au contraire, de zones plus larges qui se forment notamment lorsque des grilles ou autres formes sont présents sur l'image. Ces irrégularités peuvent perturber les algorithmes de détection, notamment en faussant l'identification des contours (détection de grille par exemple) mais aussi au niveau de l'extraction.

Afin de remédier à ce problème, nous avons décidé d'implémenter une fonction de prétraitement `apply_small_groups_reduction` capable d'éliminer ces amas de pixels indésirables. Cette fonction repose sur une analyse systématique des amas de pixels noirs présents dans l'image. Voici les grandes étapes de son fonctionnement :

Chaque pixel de l'image est analysé. Lorsqu'un pixel noir est rencontré, l'objectif est d'identifier l'amas de pixels auquel il appartient. Pour ce faire, une exploration des pixels voisins est réalisée afin de déterminer l'étendue complète de cet amas. On identifie donc les connexions entre pixels noirs.

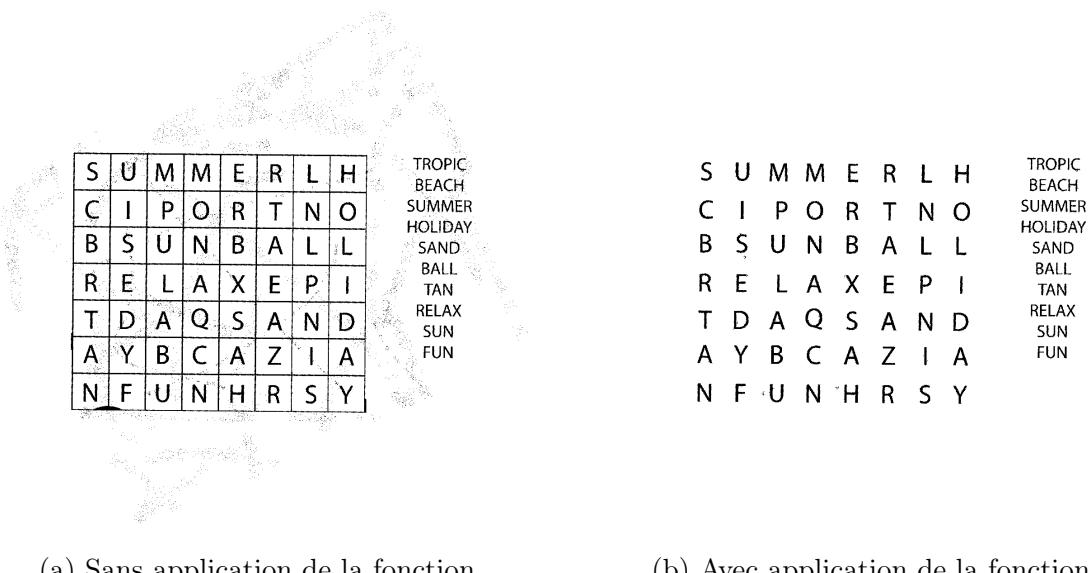
Une fois l'amas identifié, nous calculons sa taille (c'est-à-dire le nombre total de pixels qu'il contient). La taille de l'amas est ensuite comparée à des seuils prédéfinis :

- Si la taille de l'amas est inférieure à un seuil minimal, il est considéré comme un bruit parasite.
- Si la taille dépasse un certain seuil maximal, il peut s'agir d'un amas non pertinent, comme une partie de la grille, qui nuit à l'analyse de l'image.

Si l'amas ne satisfait pas les critères définis (par exemple, s'il est trop petit ou trop grand), tous les pixels de cet amas sont remplacés par des pixels blancs. Cela permet de nettoyer l'image en supprimant les zones non pertinentes.

Pour éviter de traiter plusieurs fois les mêmes pixels, un tableau est utilisé afin de marquer les pixels déjà visités. Ainsi, une fois qu'un amas a été analysé, tous ses pixels sont enregistrés comme “traités”, ce qui permet d'éviter des analyses redondantes et d'accélérer le processus.

Cette fonction s'avère donc très utile pour la réduction de bruit d'une image, mais aussi pour les images contenant beaucoup d'amas de pixels superflu. En résumé, cette fonction de prétraitement permet de s'assurer que nos images soient net, ce qui facilite toutes les taches de détection, ainsi que d'extraction qui vont venir par la suite



(a) Sans application de la fonction

(b) Avec application de la fonction

FIGURE 2 – Résultats de la fonction `apply_small_groups_reduction`

5.9 Résultats obtenus

Bien que nous vous ayons présentés plusieurs méthodes/étapes utilisables lors du prétraitement d'une image. Après les avoirs toutes essayés avec différents paramètres et dans différents ordres, nous n'avons retenus que 3 d'entre elles qui sont : la méthode de grayscale, la méthode de Sauvola et notre fonction de suppression des pixels volatiles. Vous remarquerez que l'image 2_1 n'est pas prétraité correctement, nous en sommes conscients et avons préférés faire l'impasse sur celle-ci qui était trop contraignante par rapport aux autres. En effet, des méthodes/odres de prétraitements fonctionnels sur celle-ci ne l'étaient pas sur d'autres images.

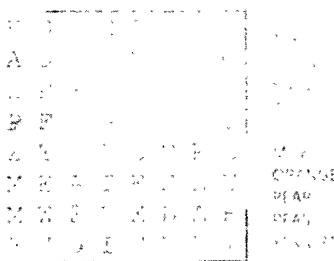
M	S	W	A	T	E	R	M	E	L	O	N
Y	T	B	N	E	P	E	W	R	M	A	E
R	R	L	W	P	A	P	A	Y	A	N	A
R	A	N	L	E	M	O	N	A	N	E	P
E	W	L	E	A	P	R	I	A	B	P	R
B	B	I	L	B	B	W	B	R	L	A	Y
K	E	M	P	M	A	W	L	R	A	R	B
C	R	E	P	R	N	R	E	R	R	G	R
A	R	Y	A	Y	A	O	A	N	L	A	M
L	Y	Y	A	R	N	E	R	K	I	W	I
B	E	B	A	A	A	N	A	A	P	R	T
Y	R	R	E	B	P	S	A	R	N	N	W
Y	R	R	E	B	E	U	L	B	L	G	I
T	Y	P	A	T	E	A	E	P	A	C	E

APPLE LEMON
BANANA
LIME ORANGE
WATERMELON GRAPE
STRAWBERRY
PAPAYA
BLUEBERRY BLACKBERRY
RASPBERRY

(a) Image 1.1

MINDFULNESS	P X U T S I N I U P R V G B M D D
IMAGINE	E H A A S P O J P E T B E Q Z L C
RELAX	A U N T E G Q T L H R Z F A T O P
COOL	S H X F N G U A X E A A Y P O M H
RESTING	Y O Y Y L D X L A K Y U Z L B S K
BREATHE	J X M U G Q T R I M A G I N E B
EASY	H F N W F X H D P B B B T N V S K
TENSION	H I I H D E S Q F U M Y E R N S X
STRESS	R P B Z H S D S L H O N B S S S
CALM	E H X A I Z I H A H O E S Q F E F
	C W Z I M V D C J Y S S I M G R W
	L A I I R Z Q Q H X D Z O Z Q T R
	W C A X E Z R G H A I Z N E C S E
	B R H F O T G N I T S E R E O V Z
	M W V W Q D U I H W Q T S B I M L
	T D T O N Z C X X R G E L K H F Q
	Q N E K S Y M O T F A L A A E W B

(b) Image 1.2



(c) Image 2.1

S U M M E R
 C I P O R T R L H
 B S U N B A L L
 R E L A X E P I
 T D A Q S A N D
 A Y B C A Z I A
 N F U N H R S Y
 TROPIC BEACH
 SUMMER HOLIDAY
 SAND BALL
 TAN
 RELAX SUN
 FUN

(d) Image 2.2

NAME _____ DATE _____

Strange Words

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	TERMAGAN
DISCOMBOBULATED	PANGLOSSIAN	SUSURRUS
OMPHALOSKEPSIS	ERYTHRISMAL	ESTIVATE
PROPIOCEPTION	PALINDROME	SPANGHEW
FATTERDEMALION	ENERVATING	FRIPPET
PUSILLANIMOUS	PALIMPSEST	SYZYGY
CRYPTOMNFSIA	SPFLUNKING	TMFSIS

(e) Image 3.1



(f) Image 4.1



(g) Image 3.2



(h) Image 4.2

FIGURE 3 – Résultats obtenus après prétraitement

6 Rotation automatique

6.1 Principe de rotation

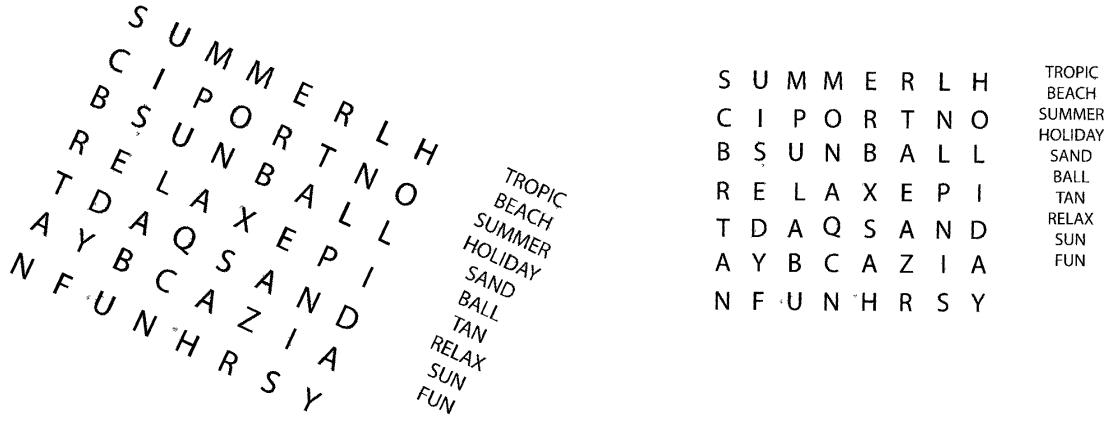
Pour corriger l'orientation de l'image, nous utilisons une matrice de rotation qui effectue une rotation inverse sur chaque pixel en fonction de l'angle fourni. La matrice de rotation pour un angle θ est donnée par :

