

# Projet OCR – Soutenance Intermédiaire

Groupe OCR-NOUPI

Novembre 2024



FIGURE 1 – Logo

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Les membres d'OCR-NOUPI</b>	<b>4</b>
2.1	DEFONTAINE JOLIVET Emilien . . . . .	4
2.2	TOUSSAINT Ulysse . . . . .	4
2.3	FORGET Clément . . . . .	4
2.4	BIGOT Lucas . . . . .	4
<b>3</b>	<b>Technologies utilisées</b>	<b>5</b>
<b>4</b>	<b>Répartition des tâches</b>	<b>5</b>
<b>5</b>	<b>Prétraitement</b>	<b>5</b>
5.1	Passage en niveaux de gris . . . . .	5
5.2	Augmentation des contrastes . . . . .	6
5.3	Flou Gaussien . . . . .	6
5.4	Binarisation de l'image . . . . .	6
5.5	Méthode d'Otsu . . . . .	6
5.6	Redressage manuel de l'image . . . . .	7
5.7	Fonctions utiles . . . . .	7
5.8	Résultats . . . . .	8
<b>6</b>	<b>Détections</b>	<b>8</b>
6.1	Détection des contours par la méthode de Canny . . . . .	9
6.2	Détection de la grille . . . . .	9
6.3	Détection des mots . . . . .	10
6.4	Détection des lettres des mots (et des lettres des grilles) . . . . .	10
6.5	Extractions . . . . .	10
6.6	Fonctions utiles . . . . .	11
6.7	Résultats . . . . .	12
6.8	Interprétation des résultats . . . . .	15
<b>7</b>	<b>Réseau de Neurones</b>	<b>15</b>
7.1	Structure globale . . . . .	15
7.2	XNOR . . . . .	16
7.3	Données d'entraînement . . . . .	17
<b>8</b>	<b>Résolveur de mots croisés</b>	<b>17</b>
<b>9</b>	<b>Avancée générale</b>	<b>18</b>
<b>10</b>	<b>Conclusion</b>	<b>19</b>

# Table des figures

1	Logo . . . . .	1
2	Résultats obtenus après pré-traitement . . . . .	8
3	Schéma modélisant le processus . . . . .	10

4	Sauvegarde des lettres extraites . . . . .	11
5	Résultats obtenus pour les détections de grilles . . . . .	13
6	Résultats obtenus pour les détections des mots . . . . .	14
7	Résultats obtenus pour les détections des lettres dans un mot . . . . .	14
8	Entraînement du réseau de neurones . . . . .	16
9	Illustrations du dataset . . . . .	17

# 1 Introduction

Ce document a pour but de présenter l'avancée du groupe OCR-NOUPI au terme de la soutenance intermédiaire du projet OCR du S3 à Epita. Au cours de ce rapport, nous allons aborder plusieurs aspects de notre projet jusqu'à présent comme les difficultés que nous avons rencontrées, les algorithmes que nous avons déjà implémentés ou encore notre ressenti global.

## 2 Les membres d'OCR-NOUPI

### 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 Technologies utilisées

Pour réaliser ce projet, nous avons utilisé **Visual Studio Code (VSCode)** comme éditeur de code, un outil particulièrement efficace et flexible pour le développement. Nous avons également utilisé le langage de programmation **C** car imposé et nous sommes limités à ses bibliothèques natives, comme `stdio.h`, `stdlib.h` et `string.h` par exemple, qui offrent des fonctionnalités de base puissantes et utiles pour la gestion des entrées/sorties, de la mémoire, et des chaînes de caractères. Cette approche nous a permis de construire un code relativement "propre". De plus, nous avons utilisé `SDL2.h` pour gérer les images et nous nous servons de `gtk 3.0` pour réaliser l'interface utilisateur.

### 4 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			
Rotation de l'image	Second			Principal
Détections & extractions	Second		Principal	
Réseau de Neurones				
XNOR	Second	Principal		
Reconnaissance des lettres	Second	Principal		
Données d'entraînement	Principal		Second	
Autre				
Solver	Principal			
Designs / UI				Principal
Rapports	Principal			

TABLE 1 – Tableau de répartition des tâches

### 5 Prétraitement

#### 5.1 Passage en niveaux de gris

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.

