OCR Rapport 1

Arthur WAMBST, Noé BELPERIN, Arthur VENDRAMINI, Sacha VAUDEY

Novembre 2024

Table des matières

1 Introduction

Ce document va présenter l'avancement du projet par ANSN Studio. Ce groupe est composé de 4 membres, Arthur WAMBST, Noé BELPERIN, Arthur VENDRAMINI et Sacha VAUDEY. Nous allons résumer l'ensemble du travail accompli sur ce projet jusqu'à cette première soutenance. Nous allons commencer par un court état de l'art pour nous rendre compte à quel point les OCR sont indispensables de nos jours. Puis, nous allons développer la répartition des tâches au sein de notre groupe. Ainsi, nous pourrons rentrer plus en détails sur nos choix techniques, les difficultés rencontrées ainsi que notre avancement à ce jour.

2 État de l'art

2.1 Utilisation des OCR aujourd'hui

Les OCR (reconnaissance optique de caractères) sont largement utilisés pour convertir des images de texte en données exploitables, facilitant l'automatisation et la numérisation. Ils sont utilisés dans de nombreux secteurs, comme la banque, pour lire des chèques ou des factures, et le e-commerce, pour extraire des informations des reçus. En entreprise, les OCR permettent de numériser des documents physiques et les rendre accessibles via des moteurs de recherche. Dans le domaine médical, ils aident à transformer les dossiers papier en données numériques. Aujourd'hui, les OCR sont devenus essentiels pour simplifier la gestion de l'information.

3 Le groupe ANSN Studios

3.1 Arthur WAMBST

Chef de projet, je suis heureux de la bonne entente dans le groupe. J'ai hâte de pouvoir appliquer ce que nous allons apprendre au cours de cette année, et je pense que nous allons faire du bon travail.

3.2 Noé BELPERIN

Aimant le domaine de l'informatique depuis très longtemps et plus particulièrement l'intelligence artificielle, j'ai le plaisir de gérer la totalité du réseau de neurone qui nous permettra de connaître chaque lettre. J'apprécie la collaboration avec mon groupe dans le but de mener à bien ce projet en réunissant les atouts de chacun, nous sommes un groupe soudé et à l'écoute les uns des autres.

3.3 Arthur VENDRAMINI

Je suis responsable du développement de l'interface utilisateur pour ce projet. L'objectif est de créer une interface moderne, qui permet de charger une image et de lancer le traitement sur cette dernière. Ce projet me permet d'élargir mes connaissances en C, en traitement d'image et en IA.

3.4 Sacha VAUDEY

Je suis passionné d'informatique et de nature très curieuse. Le projet OCR me permettra d'améliorer d'une part mes connaissances sur certains points de l'informatique que j'ignorais à savoir le traitement d'image et les prémices de l'intelligence artificielle. D'autre part, ce projet me permettra d'améliorer ma capacité de travailler en groupe

et de construire des projets. OCR étant un projet complexe et plus abstrait que le projet Unity de SUP, il m'est impératif de faire mon maximum pour comprendre et m'imprégner des notions qu'il aborde.

4 Répartition et planification des tâches

	Arthur V	**Arthur W**	**Noé**	**Sacha**
Banque d'images	X			
Détection de grille				X
Pré-traitements		X		
Détection de lettres				X
Solver			X	
Interface utilisateur	X			
Réseau de neurone			X	
Rotation d'image		X		

Table 1 – Répartition des tâches

	Avancement
Banque d'images	100%
Détection de grille	30%
Pré-traitements	100%
Détection de lettres	70%
Solver	100%
Interface utilisateur	50%
Réseau de neurone	50%
Rotation d'image	100%

Table 2 – Avancement des tâches

5 Choix techniques

5.1 Langage C : SDL et interface

5.1.1 Avantages

SDL est compatible avec de nombreuses plateformes (Windows, macOS, Linux, Android, iOS). Cela signifie que l'interface graphique fonctionnera de manière similaire sur tous ces systèmes sans qu'il soit nécessaire de réécrire le code pour chaque plateformes.
La librairie SDL est un excellent choix pour ce projet car elle offre la simplicité et la légèreté nécessaires, en restant suffisamment puissante pour créer une interface moderne et réactive pour un solveur de mots croisés.

5.2 Pré-traitement

Le pré-traitement a pour but de fournir à la suite du programme une image uniquement constituée de pixels noirs ou blancs, et constitue la base projet. En effet, si le pré-traitement de l'image fournit une image de mauvaise qualité (caractères effacés/non reconnaissables, bruit toujours présent...), cela va bloquer la suite du processus et empêcher la résolution de la grille.