$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Chaque pixel de l'image est repositionné en multipliant ses coordonnées originales par cette matrice de rotation. Cette transformation permet de calculer les nouvelles positions des pixels après rotation. Pour redresser l'image, nous appliquons cette matrice avec un angle opposé à l'inclinaison initiale, réalignant ainsi l'image. Lors de la rotation de l'image, des pixels qui n'étaient pas présents auparavant peuvent être créés car des coordonnées obtenues peuvent ne pas être des valeurs entières de la matrice représentant l'image. Dans ce cas-là, nous avons décidé de les rendre noirs.

6.2 Détection automatique de l'angle

Dans le cadre de notre projet, la détermination automatique de l'angle de rotation requis pour redresser une image inclinée s'appuie sur l'analyse précise de ses contours. Le code présenté met en œuvre cette approche en générant, dans un premier temps, deux matrices destinées à représenter l'intensité des bords et leurs orientations locales. Grâce à la fonction `detect_edges`, chaque pixel de l'image est ainsi associé à une direction et une intensité de gradient, permettant de cartographier la structure géométrique sous-jacente. Cette information sert de base à la fonction `find_dominant_angle`, qui identifie la direction privilégiée par les lignes dominantes présentes dans l'image. L'angle obtenu est ensuite ajusté afin de se conformer à des conventions plus cohérentes (par exemple, en réinterprétant un angle supérieur à 90°). Enfin, une légère correction supplémentaire est appliquée, de sorte à produire un angle de rotation final adapté aux exigences du redressement. Si l'angle résultant demeure inférieur à un seuil défini, le système considère que l'image est suffisamment droite pour ne pas nécessiter d'intervention. Cette méthode, fondée sur l'analyse géométrique des contours, assure ainsi un redressement fiable et autonome, facilitant grandement les traitements ultérieurs, tels que l'extraction de texte ou l'analyse structurée.



(a) Avant rotation automatique

(b) Après rotation automatique

FIGURE 4 – Avant/Après de la rotation automatique

7 Détections

Le filtre de Canny (ou détecteur de Canny) est utilisé en traitement d’images pour la détection des contours. L’algorithme a été conçu par John Canny en 1986 pour être optimal suivant trois critères clairement explicités : bonne détection : faible taux d’erreur dans la signalisation des contours, bonne localisation : minimisation des distances entre les contours détectés et les contours réels, clarté de la réponse : une seule réponse par contour et pas de faux positifs. (Source : Wikipédia)

7.1 Filtre de Canny

L’algorithme commence par charger une image via une structure de données `iImage`. Une fois l’image chargée, la première étape est le calcul des gradients. Pour chaque pixel, les gradients horizontaux (G_x) et verticaux (G_y) sont calculés en utilisant des filtres de Sobel. Les valeurs obtenues permettent de calculer la magnitude et la direction des gradients pour chaque pixel, qui sont stockées dans des matrices dédiées. Ces gradients permettent d’évaluer l’intensité des contours dans l’image.

La deuxième étape est la **suppression des non-maxima**. Pour chaque pixel, l’algorithme examine ses voisins dans la direction du gradient et conserve uniquement les pixels qui représentent un maximum local. Cela permet de ne garder que les contours les plus nets, en supprimant les pixels qui ne contribuent pas aux contours principaux.

Ensuite, nous appliquons un **seuil d’hystérosis**, dans lequel nous définissons deux

seuils, `low_thresh` et `high_thresh`, pour distinguer les pixels fortement associés aux contours (haute intensité) de ceux qui sont moins importants. Les pixels au-dessus du seuil haut sont marqués comme des contours sûrs, tandis que ceux entre les deux seuils ne sont considérés comme contours que s'ils sont connectés à des contours sûrs.

Pour renforcer la visibilité des contours détectés, nous appliquons une **dilatation** qui élargit les contours en ajoutant des pixels autour de ceux marqués comme contours. Cela rend les contours plus visibles et permet de capturer des formes plus complètes.

Une fois les contours bien définis, nous appliquons une détection de boîtes englobantes (**bounding boxes**) pour regrouper les pixels connectés formant des contours et identifier les zones d'intérêt. Ces boîtes sont stockées avec leurs coordonnées et dimensions, permettant ensuite un traitement personnalisé via une fonction externe fournie en paramètre. Enfin, toutes les ressources sont libérées pour éviter les fuites de mémoire.

7.2 Détection de la grille

Afin de créer notre algorithme de détection de grille, nous avons décidé d'utiliser l'algorithme de Canny que nous avons décrit à la page précédente. Cet algorithme nous permet d'obtenir les boîtes englobantes de chaque amas de pixels. Toutes les lettres de la grille, mots et imperfections sont donc contenues dans des boxes, et nous devons traiter chacune de ces boxes afin d'en extraire une grille. En calculant la médiane de la hauteur des boîtes, on peut déterminer la hauteur des lettres de la grille. En effet, si l'on regarde notre liste de boîtes, ces lettres sont majoritaires. Nous pouvons ainsi vérifier que la hauteur de chaque box est à peu près équivalente à la médiane des hauteurs des boxes. Nous utilisons ici la fonction (`compute_median2`, cette fonction prend un argument, notre liste de boxes, un paramètre `mod` qui nous permet de préciser si l'on veut calculer la médiane de la hauteur, de la largeur ou de la surface. La fonction retourne la médiane des valeurs des boxes dans le mode choisi).

Cependant, ces conditions ne sont pas suffisantes. En effet, un mot peut avoir la même hauteur qu'une lettre de la grille. Nous devons donc vérifier que la largeur de notre box soit inférieure à deux fois la hauteur (la lettre "W" est l'une des plus larges lettres de l'alphabet. Nous avons remarqué que cette lettre satisfait bien cette condition). Cela permet d'éliminer tous les mots dont la largeur est supérieure à deux fois la hauteur. Un autre problème subsiste : certains pixels ou amas de pixels provenant d'un dessin, par exemple, peuvent échapper à notre traitement. Pour y remédier, pour chacune des boxes visitées, nous vérifions qu'il y a une autre box à droite ou à gauche et une autre box au-dessus ou en dessous. Nous vérifions donc que le pattern des boxes est semblable à celui d'une

grille de lettres.

Nous pouvons également ajouter certaines conditions, par exemple sur les tailles minimales ou maximales des boxes (surface, hauteur, largeur).

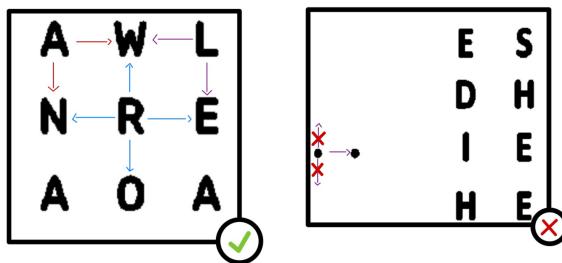


FIGURE 5 – Schéma modélisant le processus de détection

Nous pouvons donc déterminer avec une grande probabilité si une box est une lettre ou non. Afin de récupérer la grille, pour chaque lettre de notre liste de boxes, nous regardons ses coordonnées minimales et maximales sur les axes x et y. Nous cherchons les coordonnées maximales et minimales de la grille en comparant les coordonnées de chaque box (lettre) avec celles de la grille et en ajustant les coordonnées de la grille si nous rencontrons un nouveau minimum ou maximum.

7.3 Détection des lettres de la grille

Afin de détecter les lettres présentes dans la grille, nous utilisons les coordonnées de la grille pour analyser chaque boîte englobante incluse dans celle-ci. Cette approche permet d'examiner chaque région délimitée par les intersections de la grille pour déterminer si elle contient une lettre valide ou non.

Grâce à un traitement préalable, les coordonnées de la grille sont extraites. Cela permet de délimiter chaque boîte englobante, c'est-à-dire chaque cellule de la grille.

Chaque boîte est vérifiée individuellement. Cette vérification consiste à analyser le contenu de la boîte afin d'évaluer si elle contient une lettre ou simplement des groupes de pixels parasites, comme des pixels résiduels ayant échappé au prétraitement.

On obtient donc une liste de boîtes englobantes contenant chacune des lettres de notre grille.

7.4 Détection des mots à trouver

Afin de trouver les mots, nous procémons de façon inverse à la détection des lettres dans la grille. Tout d'abord, nous vérifions pour chacune des boîtes englobantes qu'elle ne soit pas incluse dans la grille. Pour cela, nous comparons les coordonnées de notre mot potentiel avec les coordonnées précédemment extraites de la grille.