## 5.2 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'étirage" 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  $\alpha$  afin de rendre les différences d'intensité encore plus marquées. La valeur de la nouvelle intensité est donc définie par :

$$\text{nouvelle\_intensité} = \alpha \times (\text{intensité} - \text{intensité\_moyenne}) + \text{intensité\_moyenne}$$

## 5.3 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  $\sigma$  représente l'écart-type de la distribution. Plus  $\sigma$  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.

## 5.4 Binarisation de l'image

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). Cependant, nous n'utilisons pas vraiment cette méthode mais une dérivée de celle-ci qui est bien plus puissante et utile dans notre cas : la méthode d'Otsu.

## 5.5 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. D'abord, la fonction `compute_otsu_threshold` 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.

## 5.6 Redressage manuel de l'image

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  $\theta$  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.

## 5.7 Fonctions utiles

Nous avons également développé des fonctions "utiles" comme par exemple une fonction qui inverse les couleurs d'une image binarisée, une fonction qui rogne une image au maximum (en cherchant les pixels blancs les plus "à l'extérieur") et une fonction qui permet de redimensionner une image en utilisant un coefficient de grossissement. Nous avons également implémenté une image qui permet de supprimer les petits groupes de pixels.

## 5.8 Résultats

Une fois que nous avons appliqué chacune des étapes mentionnées, nous obtenons les résultats suivants, avec un prétraitement qui ne marche pas bien sur l'image 2.1, à cause du bruit :

MSWATERMELON APPLE  
YTBNEPEWRMAE LEMON  
RRLWPAPAYANA BANANA  
RANLEMONANE LINE  
EWLEAPRIABPR ORANGE  
BBILBBWBRLAY WATERMELON  
KEMPMAWLARB ORANGE  
CREPRNRERRGR KUMI  
ARYAYAOANLAM STRAWBERRY  
LYYARNERNKIWI PAPAYA  
BEBAAANAAPRT BLUEBERRY  
YRREBPSARNNW BLACKBERRY  
YRREBEULBLGI RASPBERRY  
TYPATEAEPACE

(a) Image 1.1

PXUTSINIUPRVGBMDD  
EHAASPOJPETBEQZLC  
AUNTEGQTLHRZFATOP  
SHXFNGUAXEAAYPOMH  
YOYYLDXLAKYUZLBSK  
JXMUUGQTRIMAGINEB  
HFNWFXHDPBBBTNVSK  
HIIHDESQFUMYERNXS  
RPBZNHSDSLHONBSSS  
EHXAIZIHAEDESQFEF  
CWZIMVDCJYSSIMGRW  
LAIIRZQQHXDZQZTR  
WCAXEZRGHAIZNECSE  
BRHFOTGNITSEREOVZ  
MWVWQDUIHWQTSBIML  
TDTONZCXXRGELKHFQ  
QNEKSYMOTFALAAEWB

MINDFULLNESS  
IMAGINE  
RELAX  
COOL  
RESTING  
BREATHE  
EASY  
TENSION  
STRESS  
CALM

(b) Image 1.2

HG  
AO  
ZP  
F  
ZE  
M  
ATESEU  
W  
RXDAR  
FJUITEDT Y RT

(c) Image 2.1

SUMMERLH  
CIPORTNO  
BSUNBALL  
RELAXEPI  
TDAQSAND  
AYBCAZIA  
NFUNHRSY

TROPIC  
BEACH  
SUYMER  
HOLIDAY  
SAND  
BALL  
TAN  
RELAX  
SUN  
FUN

(d) Image 2.2

NAME: DATE:

**Strange Words**

YIMZWJJCETAVITTSERJEMXOMY  
PALIMPSESTUXDITTEGCNDMKY  
RSGNOITALUBAHNITNITEPDP  
OOQIGNIKNULEPSMEDFVTHEU  
PPANGLOSSIANZDCMITRAAFS  
RYKJPETRICHOENHFOUNELLEI  
IHFRIPFETQJANCTHYATUONL  
OGUFSUSURRUSXJAAJGXBSSE  
CTATTERDEMALLIONMSAOKESA  
EDEMORDNILAPUNOOTMYSETN  
PJRCNXPEDIANSWHPMMRYMPRI  
TSXMLPEDIANSWHPMMRYMPRI  
IGNITAVRENEJCLMYSTYCITO  
OGERYTHRISMALEHIXZSSEU  
NRCFMOHFGNYNALTPTSCYIGPS  
RGGAISENMOTPYRCSCESDONC

TINTINNABULATION DEFENESTRATE TERMAGANT  
DISCOMBULATED PANGLOSSIAN SUSURUS  
OMPHALOSKEPSIS ERYTHRISMAL ESTIVATE  
PROPRIOCEPTION PALINDROME SPANGHEW  
TATTERDEMATION ENERVATING FRIPPET  
PUSILLANIMOUS PALIMPSEST SYZYGY  
CRYPTOMANTISIA SPELLUNKING TMESIS

(e) Image 3.1

Find the words below and circle them (across down or diagonally)

FLAMINGO TOUCAN SWAN  
KIWI PELICAN EAGLE  
TURKEY PEACOCK  
CARDINAL PIGEON  
PARROT SEAGULL

YOTFESOPCCXWOML  
EUREBHULITORADO  
AFZAUGQRQGNFDXO  
GNLCBRCIZREFKMY  
LMYOFAMINGOFWL  
EZVCGLRWDFTPNLM  
TURKEYDYQFGEUKG  
RJSPELICANZGVOY  
TKPWKLNGTQAUBDL  
DOYAAAMASXEPCKX  
KNURWNLSSOKKQQA  
YMLCDRUWFOTBIMA  
ALKEAAMQSPCNWG  
WMGQINMTKTBDRI  
HTVLXMFSSOMYZIKJ

(f) Image 3.2

FA

WCIZGUXDBNRELS  
QXPVODHAYIMTFK  
GNLFARMJKCSZWE  
HVDOTIBSLUEDOG  
ARWCKEPOMAFJYIT  
BSHEEEPOMAJDYIT  
QZTRGWCKGRASSNL  
FDTRNGWCIMPOJUB  
OAKYSJAZEHTQPV  
OMQXLFTURLANIC  
DLJBARNPSEMYGR  
TEGMQZLCOWSKAH

(g) Image 4.1

HORSE  
PIG  
GOAT  
CHICK  
DUCK  
SHEEP  
COW  
DOG  
CAT

GOATPBN C  
JKISZRMA  
ESCODWHT  
DHLHPDHO  
IEVDICOW  
HEZOOHRG  
KPIGTISR  
WUXQCHEA  
DUCKFYMO  
GPSCHICK

(h) Image 4.2

SUMMERLH  
CIPORTNO  
BSUNBALL  
RELAXEPI  
TDAQSAND  
AYBCAZIA  
NFUNHRSY

TROPIC  
BEACH  
SUYMER  
HOLIDAY  
SAND  
BALL  
TAN  
RELAX  
SUN  
FUN

(i) Rotation Manuelle

FIGURE 2 – Résultats obtenus après pré-traitement

## 6 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)



## 6.1 Détection des contours par la méthode 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 ( $Gx$ ) et verticaux ( $Gy$ ) 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'hystérésis**. Deux seuils, `low_thresh` et `high_thresh`, sont définis 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.

## 6.2 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).

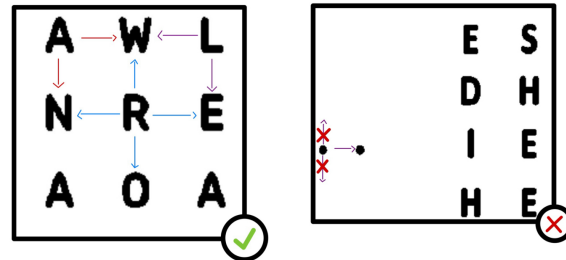


FIGURE 3 – Schéma modélisant le processus

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.