5.2.1 Échelle de gris

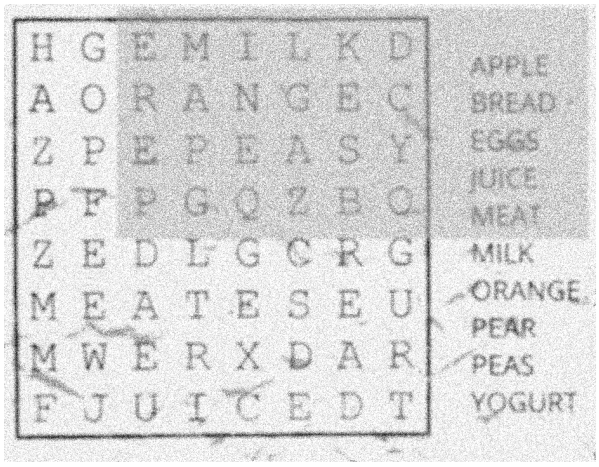
Dans un premier temps, il faut donc transformer l'image en échelles de gris. Cela consiste simplement à regarder les valeurs RGB de chaque pixel et appliquer la formule suivante :

$$0.299 * R + 0.587 * G + 0.114 * B$$

On obtient alors la valeur de gris du pixel, ce qui nous permet de transformer cette image en échelles de gris :

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

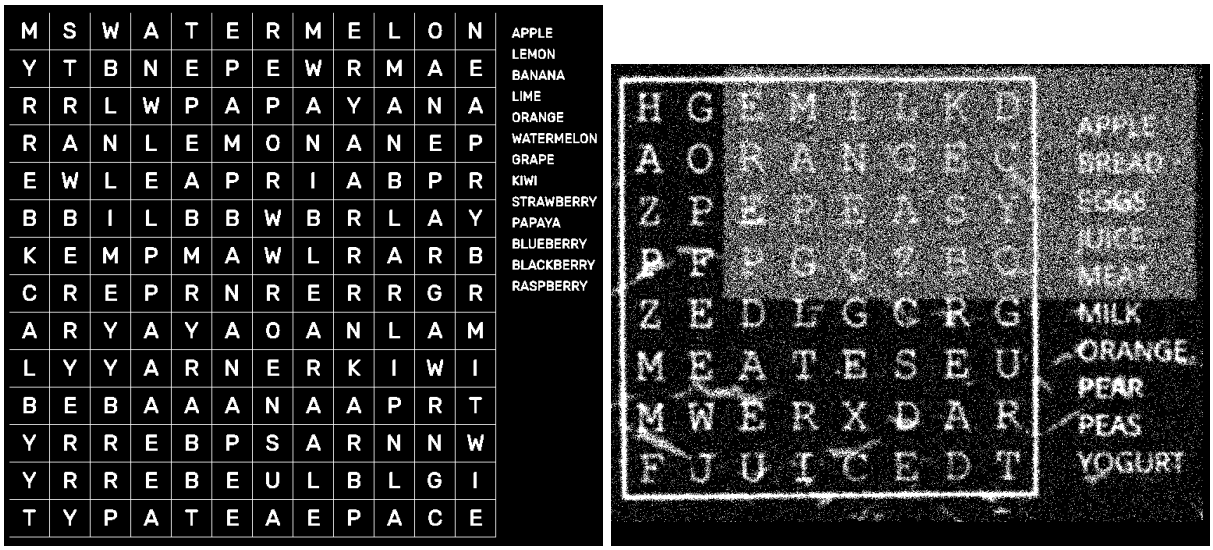
APPLE
LEMON
BANANA
LIME
ORANGE
WATERMELON
GRAPE
KIWI
STRAWBERRY
PAPAYA
BLUEBERRY
BLACKBERRY
RASPBERRY



Résultats après transformations

5.2.2 Noir et blanc

Il ne reste plus qu'a regarder la valeur de gris de chaque pixel :
Si celle-ci est supérieure à un palier, alors le pixel devient blanc, sinon il devient noir.
De manière générale, afin que le pré-traitement soit compatible avec toute image, on choisit ce palier en fonction de la moyenne des valeurs de l'image.
Ainsi, nous obtenons les images suivantes :



Résultats après transformations

5.2.3 Réduction du bruit

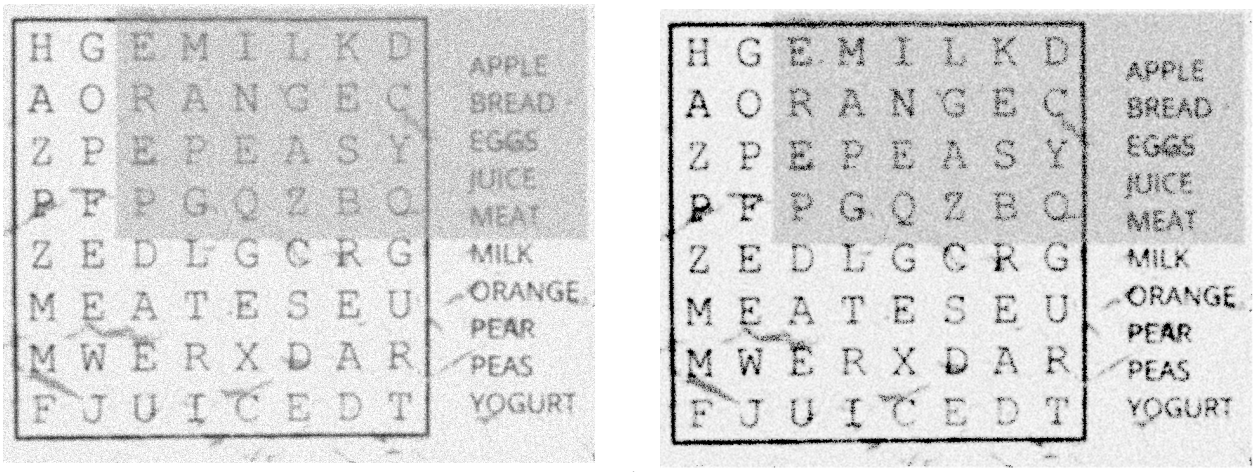
Cependant, une approche rudimentaire comme celle-ci ne fonctionnera pas sur toutes les images. Par exemple, l'image de droite ci-dessus comporte trop de résidus qui vont empêcher la reconnaissance des caractères.