Nous vérifions également certains paramètres, tels que les dimensions de la boîte, pour éliminer les amas superflus. Ce qui nous permet finalement d'obtenir une liste de boîtes englobantes contenant les mots de que nous allons devoir trouver dans la grille par la suite.

7.5 Détection des lettres dans les mots

Notre processus de détection de lettres a nécessité certains ajustements, car la méthode initiale présentait certaines limites pour certaines grilles. Au départ, nous avons tenté de récupérer les images des mots détectés dans la liste de mots, puis d'appliquer à nouveau le filtre de Canny pour détecter les contours de ces lettres. Cependant, sur certaines images, les lettres n'étaient pas détectées en raison d'une trop faible densité de pixels, ce qui empêchait le filtre de Canny d'identifier correctement les contours des lettres des mots dans les images extraites.

Pour résoudre ce problème, nous avons décidé d'implémenter une méthode inspirée de la suppression des pixels volatils. Cette méthode se base sur l'analyse des amas de pixels noirs pour déterminer leurs coordonnées maximales et minimales, afin de créer une boîte englobante pour chacun de ces amas.

Pour résoudre ce problème, nous avons mis en place un algorithme qui parcourt l'image pixel par pixel. Lorsqu'un pixel noir est détecté, nous identifions les coordonnées maximales et minimales de l'amas de pixels auquel il appartient. Cette opération est réalisée grâce à une fonction secondaire qui vérifie si le pixel noir n'a pas encore été traité. Si ce n'est pas le cas, la fonction renvoie les coordonnées de l'ensemble des pixels appartenant à cet amas vers la fonction principale. À partir de ces coordonnées, une boîte englobante est créée pour chaque amas détecté.

Chaque boîte englobante, qui correspond à une lettre, est ensuite ajoutée dans une liste associée au mot en cours de traitement. Ainsi, pour chaque mot, nous obtenons une liste de boîtes englobantes où chaque boîte correspond à une lettre spécifique.

L'utilisation d'un tableau de pixels déjà visités a permis d'améliorer significativement l'efficacité du traitement, puisque chaque pixel est traité une seule fois. De plus, cette

méthode améliore la détection des lettres présentant des pixels manquants. En effet, lorsqu'une lettre est partiellement interrompue par des pixels blancs, l'utilisation du tableau des pixels visités permet de regrouper correctement les différents amas de pixels appartenant à une même lettre. Sans ce tableau, deux amas distincts seraient identifiés comme deux lettres différentes, alors qu'ils appartiennent en réalité à la même lettre.

7.6 Résultats

Pour illustrer notre propos, voici les résultats de nos différents processus de détection sur l'image 1.1 :

M	S	W	A	T	E	R	M	E	L	O	N
Y	T	B	N	E	P	E	W	R	M	A	E
R	R	L	W	P	A	P	A	P	A	Y	A
R	A	N	L	E	M	O	N	A	N	E	P
E	W	L	E	A	P	R	I	A	B	P	R
B	B	I	L	B	B	W	B	R	L	A	Y
K	E	M	P	M	A	W	L	R	A	R	B
C	R	E	P	R	N	R	E	R	R	G	R
A	R	Y	A	Y	A	O	A	N	L	A	M
L	Y	Y	A	R	N	E	R	K	I	W	I
B	E	B	A	A	N	A	A	P	R	T	
Y	R	R	E	B	P	S	A	R	N	N	W
Y	R	R	E	B	E	U	L	B	L	G	I
T	Y	P	A	T	E	A	E	P	A	C	E
Y	T	B	N	E	P	E	W	R	M	A	E
R	R	L	W	P	A	P	A	P	A	Y	A
R	A	N	L	E	M	O	N	A	N	E	P
E	W	L	E	A	P	R	I	A	B	P	R
B	B	I	L	B	B	W	B	R	L	A	Y
K	E	M	P	M	A	W	L	R	A	R	B
C	R	E	P	R	N	R	E	R	R	G	R
A	R	Y	A	Y	A	O	A	N	L	A	M
L	Y	Y	A	R	N	E	R	K	I	W	I
B	E	B	A	A	N	A	A	P	R	T	
Y	R	R	E	B	P	S	A	R	N	N	W
Y	R	R	E	B	E	U	L	B	L	G	I
T	Y	P	A	T	E	A	E	P	A	C	E

(a) Grille détectée

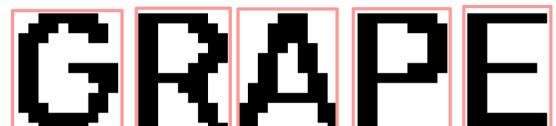
M	S	W	A	T	E	R	M	E	L	O	N
Y	T	B	N	E	P	E	W	R	M	A	E
R	R	L	W	P	A	P	A	P	A	Y	A
R	A	N	L	E	M	O	N	A	N	E	P
E	W	L	E	A	P	R	I	A	B	P	R
B	B	I	L	B	B	W	B	R	L	A	Y
K	E	M	P	M	A	W	L	R	A	R	B
C	R	E	P	R	N	R	E	R	R	G	R
A	R	Y	A	Y	A	O	A	N	L	A	M
L	Y	Y	A	R	N	E	R	K	I	W	I
B	E	B	A	A	N	A	A	P	R	T	
Y	R	R	E	B	P	S	A	R	N	N	W
Y	R	R	E	B	E	U	L	B	L	G	I
T	Y	P	A	T	E	A	E	P	A	C	E

(b) Détection des lettres de la grille

M	S	W	A	T	E	R	M	E	L	O	N
Y	T	B	N	E	P	E	W	R	M	A	E
R	R	L	W	P	A	P	A	P	A	Y	A
R	A	N	L	E	M	O	N	A	N	E	P
E	W	L	E	A	P	R	I	A	B	P	R
B	B	I	L	B	B	W	B	R	L	A	Y
K	E	M	P	M	A	W	L	R	A	R	B
C	R	E	P	R	N	R	E	R	R	G	R
A	R	Y	A	Y	A	O	A	N	L	A	M
L	Y	Y	A	R	N	E	R	K	I	W	I
B	E	B	A	A	N	A	A	P	R	T	
Y	R	R	E	B	P	S	A	R	N	N	W
Y	R	R	E	B	E	U	L	B	L	G	I
T	Y	P	A	T	E	A	E	P	A	C	E

(c) Détection des mots

APPLE
LEMON
BANANA
LIME
ORANGE
WATERMELON
GRAPE
KIWI
STRAWBERRY
PAPAYA
BLUEBERRY
BLACKBERRY
RASPBERRY



(d) Détection des lettres du mot

FIGURE 6 – Illustration des différentes extractions

8 Extractions

8.1 Extraction de la grille

L'extraction de la grille commence par un parcours complet de l'image, pixel par pixel. L'algorithme recherche les zones tracées en rouge ($R=255, G=0, B=0$), qui délimitent la grille. Dès qu'un pixel rouge est détecté, l'algorithme mesure la largeur et la hauteur de cette zone en suivant les pixels adjacents de même couleur. Une fois la taille déterminée, il s'assure que les bords droit et bas de cette zone sont bien constitués de pixels rouges pour confirmer qu'il s'agit d'une boîte fermée. Une fois validée, la zone intérieure est extraite en copiant uniquement les pixels internes à l'exclusion des bordures, et une nouvelle image est créée pour cette zone. Le contenu extrait est alors sauvegardé dans un fichier nommé spécifiquement pour la grille, dans le répertoire `resources/extracted`.

8.2 Extraction des lettres de la grille

L'extraction des lettres contenues dans la grille repose sur le même principe, mais les zones recherchées sont cette fois-ci tracées en cyan ($R=43, G=255, B=255$). L'algorithme identifie ces pixels cyan pour localiser les lettres. Après avoir mesuré et validé la boîte, la zone intérieure est extraite, puis redimensionnée à une taille standard de 32x32 pixels pour garantir l'uniformité des lettres. Cette étape utilise les fonctions `crop_image` et `resize_image`. Les images des lettres extraites sont sauvegardées dans le répertoire `resources/extracted/grid_letters` avec un nom correspondant à leurs coordonnées dans l'image d'origine. Pour signaler les zones déjà traitées et éviter les doublons, les bordures des lettres sont recoloriées en vert ($R=0, G=255, B=0$).

8.3 Extraction des mots à trouver

L'extraction des mots à trouver suit une logique similaire, mais elle s'intéresse aux zones bleues ($R=0, G=0, B=255$). L'algorithme détecte les pixels bleus, mesure la taille des boîtes qui délimitent les mots et vérifie que ces zones sont bien fermées par des bordures de même couleur. Une fois les validations effectuées, le contenu des zones bleues est extrait et sauvegardé dans des fichiers distincts, nommés `wordX.png`, où X représente un compteur pour chaque mot trouvé. Ces fichiers sont stockés dans le répertoire `resources/extracted/words`.