### 6.3 Détection des mots

La détection de mots suit le même principe que la détection de lettres. Nous utilisons toujours les boxes qui représentent chaque amas de pixels. Grâce à la fonction `merge_bounding_boxes`, qui permet de regrouper les boxes séparées par seulement quelques pixels et celles qui sont incluses dans une autre box, nous obtenons une box générale pour chaque mot. Les mots étant composés de plusieurs lettres séparées par des espaces bien plus petits que les espaces séparant chaque lettre de la grille, nous reprenons notre liste de boxes et enlevons toutes les boxes incluses dans la grille de lettres. Pour éviter de confondre certains amas de pixels qui ne sont ni des lettres ni des mots, nous ajoutons des conditions supplémentaires. Par exemple, nous vérifions que la hauteur de chaque box est inférieure à sa largeur et que des paramètres comme sa surface sont cohérents par rapport à la taille de l'image.

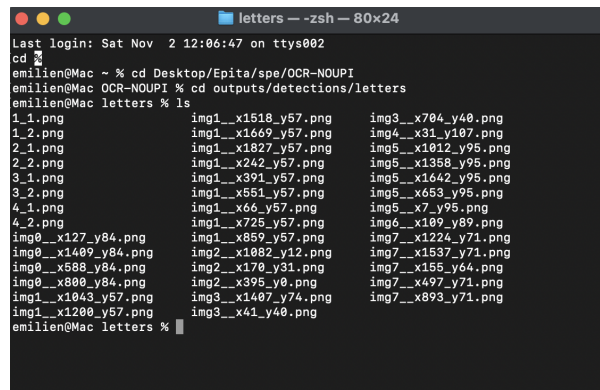
### 6.4 Détection des lettres des mots (et des lettres des grilles)

Pour détecter les lettres d'un mot, nous appelons l'algorithme de Canny avec l'image du mot (ou de la grille) en paramètre et une fonction qui contient les conditions que nous avons fixées pour détecter les lettres des mots (ou des grilles). Les résultats sont plutôt satisfaisants cependant, il subsiste quelques problèmes liés au fait que des lettres d'un mot peuvent se retrouver collées, à cause notamment des méthodes de prétraitement mentionnées précédemment.

### 6.5 Extractions

L'extraction des lettres est réalisée en identifiant et en isolant les zones de l'image originale qui sont délimitées par des rectangles rouges. Pour chaque rectangle détecté, le script extrait ce qui se trouve à l'intérieur en excluant les bordures rouges et le sauvegarde comme une nouvelle image qui a dans son nom la position en  $x$  et en  $y$ , ce qui nous permettra par la suite de reconstruire la grille tout en sachant à quel endroit va quelle lettre dans ladite grille. Les bordures rouges sont ensuite marquées en bleu pour

indiquer qu'elles ont été traitées. Cela nous permet ainsi d'isoler et d'extraire les lettres délimitées par des rectangles rouges dans l'image originale.

A terminal window titled 'letters --zsh-- 80x24' showing a series of commands and their outputs. The user navigates to a directory and lists files, which are organized into a grid of image filenames.

```
last login: Sat Nov  2 12:06:47 on ttys002
cd
emilien@Mac ~ % cd Desktop/Epita/spe/OCR-NOUPI
emilien@Mac OCR-NOUPI % cd outputs/detections/letters
emilien@Mac letters % ls
1_1.png      img1__x1518_y57.png  img3__x704_y40.png
1_2.png      img1__x1669_y57.png  img4__x31_y107.png
2_1.png      img1__x1827_y57.png  img5__x1012_y95.png
2_2.png      img1__x242_y57.png   img5__x1358_y95.png
3_1.png      img1__x391_y57.png   img5__x1642_y95.png
3_2.png      img1__x551_y57.png   img5__x653_y95.png
4_1.png      img1__x66_y57.png    img5__x7_y95.png
4_2.png      img1__x725_y57.png   img6__x109_y89.png
img0__x127_y84.png  img1__x859_y57.png   img7__x1224_y71.png
img0__x1409_y84.png img2__x1082_y12.png  img7__x1537_y71.png
img0__x588_y84.png  img2__x170_y31.png   img7__x155_y64.png
img0__x800_y84.png  img2__x395_y0.png    img7__x497_y71.png
img1__x1043_y57.png img3__x1407_y74.png   img7__x893_y71.png
img1__x1200_y57.png img3__x41_y40.png
emilien@Mac letters %
```