On doit alors appliquer différents algorithmes qui vont avoir pour but de supprimer le bruit ou les résidus de l'image.

Débruitage par patchs

Cet algorithme va consister à faire, pour chaque pixel, la moyenne des pixel alentours, et lui donner cette valeur. Cet algorithme est efficace surtout contre les images contenant des résidus dont les pixels ne sont pas tout à fait collés.

Cela va faire ressortir nettement les pixels avec des voisins, soit les traits continus, et donc les lettres qui nous intéressent :

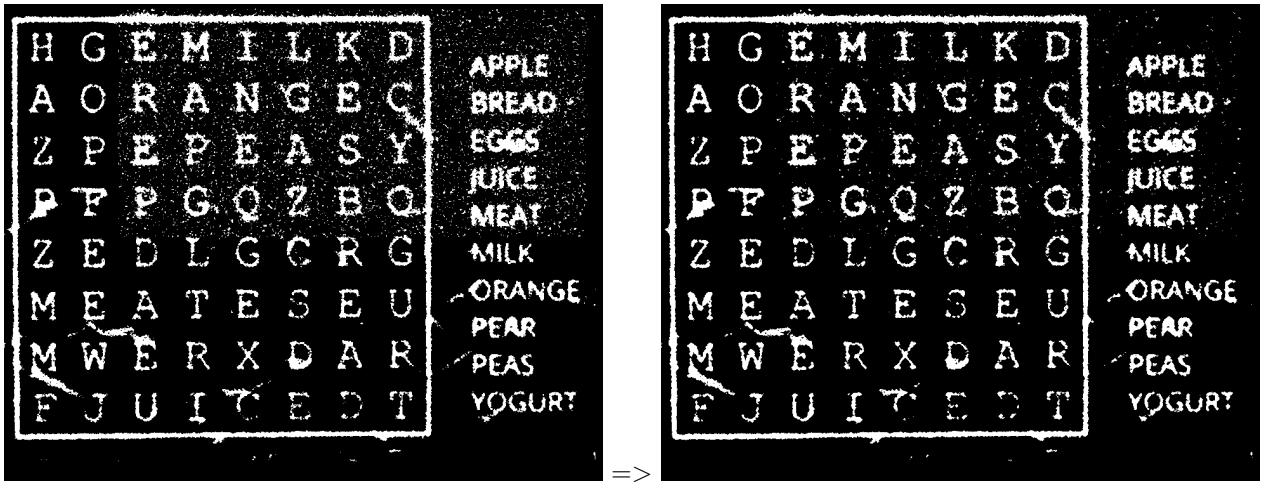


Résultats après transformations

Débruitage par voisins

A nouveau pour chaque pixel, cet algorithme va tout simplement regarder s'il a bien au moins un voisin, sinon le pixel va devenir noir.

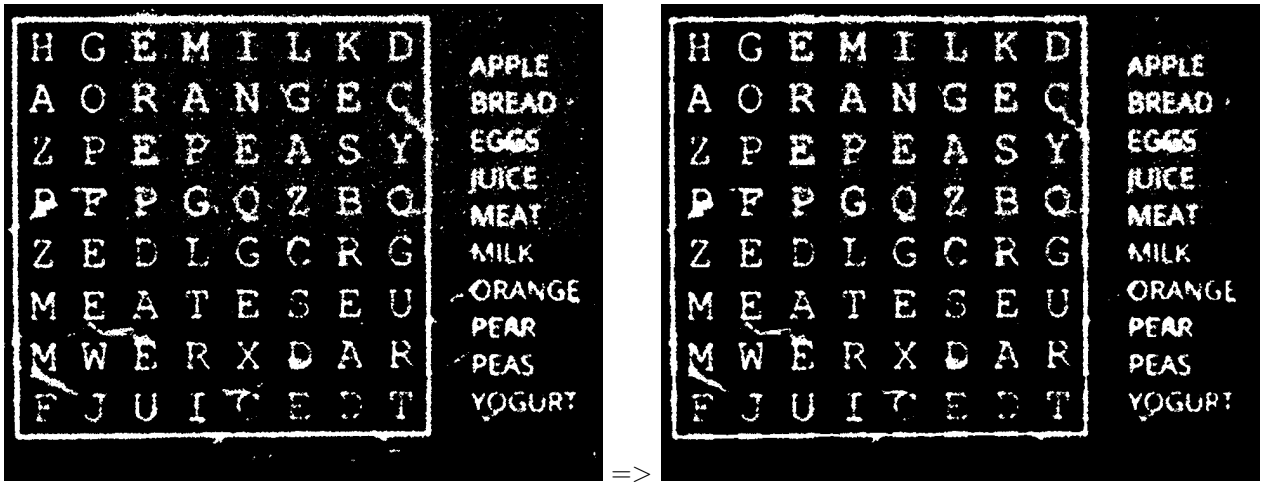
Cela va éliminer les pixels isolés, qui en général ne sont que des résidus :