8.4 Extraction des lettres des mots

Enfin, l'extraction des lettres des mots concerne les zones tracées en rose ($R=255, G=192, B=203$). Lorsqu'une boîte rose est détectée, son contenu est extrait, puis re-

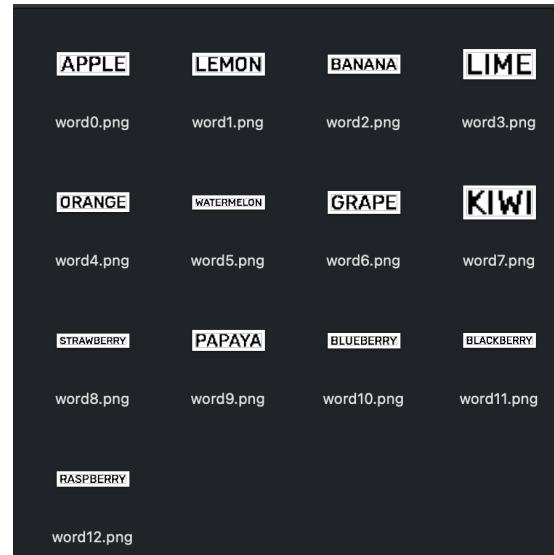
dimensionné à une taille standard de 32x32 pixels, comme pour les lettres de la grille. Chaque lettre est sauvegardée individuellement sous un nom unique, dans le répertoire `resources/extracted/word_letters`. Pour marquer les zones déjà traitées, les bordures des boîtes roses sont également recoloriées en vert.

8.5 Résultats

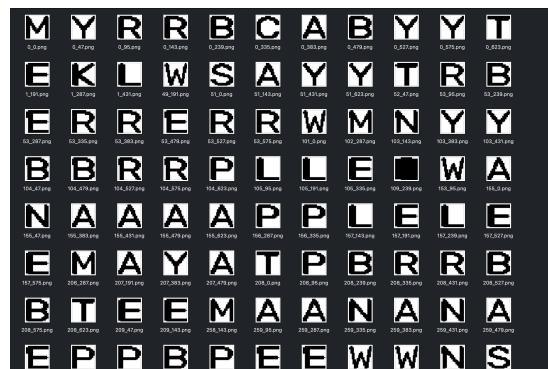
Ainsi, l'algorithme segmente efficacement l'image en différentes composantes (grille, mots et lettres) en exploitant des zones de couleurs préalablement tracées. Chaque étape garantit une extraction précise des informations visuelles tout en assurant l'uniformité des formats grâce aux opérations de recadrage et de redimensionnement. Cette approche permet de préparer les données pour des traitements ultérieurs tels que la reconnaissance des lettres et la reconstitution des mots. Voici quelques images pour illustrer notre propos :

M	S	W	A	T	E	R	M	E	L	O	N
Y	T	B	N	E	P	E	W	R	M	A	E
R	R	L	W	P	A	P	A	Y	A	N	A
R	A	N	L	E	M	O	N	A	N	E	P
E	W	L	E	A	P	R	I	A	B	P	R
B	B	I	L	B	B	W	B	R	L	A	Y
K	E	M	P	M	A	W	L	R	A	R	B
C	R	E	P	R	N	R	E	R	R	G	R
A	R	Y	A	Y	A	O	A	N	L	A	M
L	Y	Y	A	R	N	E	R	K	I	W	I
B	E	B	A	A	A	N	A	A	P	R	T
Y	R	R	E	B	P	S	A	R	N	N	W
Y	R	R	E	B	E	U	L	B	L	G	I
T	Y	P	A	T	E	A	E	P	A	C	E

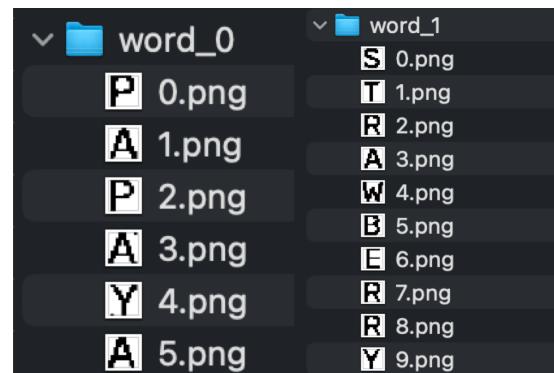
(a) Grille extraite



(b) Mots à trouver



(c) Lettres de la grille



(d) Lettres des mots

FIGURE 7 – Illustration des étapes d'extraction : grille, lettres et mots.

9 Réseau de neurones

Les réseaux neuronaux constituent un programme ou un modèle de machine learning qui prend des décisions d'une manière comparable au cerveau humain, en utilisant des processus qui reproduisent la façon dont les neurones biologiques fonctionnent de concert pour identifier des phénomènes, évaluer des options et arriver à des conclusions. (Source : IBM)

La rétropropagation est une technique de machine learning essentielle à l'optimisation des réseaux neuronaux artificiels. Elle facilite l'utilisation d'algorithmes de descente de gradient pour mettre à jour les poids du réseau, ce qui permet aux modèles d'apprentissage profond, qui sous-tendent l'intelligence artificielle moderne, d'« apprendre ». (Source : IBM)

9.1 Fonctionnement général

Pour ce projet, nous avons implémenté un **réseau de neurones multicouche** pour la reconnaissance de caractères. Cette approche repose sur une architecture classique organisée autour de trois composantes principales : une **couche d'entrée**, une ou plusieurs **couches intermédiaires** (couches cachées) et une **couche de sortie**. Chaque partie du réseau joue un rôle spécifique pour permettre au modèle d'apprendre à identifier les lettres à partir d'images pixelisées.

- **Initialisation et Allocation Mémoire** L'initialisation du réseau de neurones est une étape cruciale qui prépare les structures de données nécessaires pour stocker les paramètres du modèle. La mémoire est allouée dynamiquement pour les **poids**, les **biais** et les valeurs activées des couches intermédiaires et de sortie.

La couche d'entrée du réseau correspond directement aux pixels de l'image en entrée, où chaque pixel est traduit en une valeur binaire (0 pour un pixel noir, 1 pour un pixel blanc).

Les **poids** sont initialisés avec des valeurs aléatoires comprises entre -1 et 1 grâce à la fonction `init_weights`. Cette distribution aléatoire permet au réseau de partir avec une diversité suffisante pour l'apprentissage. Les **biais**, qui agissent comme des ajustements individuels pour chaque neurone, sont également initialisés aléatoirement.

- **Propagation Avant : Calcul des Sorties** La propagation avant est le cœur de

l’inférence dans un réseau de neurones. Lors de cette étape, les entrées (valeurs binaires des pixels) traversent successivement les couches du réseau pour produire une prédiction finale.

1. **Combinaison linéaire et activation** Pour chaque neurone de la couche cachée, la somme pondérée des entrées est calculée. Ensuite, une fonction d’activation non linéaire, la **sigmoïde**, est appliquée :

$$\text{Sigmoïde}(x) = \frac{1}{1 + e^{-x}} \quad (1)$$

Cette fonction permet de transformer la somme pondérée en une valeur comprise entre 0 et 1, introduisant ainsi la non-linéarité nécessaire pour modéliser des motifs complexes (comme des contours ou lignes).

2. **Couche de sortie et Softmax** Les sorties de la couche cachée sont ensuite transmises à la **couche de sortie**. Une transformation finale via la fonction **Softmax** convertit les activations en une distribution de probabilités normalisées :

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (2)$$

L’utilisation de Softmax garantit que la somme des probabilités est égale à 1, ce qui permet d’identifier la lettre ayant la plus haute probabilité comme étant la prédiction finale.

- **Rétropropagation et Ajustement des Poids** L’apprentissage repose sur un processus itératif appelé **rétropropagation**. Après avoir effectué la propagation avant, le réseau compare les prédictions aux **valeurs attendues** pour calculer l’**erreur**.

Cette erreur est utilisée pour ajuster les poids et les biais. Pour chaque neurone, le gradient indique la direction d’ajustement nécessaire afin de minimiser l’erreur :

$$w_{ij} \leftarrow w_{ij} - \eta \cdot \Delta w_{ij}, \quad (3)$$

où η est le **taux d’apprentissage** et Δw_{ij} représente le gradient calculé. La dérivée de la fonction sigmoïde, donnée par $x(1 - x)$, est utilisée pour moduler ces ajustements.

- **Prédiction des Lettres** Une fois le réseau entraîné, il peut être utilisé pour

prédire des lettres à partir de nouvelles images via la fonction `neural_predict`. Les étapes sont les suivantes :

1. L'image est chargée et convertie en valeurs binaires.
2. Les valeurs binaires sont transmises au réseau via la propagation avant.
3. La sortie ayant la probabilité la plus élevée est sélectionnée comme prédiction finale.

Les indices de 0 à 25 correspondent aux lettres majuscules de *A* à *Z*, tandis que les indices supérieurs sont utilisés pour les lettres minuscules.

- **Sauvegarde et Rechargement du Modèle** Pour éviter de réentraîner le réseau à chaque exécution, les poids et les biais peuvent être sauvegardés dans un fichier binaire via la fonction `save_neural_network`. À l'inverse, la fonction `load_neural_network` permet de recharger ces paramètres pour restaurer l'état du modèle.

9.2 Entrainement

Pour entraîner notre réseau de neurones, nous avons implémenté un processus de chargement d'un dataset d'images, suivi de l'entraînement d'un réseau de neurones pour effectuer une tâche de reconnaissance de caractères. La première partie du code est dédiée au chargement des images à partir d'un répertoire structuré par lettres (de *A* à *Z*), où chaque lettre représente une classe distincte. Chaque image est convertie en une structure `iImage`, qui contient ses dimensions, ses pixels, et son label associé. Une fonction, `load_dataset`, explore récursivement les sous-répertoires pour lire et charger les images tout en vérifiant que les chemins générés sont valides.