FIGURE 4 – Sauvegarde des lettres extraites

## 6.6 Fonctions utiles

Pour cette section aussi, nous avons implémenté toutes sortes de fonctions utiles, nous facilitant la vie pour manipuler la liste de boxes détectées par l'algorithme de Canny. Entre autres, une fonction qui efface une box sur l'image, une fonction qui permet de trier les boxes en fonction de certains champs des boxes ou encore des fonctions qui calculent la médiane, la moyenne ou même l'histogramme de la liste de boxes.

## 6.7 Résultats

Voici les résultats que nous obtenons une fois que nous avons effectué les détections :



(a) Image 1.1



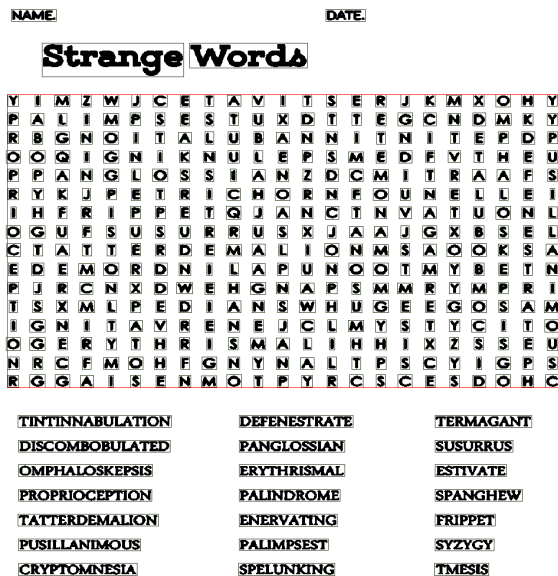
(b) Image 1.2



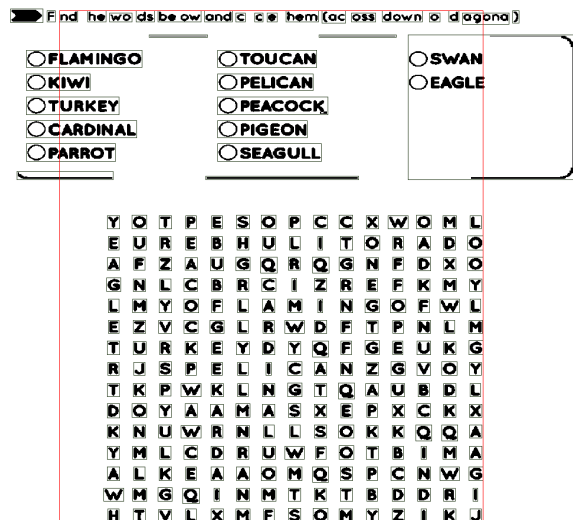
(c) Image 2.1



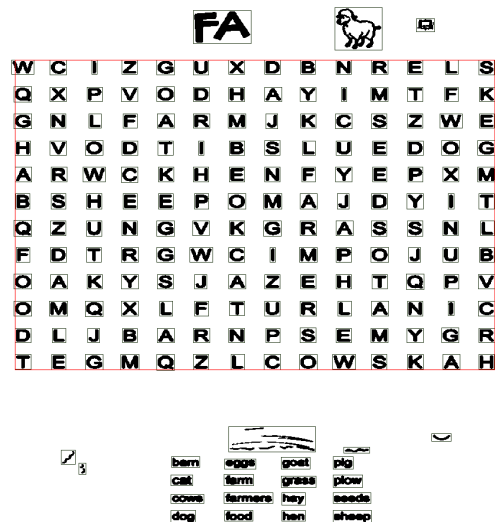
(d) Image 2.2



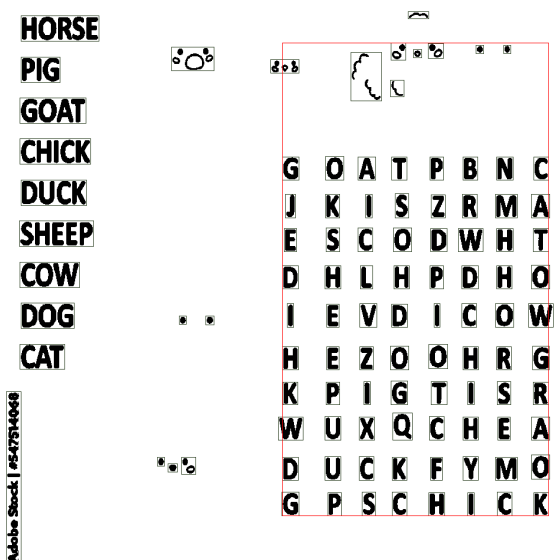
(e) Image 3.1



(f) Image 3.2



(g) Image 4.1



(h) Image 4.2

FIGURE 5 – Résultats obtenus pour les détections de grilles



(a) Image 1.1



(b) Image 1.2



(c) Image 2.1



(d) Image 2.2

**ESTIVATE**

(e) Image 3.1

**TOUCAN**

(f) Image 3.2

**sheep**

(g) Image 4.1

**CHICK**

(h) Image 4.2

FIGURE 6 – Résultats obtenus pour les détections des mots

**LIME**

(a) Image 1.1

**RESTING**

(b) Image 1.2

**MILK**

(c) Image 2.1

**SUN**

(d) Image 2.2

**ESTIVATE**

(e) Image 3.1

**TOUCAN**

(f) Image 3.2

**sheep**

(g) Image 4.1

**CHICK**

(h) Image 4.2

FIGURE 7 – Résultats obtenus pour les détections des lettres dans un mot

## 6.8 Interprétation des résultats

De manière générale, l'image 2.1 nous pose beaucoup de problèmes. Concernant la détection de grille, nous arrivons quasiment tout le temps à la détecter, en revanche, nous devons être plus précis dans certains cas. Il en va de même pour la détection de mots. Cependant, pour l'extraction de lettres, nous n'arrivons pas toujours à extraire chacune des lettres