Résultats après transformations

Débruitage ligne/colonne

Toujours pour chaque pixel, on va cette fois regarder la valeur moyenne des lignes et colonne de la surface et éliminer les pixels ayant peu de voisins sur des lignes/colonnes possédant aussi peu de pixels blancs.
Cela va éliminer les pixels qui ne font de manière générale pas partie de groupement de pixels, qui en sont eux aussi en général que des résidus :

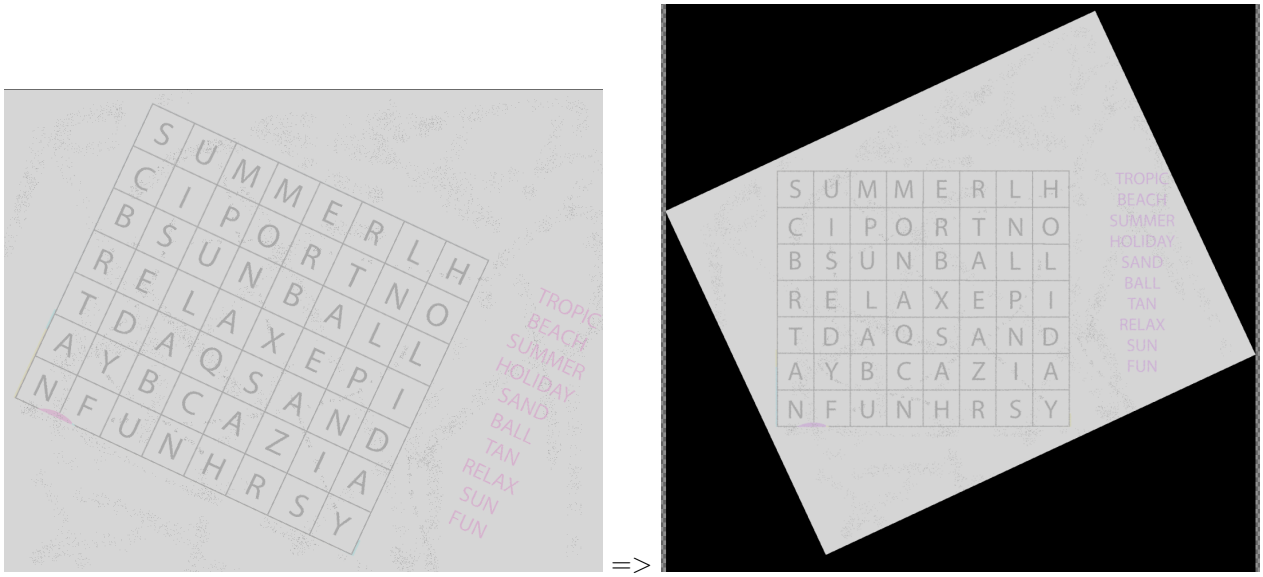


Résultats après transformations

5.2.4 Rotation d'images

La dernière partie du pré-traitement d'image est la rotation. Pour pouvoir faire la rotation sans perdre ni ajouter de pixels, on va procéder avec un raisonnement pouvant paraître un peu contre intuitif.
En effet, il faut dans un premier temps, calculer la position et future taille de l'image et des pixels, puis, pour chaque pixel dans la nouvelle image, calculer sa position dans l'image de départ et prendre sa couleur si elle existe.

Ainsi, en précisant l angle manuellement, on obtient une rotation d'image propre :



On peut alors s'attaquer aux détections sur l'image !

5.3 Détection des caractères

Les algorithmes de détection sont la deuxième étape après la réalisation de pré-traitement sur l'image. En effet, les fonctions et filtres appliqués précédemment ont pour but d'améliorer les performances des algorithmes de détection qui vont être explicités dans un instant. Une difficulté peut déjà être observée dans ce processus de détection puisque les grilles de mots possèdent, pour très peu d'entre-elles, des lignes délimitant les lettres entre elles. De ce fait, notre algorithme de détection ne pourra se reposer sur aucune méthode de détection de ligne mais sur une détection pure de contours et autres types de détection pour connaître l'emplacement précis des lettres de notre grille.