Une fois le dataset chargé, les images sont mélangées aléatoirement à l'aide de la fonction `shuffle_images`, pour assurer un apprentissage sans biais au cours des époques. La fonction principale `train` initialise un réseau de neurones et, si un fichier de poids préexistant est trouvé, charge ces poids pour continuer l'entraînement. Sinon, un nouvel entraînement est lancé en parcourant chaque image du dataset pour préparer les données d'entrée (normalisées à des valeurs binaires) et les labels attendus. À chaque itération, une passe avant (forward pass) et une rétropropagation (backpropagation) sont effectuées pour ajuster les poids du réseau selon un taux d'apprentissage fixe.

Enfin, une fois l'entraînement terminé, les poids du réseau sont sauvegardés dans un fichier, et toutes les ressources mémoire allouées (images, pixels, chemin, etc.) sont libérées pour éviter les fuites. Le code s'assure également de nettoyer les ressources de la bibliothèque `SDL` et `SDL_Image` utilisées pour la manipulation des images.

9.3 Sauvegarde de l'entraînement

Nous avons implémenté une fonction `save_neural_network` à pour but de sauvegarder les paramètres d'un réseau de neurones dans un fichier binaire. Ces paramètres incluent les poids et les biais des différentes couches du réseau. La fonction commence par ouvrir un fichier spécifié en mode binaire (`wb`) pour l'écriture. Si l'ouverture échoue, une erreur critique est générée à l'aide de la fonction `err`, ce qui garantit que l'utilisateur est informé du problème.

Les données du réseau sont ensuite écrites dans le fichier de manière séquentielle. Les poids reliant la couche d'entrée à la couche cachée sont sauvegardés en premier. Cela représente un total de `INPUTS_NUMBER × HIDDEN_NODES_NUMBER` doubles. Les biais de la couche cachée suivent, avec `HIDDEN_NODES_NUMBER` valeurs. Ensuite, les poids reliant la couche cachée à la couche de sortie, soit `HIDDEN_NODES_NUMBER × OUTPUTS_NUMBER` doubles, sont sauvegardés. Finalement, les biais de la couche de sortie sont ajoutés. Chaque ensemble de données est écrit directement en mémoire à l'aide de `fwrite`, garantissant un stockage rapide et sans conversion intermédiaire.

Une fois toutes les données sauvegardées, le fichier est fermé proprement avec `fclose`, et un message est affiché à l'utilisateur pour indiquer que l'opération a été effectuée avec succès.

En ce qui concerne le choix d'un fichier binaire (`.dat`) plutôt qu'un fichier texte (`.txt`), cela présente plusieurs avantages. Tout d'abord, un fichier binaire permet une lecture et une écriture plus rapides car les données sont stockées directement sous leur format mémoire natif, sans conversion en chaînes de caractères. Cette approche minimise le temps d'accès disque et l'utilisation des ressources processeur.

De plus, un fichier binaire conserve la précision des données. Contrairement à un fichier texte où les nombres flottants sont convertis en représentations textuelles qui peuvent introduire des erreurs d'arrondi, un fichier binaire stocke les valeurs exactes. Cela est crucial pour les réseaux de neurones, où des petites variations dans les poids peuvent avoir un impact significatif sur les performances.

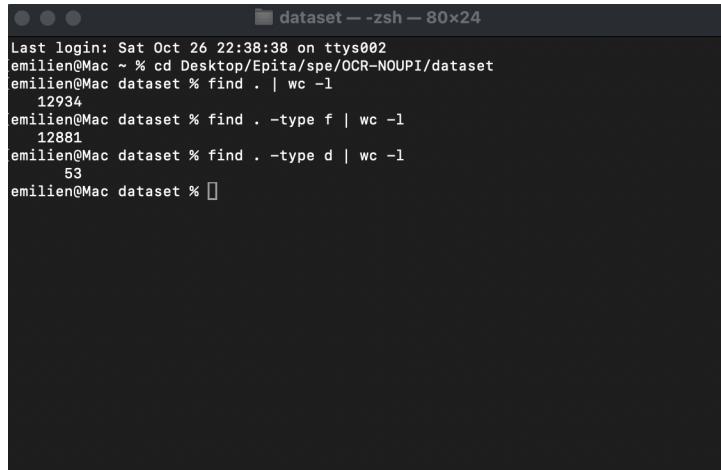
L'utilisation d'un fichier binaire permet également de réduire la taille des fichiers générés. Les fichiers texte incluent des caractères supplémentaires, comme les espaces ou les sauts de ligne, qui augmentent la taille totale. En revanche, un fichier binaire ne contient que les données elles-mêmes, ce qui le rend plus compact.

Enfin, un fichier binaire est plus difficilement lisible et modifiable par un utilisateur non averti, offrant une certaine protection contre les modifications accidentelles ou intentionnelles. Cela est particulièrement utile pour garantir l'intégrité des données du réseau de neurones entre différentes sessions d'entraînement ou d'utilisation.

En somme, le choix d'un fichier binaire pour la sauvegarde des paramètres d'un réseau de neurones garantit une meilleure efficacité, une précision accrue et une gestion plus robuste des données, en simplifiant à la fois leur sauvegarde et leur chargement ultérieur.

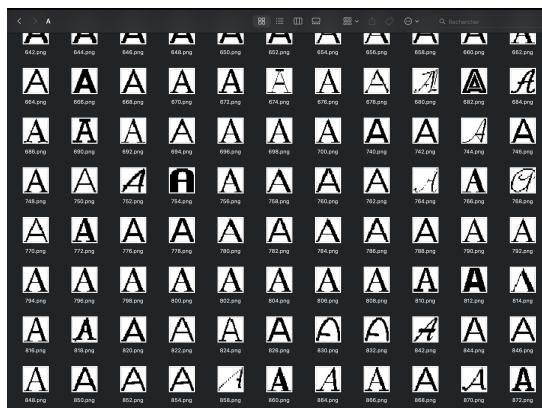
9.4 Jeu de données

Pour nous constituer une banque de données suffisamment importante, nous avons récupéré toutes les polices qui étaient déjà préinstallées dans le système d'exploitation Mac OS, puis nous les avons tracées à l'aide d'un script Python. Ensuite, nous les avons rognées au maximum et redimensionnées dans un format 32 pixels par 32 pixels, à l'aide des fonctions `crop` et `resize` que nous avons implémentées. Grâce à cette méthode, nous avons un dataset d'un peu moins de 13 000 images, majuscules et minuscules confondues, sur lequel nous pouvons entraîner notre réseau de neurones.

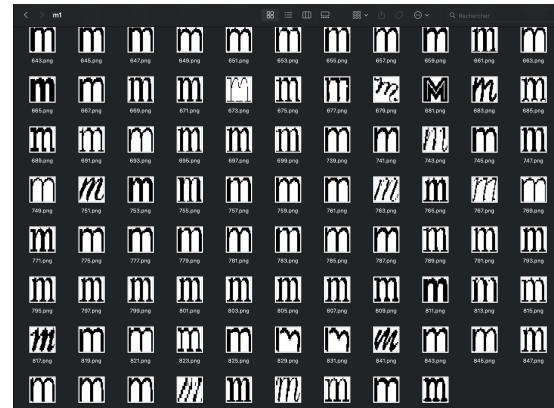


```
Last login: Sat Oct 26 22:38:38 on ttys002
emilien@Mac ~ % cd Desktop/Epita/spe/OCR-NOUPI/dataset
emilien@Mac dataset % find . | wc -l
12934
emilien@Mac dataset % find . -type f | wc -l
12881
emilien@Mac dataset % find . -type d | wc -l
53
emilien@Mac dataset %
```

(a) Taille du dataset



(b) Exemple sur majuscule



(c) Exemple sur minuscule

FIGURE 8 – Illustrations du dataset

10 Reconstructions

Dans cette partie, le code exploite un réseau de neurones déjà entraîné pour la reconnaissance de caractères. Après avoir vérifié la présence du fichier contenant les poids entraînés (`neural_network_weights.dat`), le programme charge l'intégralité du modèle,

incluant l'architecture et les paramètres optimisés lors de l'apprentissage. Chaque image de lettre, extraite au préalable, est convertie en une matrice binaire (0 ou 1), puis injectée dans le réseau lors d'un passage avant (*forward pass*). Au fil des couches, la combinaison linéaire et non linéaire des valeurs permet d'obtenir une sortie où la classe la plus probable correspond à la lettre reconnue. Cette identification est réalisée de manière autonome, fondée sur l'expertise intégrée dans le réseau, et fournit ainsi une reconnaissance fiable et automatisée des caractères.

10.1 Sauvegarde des mots dans un fichier texte

Une fois chaque lettre identifiée, le code s'emploie à les rassembler afin de former des mots complets. Dans un premier temps, ces lettres sont inscrites dans un fichier texte temporaire. Le programme élimine ensuite les lignes vides pour ne conserver qu'un résultat nettoyé, plus lisible. Ce processus aboutit à un fichier final où chaque mot ou séquence de lettres apparaît clairement, sans espaces superflus. La transformation est donc complète : la reconnaissance individuelle des caractères s'intègre à une vue d'ensemble, reconstituant du texte exploitable, prêt à être réutilisé, diffusé ou analysé.