mais nous comprenons pourquoi et savons comment corriger le problème (nous nous en sommes rendu compte trop tard pour tout changer), le problème vient de l'implémentation de la méthode d'Otsu et de mauvaises valeurs utilisées pour l'augmentation de contrastes, collant des images entre elles, ce qui fait que le filtre de Canny ne reconnaît qu'une seule forme. Aussi, nous avons constaté que sur des mots écrits en petits, ces méthodes peuvent les rendre illisibles. Nous ferons le nécessaire pour que ces bugs soient corrigés pour la soutenance finale.

## 7 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)*

### 7.1 Structure globale

Dans ce projet, le réseau de neurones mis en place utilise une architecture multicouche, optimisée pour identifier des caractères. Cette structure est organisée autour de trois éléments principaux : une couche d'entrée, des couches intermédiaires et une couche de sortie.

**Couche d'entrée :** À ce niveau, une image de la lettre est reçue sous forme de matrice de pixels, chaque pixel étant converti en binaire (0 pour un pixel noir, 1 pour un blanc). En langage C, cela se traduit par un tableau multidimensionnel, chaque cellule représentant individuellement chaque pixel.

**Couches intermédiaires :** Ces couches, aussi appelées "cachées", sont essentielles pour repérer des motifs spécifiques, tels que des contours ou des lignes. Chaque neurone combine des poids et des valeurs d'entrée pour faire ressortir des informations essentielles de l'image. Ces calculs passent par des boucles imbriquées traversant chaque neurone pour calculer une somme pondérée, qui passe ensuite par une fonction d'activation, dans notre cas sigmoïde, définie par :

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

Cela permet de limiter les résultats entre 0 et 1, facilitant l'apprentissage des formes distinctives de chaque caractère.