5.3.1 Algorithme de Canny

Dans le traitement d'image, l'algorithme de Canny est très connu. En effet, il s'agit d'un ensemble de méthodes utilisées pour réaliser de la détection de contours. C'est cette méthode que nous allons utiliser pour détecter nos lettres. Comme indiqué juste avant, Canny utilise diverses techniques afin de pouvoir détecter correctement les contours.

Dans la suite de ce rapport, nous ne travaillerons plus sur des images en tant que telles, constituées de pixels, mais sur une représentation binaire de celles-ci, obtenue facilement après une conversion des pixels noirs en 1 et des pixels blancs en 0. Cette représentation binaire sera la plus adaptée pour l'application des filtres détaillés ci-après.

Filtre de Sobel La première de ces méthodes et d'utiliser le filtre de Sobel (ou filtre de gradient). Ce filtre calcule pour chacun des pixels de notre image un gradient d'intensité. Celui-ci indique la direction de la plus forte variation de couleur (entre une couleur claire et une couleur foncée) depuis ce pixel. C'est la que réside l'intérêt du pré-traitement puisque cet **algorithme ne peut fonctionner que sur une image en noir et blanc.**

Le filtre de Sobel utilise deux matrices (ou masques) initialisées en début de programme aux valeurs suivantes :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix} G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Suite à cela, on parcourt l'ensemble des pixels de notre image en appliquant une convolution. Pour faire simple, il s'agit de prendre les pixels voisins à notre pixel actuel (dans une fenêtre 3x3) et de multiplier cette matrice créée par notre masque. Ainsi, la multiplication par la matrice Gx donne le gradient dans le sens horizontal, tandis que la multiplication par la matrice Gy nous donne le gradient dans le sens vertical.

Enfin, l'intensité du contour d'un pixel est calculée de la manière suivante :

$$\sqrt{gx\check{s} + gy\check{s}}$$

Suppression des non-maximum

Le filtre de non-maximum est appliqué à la suite du filtre de Sobel. En réalité, ce filtre est plus une continuité du filtre de Sobel plutôt qu'un filtre à part entière.

Ce filtre permet en effet de ne conserver que les pixels dont le gradient d'intensité est un maximum local. Cette méthode permet également de réduire le bruit (suppression des gradients d'intensité inutiles ou négligeables par rapport à d'autres) et d'obtenir ainsi des contours plus propres, plus facilement utilisables pour la suite de notre détection.

D'un point de vue technique, pour chaque pixel, le programme va regarder dans la direction du gradient du pixel sur lequel on se trouve et identifier si le pixel vers lequel on pointe a une valeur de gradient supérieure ou inférieure à celle du pixel actuel. Suite à cela, le principe expliqué plus haut est appliqué.

Filtre d'Hystérésis

Après l'étape du filtre de Sobel, le filtre d'Hystérésis constitue sans doute un point central dans le processus de détection d'image et de contours.

Le filtre d'Hystérésis a une méthode simple : classer l'ensemble des pixels de notre image selon trois catégories : les pixels forts, les pixels faibles et les pixels sans intérêt.

D'un point de vue technique, notre fonction prend en paramètre notre image, mais aussi deux valeurs qui vont constituer notre seuil haut et notre seuil bas pour cette fonction. Concrètement, si la valeur de gradient d'un pixel est supérieure au seuil haut, il sera considéré comme fort. Si la valeur de son gradient se situe entre le seuil haut et le seuil bas, le pixel sera considéré comme faible. Sinon, il sera considéré comme sans importance.

À la suite de la première itération réalisée sur l'image, une nouvelle itération va être effectuée sur l'ensemble des pixels qui ont été considérés comme faibles. Si un des pixels est entouré d'au moins un pixel fort (dans n'importe laquelle des 8 directions autour de ce pixel), il sera alors converti en pixel fort. Cette opération permet de fiabiliser le processus et ainsi de renforcer les contours (symbolisés par les pixels forts).