10.2 Sauvegarde de la grille dans un fichier texte

En parallèle de la reconnaissance des lettres, la dimension spatiale est conservée. Les noms des fichiers images contiennent déjà des indices de position (coordonnées x, y) ; le code extrait ces informations afin de les enregistrer dans un fichier texte dédié. Cette grille de coordonnées permet de restituer la position exacte des lettres, reflétant la mise en page initiale. Au-delà de la simple liste de caractères reconnus, on dispose ainsi d'une carte précise, ancrant chaque lettre dans l'espace. Cette démarche facilite la reconstitution d'un document complexifié, aligné sur l'original, et pose les bases d'un travail ultérieur d'analyse ou de présentation.

10.3 Chargement de la grille depuis le fichier texte

La sauvegarde de la grille et du texte au format brut offre la possibilité de recharger ces données à volonté. Le code peut ainsi rouvrir les fichiers générés pour reconstituer la structure spatiale et textuelle du document initial. Cette opération consiste à relire, découper et interpréter le contenu textuel, afin de recréer en mémoire l'agencement des lettres et leurs positions. Ce chargement différé permet de reprendre le flux de travail à tout moment, sans devoir répéter la phase initiale de détection. On retrouve instantanément le lien entre le texte et sa géométrie, crucial pour le traitement en aval : corrections, mises en page supplémentaires, ou export vers d'autres formats.

10.4 Réajustement de la grille

L'étape finale vise à rendre la grille plus cohérente et fidèle à l'organisation originale du document. Le code réajuste les coordonnées des lettres, fusionnant celles situées à des hauteurs très proches pour refléter l'appartenance à une même ligne. Ce traitement affine la représentation textuelle, garantissant un alignement plus naturel. Le tri des lettres selon leurs positions spatiales offre un résultat ordonné, lisible, et plus proche de la réalité de la source. Le produit final n'est plus seulement une accumulation de données, mais un tableau ajusté et cohérent, prêt à être affiché, analysé ou exporté dans des conditions optimales.

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

(a) Grille de base

```
1 PXUTSINIUPRVGBMDD
2 EHAASPOJPETBEQZLC
3 AUNTEGQTLHRZFATOP
4 SHXFNGUAXEAAYPOMH
5 YOYYLDXLAKYUZLBSK
6 JXMUUGQTRIMAGINEB
7 PBBSHFNWFXHDBTNVK
8 HIIHDESQFUMYERNSX
9 RPBZNHSDSLHONBSSS
10 EHXAIZIHAHOESQFEF
11 CWVDCJSIGRZIMVMW
12 LAIIRZQQHxDZozQTR
13 WCAXEZRGHAIZNECSE
14 BRHFOTGNITSEREOVZ
15 WVWQDIHWQTSBILMUM
16 TDTONZCXRGELKFQ
17 QNEKSVMOTFALAAEWB
```

(b) Avant réajustement

```
1 PXUTSINIUPRVGBMDD
2 EHAASPOJPETBEQZLC
3 AUNTEGQTLHRZFATOP
4 SHXFNGUAXEAAYPOMH
5 YOYYLDXLAKYUZLBSK
6 JXMUUGQTRIMAGINEB
7 HFNWFXHDPBBBTNVSK
8 HIIHDESQFUMYERNSX
9 RPBZNHSDSLHONBSSS
10 EHXAIZIHAHOESQFEF
11 CWZIMVDCJVSSIMGRW
12 LAIIRZQQHxDZozQTR
13 WCAXEZRGHAIZNECSE
14 BRHFOTGNITSEREOVZ
15 MWVWQDUIHWQTSBIML
16 TDTONZCXXRGELKFQ
17 QNEKSVMOTFALAAEWB
```

(c) Après réajustement

FIGURE 9 – Avant/Après réajustement de la grille

11 Résolution de la grille

11.1 Le Solver

Le résolveur a été la première chose que nous avons réalisée durant ce projet, car il nous semblait être une des tâches les plus simples. Notre algorithme est simple : il consiste à charger la grille qui se trouve dans un fichier puis, à partir du contenu de ce fichier, le script crée une matrice de taille $n \times p$, avec n le nombre de lignes et p le nombre de colonnes.

Une fois la matrice créée, le script parcourt les éléments de celle-ci jusqu'à trouver une occurrence de la première lettre du mot recherché. Ensuite, l'algorithme cherche de manière horizontale, puis verticale, puis enfin en diagonale. Avant de se lancer dans une telle recherche, le script vérifie évidemment s'il y a suffisamment de place par rapport aux extrémités de la matrice pour que le mot s'y cache. Si le mot n'est pas trouvé à partir de cette occurrence, le parcours de la matrice reprend à la lettre suivante.

Si, finalement, la matrice est parcourue entièrement sans avoir trouvé le mot, le programme renvoie la valeur `None`, sinon un couple constitué des coordonnées de la première et de la dernière lettre.

11.2 Recherche des mots dans la grille

Pour trouver les mots présents dans la grille (que nous avons écrits dans un fichier `text`), nous appliquons notre *solver* précédemment décrit à la matrice représentant la grille, préalablement chargée à partir d'un fichier texte. Le rôle du *solver* est de rechercher, dans cette matrice, un mot donné et de renvoyer les coordonnées de la première et de la dernière lettre du mot s'il est trouvé. Ces coordonnées sont exprimées dans le référentiel de la matrice.

Une fois ces coordonnées récupérées, il est nécessaire de les ajuster pour les exprimer dans le référentiel de l'image principale contenant la grille. Pour ce faire, nous procédons en deux étapes :

1. Nous utilisons les coordonnées de la première et de la dernière lettre dans la matrice pour identifier les coordonnées correspondantes dans la grille. Cette opération est rendue possible grâce à un fichier texte distinct dans lequel nous avons au préalable enregistré les coordonnées précises de chaque lettre au sein de la grille.
2. Nous ajoutons ensuite un décalage global pour compenser la différence entre l'origine de la grille (dans l'image principale) et l'origine de la matrice. Ce décalage est déterminé en fonction de la position de la grille dans l'image principale.

De plus, un ajustement supplémentaire est appliqué aux coordonnées via un tri en miroir, ce qui permet de s'assurer que les valeurs récupérées correspondent parfaitement à la disposition réelle des lettres dans l'image.

Une fois les coordonnées ajustées et correctement référencées, nous les transmettons à nos fonctions d'entourage. Ces fonctions, que nous présenterons dans la section suivante, permettent de mettre en évidence les mots détectés directement dans l'image.

Cette approche structurée et méthodique garantit une détection précise des mots tout en prenant en compte les décalages et transformations nécessaires entre les référentiels de la matrice et de l'image dont voici un exemple :

```
resources > extracted > txt_data > letters.txt
1  MSWATERMELON
2  YTBNPEPEWRMAE
3  RRLWPAPAYANA
4  RANLEMONANEP
5  EWLEAPRIABPR
6  BBILBBWBRLAY
7  KEMPMMAWLARB
8  CREPRNRERRGR
9  ARYAYAOANLAM
10 LYYARNERKIWI
11 BEBAAANAAPRT
12 YRREBPSARNNW
13 YRREBEULBLGI
14 TYPATEAEPACE
```

(a) Lettres dans la grille

```
resources > extracted > txt_data > coordinates.txt
1  (0,0) (51,0) (101,0) (155,0) (208,0) (261,0) (312,0) (362,0) (417,0) (469,0) (520,0) (571,0)
2  (0,47) (52,47) (104,47) (155,47) (209,47) (261,47) (313,47) (361,47) (416,47) (466,47) (519,47) (573,47)
3  (0,95) (53,95) (105,95) (153,95) (208,95) (259,95) (312,95) (363,95) (415,95) (467,95) (519,95) (571,95)
4  (0,143) (51,143) (103,143) (157,143) (209,143) (256,143) (312,143) (363,143) (415,143) (467,143) (521,143) (572,143)
5  (1,191) (49,191) (105,191) (157,191) (207,191) (261,191) (312,191) (369,191) (415,191) (469,191) (520,191) (572,191)
6  (0,239) (53,239) (109,239) (157,239) (208,239) (261,239) (309,239) (364,239) (416,239) (469,239) (519,239) (571,239)
7  (1,287) (53,287) (102,287) (156,287) (206,287) (259,287) (309,287) (365,287) (416,287) (467,287) (520,287) (572,287)
8  (0,335) (53,335) (105,335) (156,335) (208,335) (259,335) (312,335) (365,335) (416,335) (469,335) (519,335) (572,335)
9  (0,383) (53,383) (103,383) (155,383) (207,383) (259,383) (312,383) (363,383) (415,383) (469,383) (519,383) (570,383)
10 (1,431) (51,431) (103,431) (155,431) (208,431) (259,431) (313,431) (364,431) (417,431) (473,431) (517,431) (577,431)
11 (0,479) (53,479) (104,479) (155,479) (207,479) (259,479) (311,479) (363,479) (415,479) (469,479) (520,479) (572,479)
12 (0,527) (53,527) (104,527) (157,527) (208,527) (261,527) (311,527) (363,527) (416,527) (467,527) (519,527) (569,527)
13 (0,575) (53,575) (104,575) (157,575) (208,575) (261,575) (311,575) (365,575) (416,575) (469,575) (519,575) (577,575)
14 (0,623) (51,623) (104,623) (155,623) (208,623) (261,623) (311,623) (365,623) (416,623) (467,623) (520,623) (573,623)
```

(b) Coordonnées correspondantes

FIGURE 10 – Recherche des mots et de leurs coordonnées

11.3 Mise en évidence des mots dans la grille

Pour entourer les mots détectés dans la grille, nous avons mis en place un processus en plusieurs étapes alliant analyse algorithmique et traitement graphique.