**Couche de sortie :** Cette dernière couche produit la prédiction finale, c'est-à-dire la lettre supposée par le modèle. Elle utilise une fonction comme Softmax, qui convertit les résultats en une distribution de probabilité parmi toutes les lettres possibles, en sélectionnant ainsi la lettre la plus probable. La fonction Softmax est mathématiquement définie par :

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

où chaque sortie  $z_i$  est exponentiée avant d'être divisée par la somme de toutes les exponentielles des sorties.

## 7.2 XNOR

La version XNOR, conçue comme un proof of concept, diffère du réseau principal en deux points : elle n'a pas de couche cachée et utilise un nombre réduit de neurones en entrée et en sortie.

Structure simplifiée : Ce réseau utilise uniquement une couche d'entrée et une couche de sortie. La couche d'entrée prend deux bits et la couche de sortie génère une seule valeur, répondant à la table de vérité du XNOR.

Table de vérité XNOR : Le réseau prend en entrée deux valeurs binaires, et selon les règles XNOR, il doit retourner 1 si les deux valeurs d'entrée sont identiques (0-0 ou 1-1) et 0 sinon. Les poids sont programmés pour que le réseau apprenne cette logique sans couches cachées, en ajustant les connexions pour obtenir la sortie correcte selon chaque combinaison d'entrée.

Implémentation en C : En codant cette architecture en C, la structure simplifiée permet d'utiliser directement les opérations binaires sans nécessiter de transformations ou d'activations complexes. Chaque calcul suit simplement les règles du XNOR, ce qui valide que le réseau est capable de reproduire des fonctions logiques élémentaires avec précision, assurant ainsi la robustesse de l'architecture pour des tâches de base.

Cela montre que l'architecture, même sans couches cachées et avec une logique simple, peut effectuer des calculs binaires précis en suivant la table de vérité.