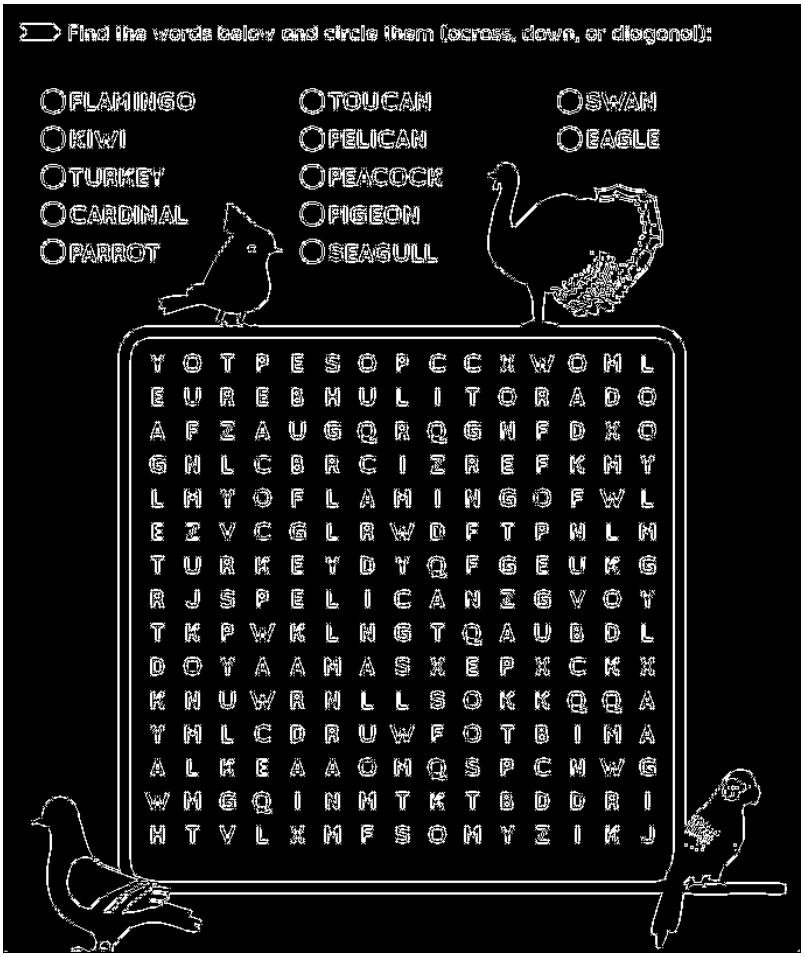
Une dernière itération est réalisée sur notre image. Cette itération va permettre de "nettoyer" celle-ci en supprimant tous les pixels qui ne sont pas considérés comme forts. Ainsi, après cette dernière itération, notre image ne contiendra que les véritables contours qui nous intéressent.

Principe de dilatation

Une fois nos filtres appliqués à notre image, nous allons appliquer à celle-ci le principe de dilatation. Ce principe est assez simple à comprendre puisqu'il permet d'augmenter la taille des objets (ici des contours) présents sur notre image.

En termes techniques, nous allons paramétrer un coefficient de dilatation (coefficient 2 dans notre code), puis itérer sur notre image en appliquant ce coefficient à tous nos pixels. Ainsi, nos contours seront intensifiés, proportionnellement au facteur de dilatation. Cela permettra par la suite de rendre notre détection de lettres plus efficace.

Il s'agit du dernier filtre appliqué à notre image pour le processus de Canny. Nous obtenons ainsi une image comme celle-ci :



5.3.2 Détection des lettres

Après l'application du filtre Canny, il nous faut implémenter l'ensemble des fonctions qui nous permettrons de réaliser la détéction pure de notre lettres de notre grille.

Pour cela, nous avons appliqué une fonction qui nous permet d'entourer nos lettres fonctionnant de la manière suivante :

Détection des arrêtes

Notre programme itère à travers notre image. Dès que le pixel rencontré est de couleur blanche (code RGB de 255 pour les trois composantes) et que ce pixel n'a pas encore été traité, le programme génère alors une nouvelle boîte. Nous appelons sur cette nouvelle boîte un algorithme de remplissage par diffusion afin de détecter la taille générale de cette boîte.

Avant de poursuivre, nous allons expliciter le principe de l'algorithme de remplissage par diffusion.

Remplissage par diffusion

Le remplissage par diffusion est un principe de remplissage de pixels auquel on a souvent été confronté. En effet, lorsque l'on utilise un éditeur de dessin en ligne, c'est cet algorithme qui est appliqué dès lors que l'on souhaite remplir une forme de notre dessin avec la fonction "pot de peinture".

Techniquement, cet algorithme est assez simple dans sa conception et sa réalisation. En effet, le pixel à partir duquel nous allons commencer notre programme est ajouté dans une file. Pour chaque pixel défilé, nous allons ajouter les pixels voisins de couleur blanche à notre file et considérer que ce pixel défilé est traité.

En répétant ce processus jusqu'à ce que la file soit vide, cet algorithme renvoie les valeurs des positions des pixels extrêmes ayant été traités et étiquetés.

Notre fonction recevra donc ces valeurs de position de pixel et tracera alors un rectangle selon ces coordonnées. Ce rectangle contiendra alors les lettres détectées.

Améliorations de résultats

Bien que ces fonctions apportent déjà des résultats satisfaisants, diverses conditions vont tout de même encore être ajoutées afin de finaliser le fonctionnement du programme. L'ensemble de ces conditions est défini ci-dessous :

- Une boîte englobante ne pourra être créée que si sa surface est supérieure à 10 pixels. Dans le cas contraire, sa création sera ignorée.
- Dans le même principe, une boîte englobante d'une surface supérieure à 100 pixels sera également ignorée.
- Bien que cela n'ait pas été réalisé, on pourrait penser qu'une condition pour la création d'une boîte englobante nécessite qu'elle ne se superpose pas à une boîte déjà existante.