Dans un premier temps, nous lisons la liste des mots à rechercher depuis un fichier texte nommé `words.txt`. Pour chaque mot, nous utilisons la fonction `solver` afin de déterminer ses positions de départ et d'arrivée dans la grille. Ces positions, initialement exprimées en indices de grille, sont stockées sous forme de tuples (`t1` pour le point de départ et `t2` pour le point d'arrivée). Nous convertissons ensuite ces indices en coordonnées pixels en exploitant les informations de position extraites du fichier `coordinates.txt`.

Une fois les coordonnées des mots récupérées, nous analysons leur orientation pour appliquer le traitement graphique approprié :

- Pour un mot orienté **horizontalement**, nous traçons un rectangle autour de celui-ci en ajustant sa largeur en fonction de la longueur du mot.
- Pour un mot positionné **verticalement**, nous dessinons un rectangle adapté en hauteur afin d'englober les lettres.
- Lorsque les mots sont orientés en **diagonale**, nous utilisons la fonction `draw_diagonal` pour tracer une ligne inclinée reliant les positions de départ et d'arrivée du mot, qu'elle soit ascendante ou descendante.

Afin d'assurer une meilleure lisibilité des résultats, nous attribuons une **couleur aléatoire** à chaque mot détecté. Cela permet de les différencier visuellement. Une fois l'ensemble des mots traités, nous sauvegardons l'image finale, contenant les mots entourés, dans un fichier de sortie nommé `solved.png`. Ce fichier constitue le résultat final de notre traitement, prêt à être présenté pour validation ou analyse.

12 Interface utilisateur

L'interface utilisateur a pour but de rendre le solveur OCR NOUPI à la fois accessible et intuitif. L'objectif principal est de proposer un design clair et intuitif qui permet aux utilisateurs de profiter pleinement des capacités du solveur tout en garantissant une bonne expérience d'utilisation.

12.1 Architecture de l'interface

L'interface est divisée en trois zones principales pour simplifier la navigation :

- **Menu latéral** : Il regroupe les fonctionnalités clés (*sauvegarde, prétraitement, rotation* et *entraînement du modèle*). Les options sont disposées verticalement pour suivre un flux logique, guidant l'utilisateur pas à pas.
- **Zone centrale** : Cette zone affiche la grille des mots croisés et les résultats du solveur. Les mots détectés sont entourés de cadres colorés pour faciliter leur visualisation.
- **Barre d'action** : Deux boutons permettent de lancer l'analyse (**Solve**) ou de réinitialiser la grille (**Reset**), avec des couleurs (**vert** / **rouge**) pour rendre leurs fonctions évidentes.

12.2 Principes d'ergonomie

Le thème sombre de l'interface a été choisi pour réduire la fatigue visuelle et mettre en avant les résultats et les boutons. J'ai utilisé CSS pour personnaliser le style et garantir une esthétique propre et minimaliste, adaptée à une utilisation prolongée. Le choix des couleurs a été fait avec soin pour garantir un bon contraste et une lisibilité optimale, même pour des utilisateurs travaillant sur des écrans de résolution moyenne.

Les maquettes ci-dessous ont été conçues pour orienter le développement et définir une direction claire à suivre.

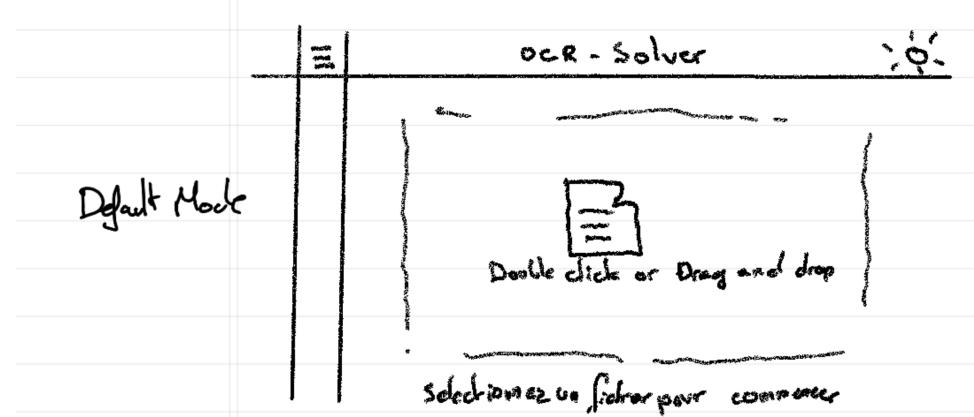


FIGURE 11 – Maquette 1 : Menu latéral

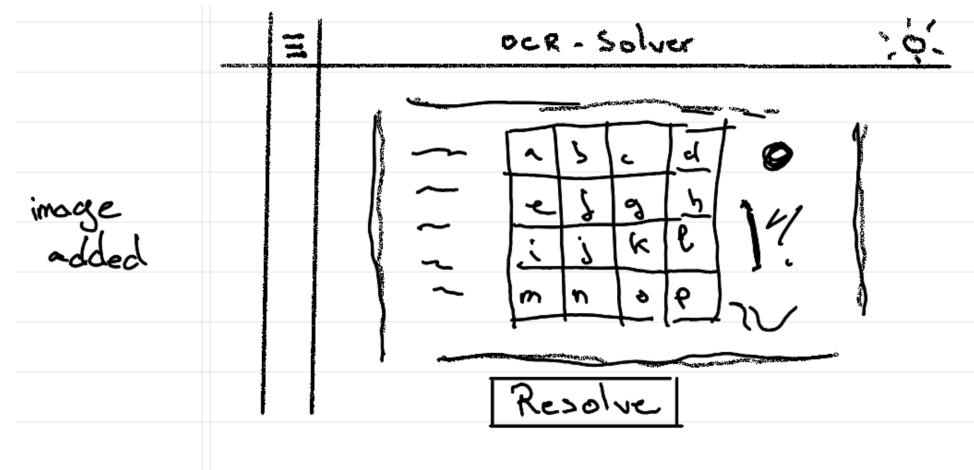


FIGURE 12 – Maquette 2 : Zone centrale

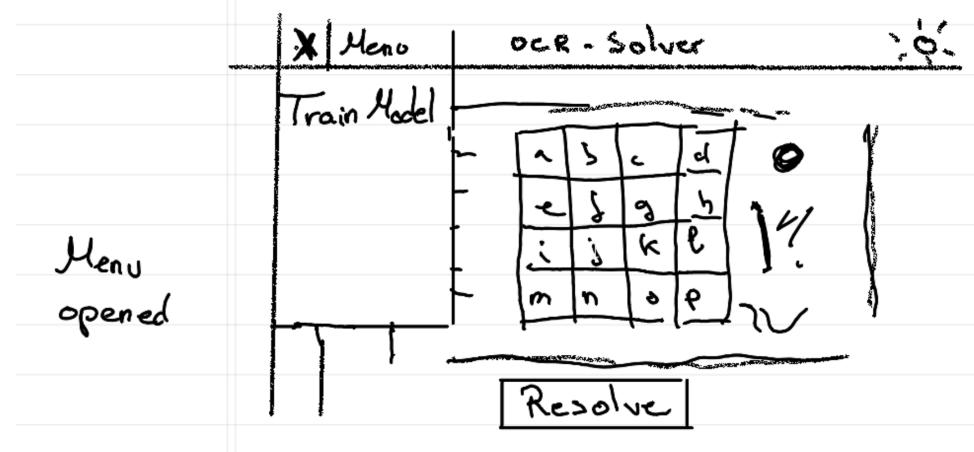


FIGURE 13 – Maquette 3 : Barre d'action

12.3 Implémentation

L'interface a été développée avec GTK pour la partie graphique et personnalisée avec CSS. Le back-end en C lie le moteur OCR du projet aux différentes fonctionnalités proposées par l'application. L'intégration entre l'interface et le solveur permet un retour visuel en temps réel, ce qui était une priorité pour offrir une expérience fluide, dont voici le résultat final :