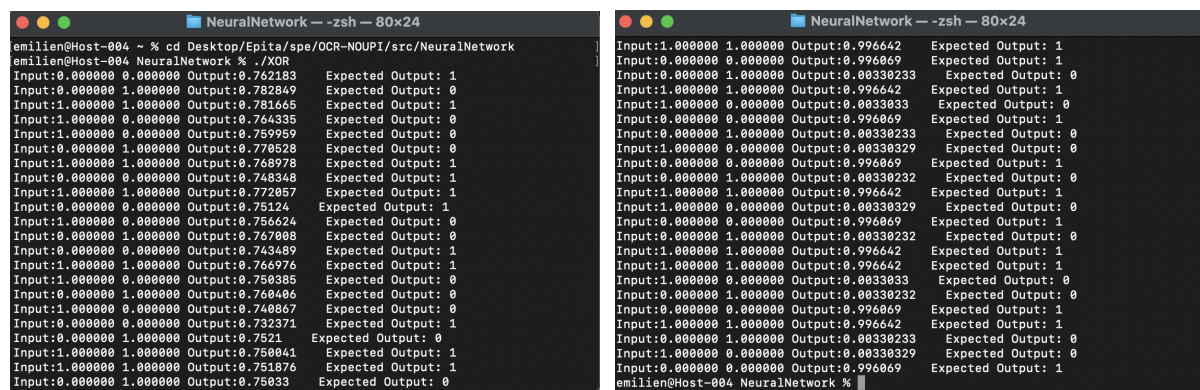


FIGURE 8 – Entraînement du réseau de neurones

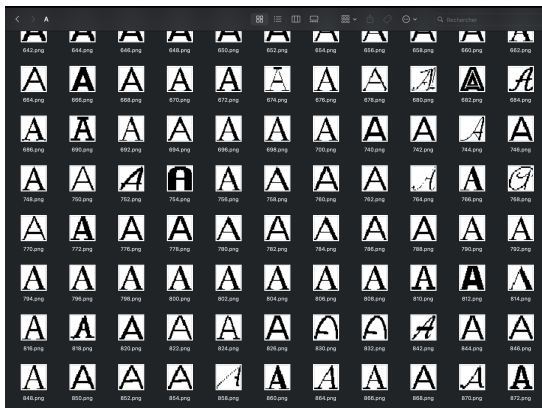


### 7.3 Données d'entraînement

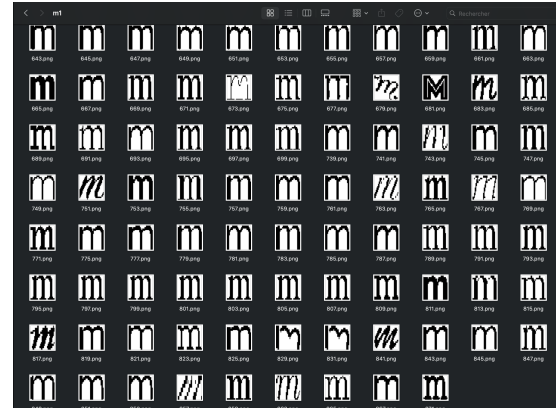
Pour nous constituer une banque de données suffisamment importante, nous avons récupéré toutes les polices qui étaient déjà préinstallées dans le système d'exploitation Mac OS, puis nous les avons tracées à l'aide d'un script Python. Ensuite, nous les avons rognées au maximum et redimensionnées dans 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.

```
dataset — zsh — 80x24
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 9 – Illustrations du dataset

## 8 Résolveur de mots croisés

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.

## 9 Avancée générale

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

Tâches	Avancement
<b>Gestion de l'image</b>	
Prétraitement	85%
Rotation de l'image	50%
Détections & extractions	65%
<b>Réseau de Neurones</b>	
XNOR	100%
Reconnaissance des lettres	80%
Données d'entraînement	100%
<b>Autre</b>	
Résolution de la grille	100%
Designs / UI	0%

TABLE 2 – Tableau d'avancement du projet

De manière générale, bien que tout ne soit pas parfait, nous n'estimons pas être en retard. Cependant, il va falloir fournir encore beaucoup d'efforts pour arriver à bout de ce projet. Les principaux problèmes que nous rencontrons actuellement pour la détection et le pré-traitement sont liés au fait que nous ne parvenons pas à trouver des paramètres qui marchent avec toutes les images. La plus contraignante étant l'image 2.1.

Pour récapituler, d'ici la prochaine soutenance nous devons :

- Corriger le prétraitement afin que les lettres ne soient plus collées. Mais surtout faire en sorte que l'image 2.1 soit exploitable.
- Implémenter la rotation automatique.
- Améliorer nos détections, en les rendant plus précises, cela viendra en grande partie à l'aide d'un pré-traitement encore meilleur.
- Reconstruire la grille dans un fichier texte à partir des lettres extraites. Nous devons donc lier notre réseau de neurones final qui détecte et reconnaît des lettres au reste du projet.
- Créer une fonction pour sauvegarder et charger les poids du réseau de neurones à partir d'un fichier texte. Pour ne pas avoir à constamment entraîner notre réseau de neurones. En effet, pour le moment nous ne sauvegardons pas les poids dans un fichier texte, ce qui veut dire qu'à chaque fois que l'on s'en sert, il faut relancer l'entraînement.
- Lier le solveur à la grille recrée dans un fichier texte.
- Retourner la grille avec les mots entourés.
- Réaliser entièrement l'interface utilisateur.

Nous n'aurons pas le temps de nous pencher sur les bonus qui consistent à créer un site internet et réaliser des grilles.

## 10 Conclusion

Pour conclure, nous sommes sur la bonne voie, notre avancement est légèrement en deçà des objectifs que nous nous étions fixés pour cette première soutenance au début du projet. Comme mentionné précédemment, il nous reste encore beaucoup d'efforts à fournir mais rien d'insurmontable. Pour la prochaine soutenance, notre objectif est de fournir un logiciel qui soit fonctionnel sur toutes les images de tests fournies dans le sujet, en reliant chaque partie que nous avons déjà implémentée et en y ajoutant celles que nous allons implémenter. En revanche, nous ne pensons pas que nous aurons le temps de faire le site internet ainsi que la génération de grilles.