Enregistrement des lettres

Une fois que les lettres ont été détectées, il devient alors assez simple d'enregistrer pour chaque rectangle de notre image, une image au format PNG. Ces images seront alors utiles pour être analysées par le réseau de neurones afin que notre programme puisse identifier la lettre détectée.

5.4 Détection de grilles

Une fois que les lettres ont été détectées, nous pouvons alors nous concentrer sur la détection de la grille et des mots à trouver dans celle-ci.

5.4.1 Détection de la grille

La détection de grille se base sur la liste de l'ensemble des boîtes englobantes déterminées grâce au processus de détection précédent.

Une itération est réalisée sur ces boîtes afin de déterminer l'espacement entre chacune d'entre elles. Si l'espacement entre deux boîtes est inférieur (ou égal) à l'espacement admissible maximum (défini par une constante dans notre programme), mais aussi supérieur à la distance minimale admissible (également définie par une constante dans notre programme), alors cette boîte est ajoutée à la liste des boîtes pouvant constituer notre grille de jeux.

5.4.2 Détection des mots

Dans le même principe, une itération va être réalisée sur les boîtes restantes et déterminer si leur espacement est inférieur au seuil minimal. Dans ce cas, toutes les boîtes ayant une coordonnée en y équivalente et qui respectent cette condition d'espacement seront considérées comme faisant partie du même mot.

5.4.3 Information complémentaire

Bien que la théorie de la détection de grille et de mots ait été préconisée ici, elle n'a pas pu être réalisée étant donné que la détection de lettres n'est pas pleinement fonctionnelle. Mais une implémentation de la détection de grille a pu tout de même être faite.

5.5 Réseau de neurone

5.5.1 Propagation

Le front propagation utilise la fonction d'activation sigmoïde pour prédire l'output : elle se caractérise par :

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Le back propagation utilise la fonction dérivée de sigmoïde pour mieux détecter la valeur de l'erreur. Elle se caractérise par :

$$f(x) = x * (1 - x) \tag{2}$$

5.5.2 XOR

La fonction Xor va fonctionner de la même manière que notre réseau de neurone principal, à la petite différence que l'input sera entré manuellement. Le tableau de cette fonction peut se caractériser par :

a	b	$\overline{a \oplus b}$
0	0	1
1	0	0
0	1	0
1	1	1

TABLE 3 – Fonction $\overline{A \oplus B}$

La fonction Xor étant très simple car elle ne possède que deux entrées, va s'approcher très rapidement d'un taux d'erreur quasi nul. Voici des exemples d'outputs montrant l'apprentissage :

```
Input: (1,0), Predicted output: 0, Output: 0.498473
Input: (1,1), Predicted output: 1, Output: 0.462749
Input: (0,0), Predicted output: 1, Output: 0.513647
Input: (0,1), Predicted output: 0, Output: 0.567237
```

FIGURE 1 – 100 tests

```
Input: (1,1), Predicted output: 1, Output: 0.666296
Input: (0,1), Predicted output: 0, Output: 0.117172
Input: (1,0), Predicted output: 0, Output: 0.596044
Input: (0,0), Predicted output: 1, Output: 0.661233
```

FIGURE 2 – 2000 tests

```
Input: (1,1), Predicted output: 1, Output: 0.998734
Input: (0,0), Predicted output: 1, Output: 0.998477
Input: (1,0), Predicted output: 0, Output: 0.00203488
Input: (0,1), Predicted output: 0, Output: 0.00136715
```

FIGURE 3 – 10000 tests

5.5.3 Traitements des images

Pour que le réseau de neurone soit fonctionnel, il a besoin d'être entraîné de manière bien précise. Il va prendre en entrée une image qu'il va devoir reconnaître. Les caractères reconnus seront obligatoirement les 26 lettres de l'alphabet majuscules ainsi que minuscules. Pour cela, il doit convertir chaque image en chaîne binaire pour que le réseau fonctionne correctement. Chaque image entrée en paramètre sera automatiquement redimensionnée en 30x30 pixels. Ainsi, une fonction de traitements d'images va transformer cette nouvelle image en tableau de 900 valeurs qui varient entre 0 et 1 en fonction de la couleur du pixels.