FIGURE 14 – UI au démarrage

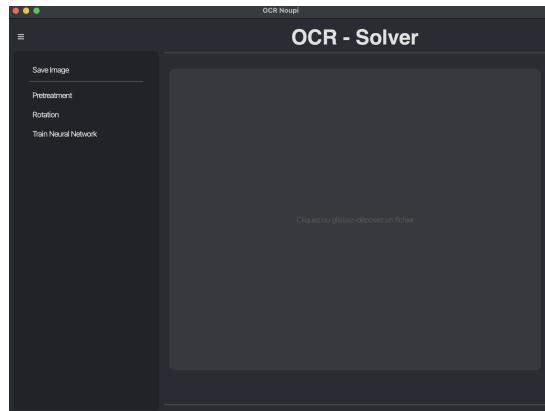


FIGURE 15 – UI avec menu ouvert

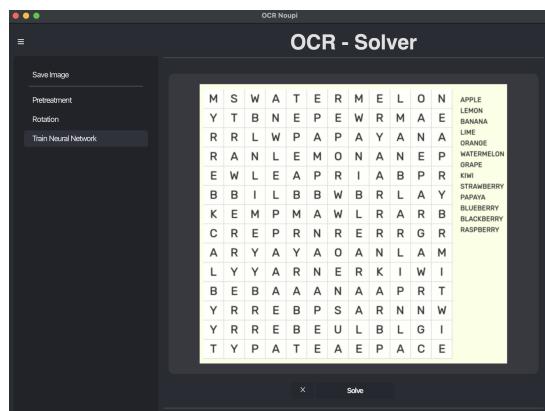


FIGURE 16 – UI avec une image chargée

13 Avancement final

Voici la répartition de l'avancement final estimé pour chacune des tâches que nous avons répertoriées :

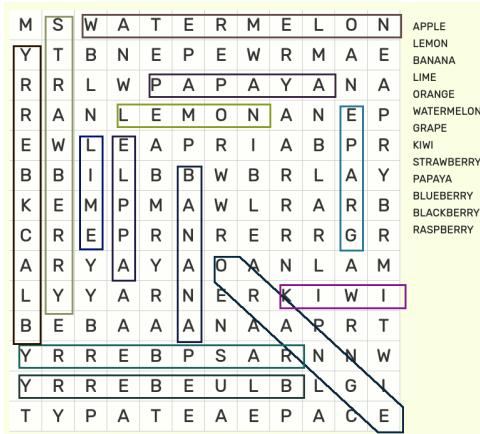
Tâches	Avancement
Gestion de l'image	
Prétraitement	95%
Rotation de l'image	100%
Détections & extractions	100%
Réseau de Neurones	
XNOR	100%
Entraînement du réseau de neurones	100%
Reconnaissance des lettres	95%
Sauvegarde & chargement des poids	100%
Données d'entraînement	100%
Résolution de la grille	
Chargement & réajustement de la grille	100%
Chargement des mots	100%
Solver	100%
Entourage des mots	100%
Autre	
Designs / UI	100%

TABLE 2 – Tableau d'avancement final

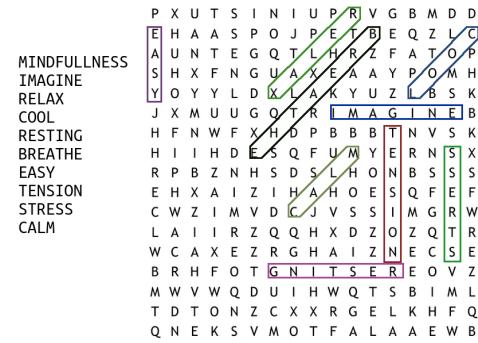
Vous remarquerez que deux tâches ne sont pas terminées à 100%, c'est parce que nous avons finalement préféré ne pas entraîner notre réseau de neurones sur des lettres minuscules pour ne pas fausser les reconnaissances sur les autres grilles (car il n'y a qu'une seule grille qui nécessite la reconnaissances de lettres minuscules). Concernant le prétraitement, nous avons fait le choix de sacrifier l'image 1 du niveau deux pour nous concentrer sur les autres images.

14 Résultats finaux

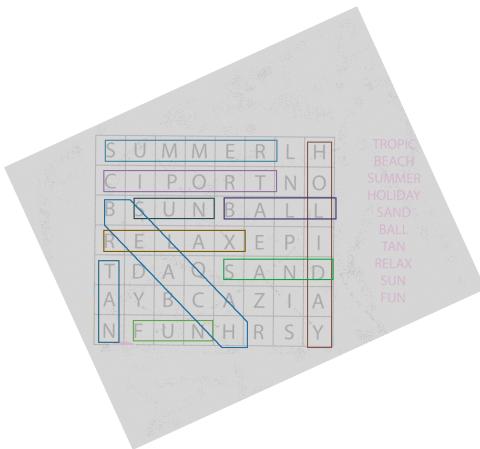
Après un nombre incalculable d'heures passées sur ce projet pour certains, voici les résultats obtenus sur les grilles fournies dans le sujet. Vous remarquerez que les images 2.1 et 4.2 ne sont pas présentes. Cela s'explique par deux raisons : nous n'avons pas réussi à prétraiter correctement l'image 2.1, et nous avons finalement choisi de ne pas entraîner notre réseau de neurones avec des lettres minuscules, de peur de fausser les résultats.



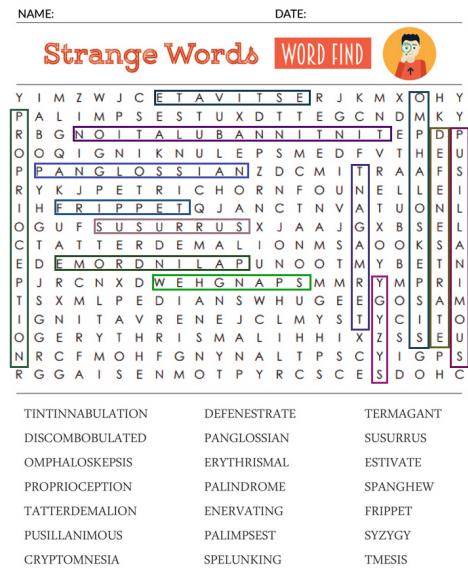
(a) Image 1.1



(b) Image 1.2



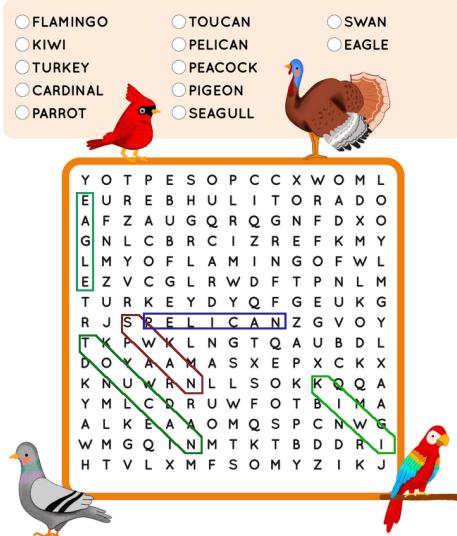
(c) Image 2.2



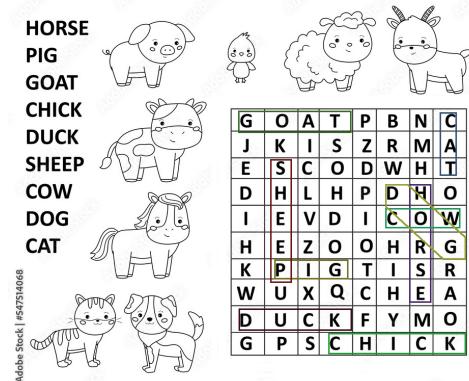
(d) Image 3.1

FIGURE 17 – Résultats obtenus sur les grilles fournies (1/2)

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



(a) Image 3.2



(b) Image 4.2

FIGURE 18 – Résultats obtenus sur les grilles fournies (2/2)

15 Impressions de chacun

15.1 DEFONTAINE JOLIVET Emilien

Me concernant, ce projet n'a pas toujours été facile. Il a fallut composer avec les forces, les faiblesses mais aussi les disponibilités de chacun... Néanmoins, du fait que j'ai été engagé sur pratiquement tous les fronts de ce projet, j'ai beaucoup appris que ce soit en C, en LaTeX et en culture générale, notamment sur les réseaux de neurones et le prétraitement des images. Je suis tout de même fier de ce que nous avons réussi à produire et espère pouvoir faire encore mieux pour les prochains projets à venir tout au long de ma scolarité à l'EPITA.

15.2 TOUSSAINT Ulysse

Ce projet m'a permis de travailler sur un outil OCR en découvrant de nouvelles approches et librairies comme GTK pour concevoir une interface utilisateur adaptée. J'ai trouvé intéressant de voir comment intégrer des solutions techniques pour rendre un processus complexe plus intuitif et accessible. Cette expérience m'a apporté une vision plus concrète des enjeux liés au développement d'un tel projet.

15.3 FORGET Clément

Les objectifs principaux du projet OCR ont été atteints. Nous avons réussi à développer un système fonctionnel malgré quelques limites, notamment au niveau de la précision

et de l'optimisation des performances. Ces aspects pourraient être améliorés avec davantage de temps et de ressources, mais le résultat final est satisfaisant.

15.4 BIGOT Lucas

J'ai pris beaucoup de plaisir à réaliser ce projet, même si cela aurait été plus simple avec une équipe plus disponible et investie. Cela m'aurait permis de me concentrer davantage sur certains aspects du projet. Cette expérience m'a permis de développer de nouvelles compétences et d'apprendre des choses que je n'aurais peut-être pas découvertes autrement. Bien qu'il y ait toujours des points à améliorer, je suis satisfait du résultat final et fier de ce que nous avons accompli. J'espère faire encore mieux dans mes futurs projets à l'EPITA.

16 Conclusion

Pour conclure, nous sommes tous très fier du travail que nous avons réalisés et sommes content du résultat obtenu et nous espérons que notre logiciel OCR-NOUPI saura répondre aux attentes de ses utilisateurs.