5.5.4 Entraînements

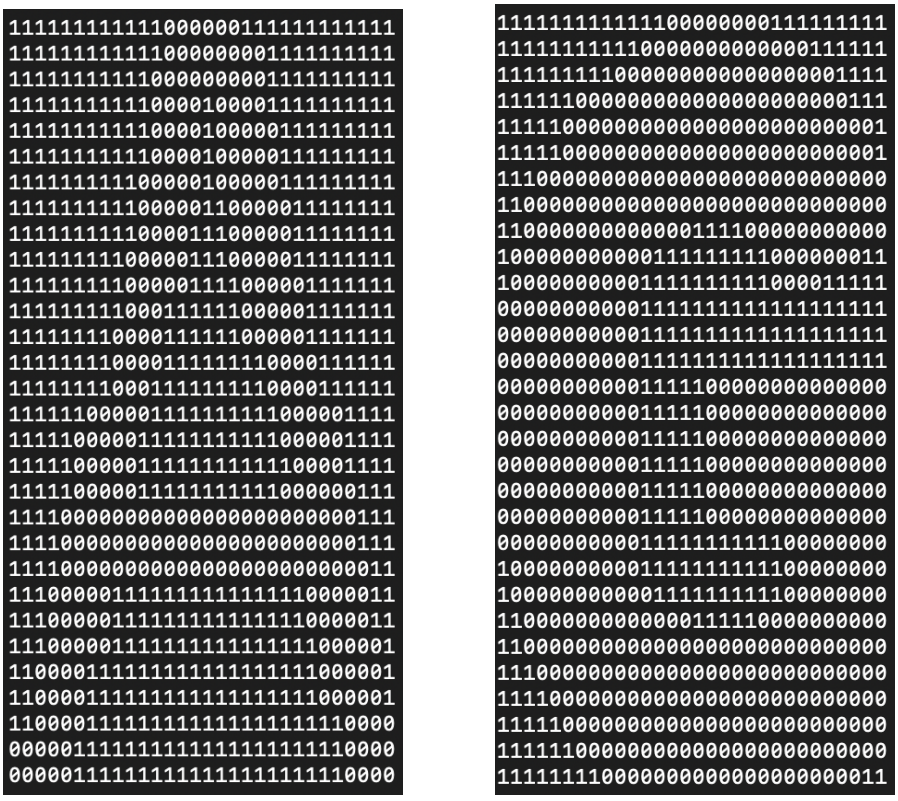
Lors de l'entraînement, ce réseau de neurone va fonctionner de la même manière que celui du Xor. En effet elle utilise la fonction sigmoïde, sa dérivée, le back propagation et le front propagation. La différence entre les deux réside dans le fait qu'elle prend en entrée un tableau de 900 valeurs et qu'il y a 52 outputs différents pour les 26 lettres de l'alphabet majuscules et minuscules. De plus, nous venons rajouter la fonction softmax sur la dernière couche ainsi que la fonction shuffle pour améliorer la précision de ce réseau. Actuellement, le réseau fonctionne avec un taux de réussite de 70% pour 20 polices d'écriture différentes.

5.5.5 Sauvegarde et prédiction

Pour optimiser au maximum le réseau de neurone, nous avons choisi de l'exécuter une fois avec le nombre de polices différentes que l'on veut tester puis de sauvegarder les données qui nous seront utiles dans un dossier. Ainsi, des lors que nous voudrons savoir à quelle lettre correspond l'image qui nous entrerons, il suffira de récupérer ces valeurs dans le fichier et de chercher la lettre qui aura la plus grande probabilité d'être la même que celle entrée en paramètre.

5.6 Solver

Le script du solver est quant à lui assez simple, nous avons opté pour la méthode de force brute. En effet, lorsqu'il recherche un mot, il parcourt la première lettre dans le tableau jusqu'à la trouver. Si c'est le cas, il va chercher dans toutes les diagonales, les sens horizontaux et verticaux. Il effectue ces tests uniquement si le nombre de lettres du mot est inférieur au nombre de lettres restantes dans la ligne à tester. Il affiche la position dans la grille de la première et dernière lettre du mot s'il est trouvé, "Not Found" sinon. Il prend en paramètre une fichier avec les lettres que contiennent initialement la grille.



(a) Représentation du A

(b) Représentation du G

6 Obstacles rencontrés

6.1 Pré-traitement

Les images du niveau 2 ont clairement été les images qui ont posé le plus de problème. Le fait que l'image 1 ait énormément de résidus force le pré-traitement à être qualitatif, mais cela a posé beaucoup de difficultés car cela mène parfois à l'effacement de caractères. Aussi, pour l'image 2.2, la rotation à été compliqué, mais a été assez intéressante pour trouver la bonne logique et pouvoir tourner une image sans perdre de pixels.

6.2 Détection de la grille et des caractères

Il s'agit sûrement de l'une des parties les plus complexes du projet OCR. En effet, la détection de lettres repose sur des algorithmes complexes de détection de contours dans lesquels de multiples filtres doivent être appliqués sur notre image pour en extraire les caractéristiques importantes et nécessaires à la construction de la reconnaissance de lettres. De plus, l'ensemble de ces filtres repose majoritairement sur des paramètres à modifier légèrement pour adapter le résultat. De plus, trouver le réglage complexe s'avère assez difficile. Les problèmes rencontrés ont notamment été une mauvaise détection des lettres, des box trop grandes ou trop petites, voir même entourant des éléments incohérents de l'image.

La détection de la grille est la aussi une partie complexe. En effet, elle ne peut être envisagée que si la position

des lettres a été repérée au préalable. C'est la position des lettres qui va en effet définir la position de la grille et des mots à trouver. La difficulté ici est d'arriver à avoir une détection de lettre efficace pour que la grille puisse correctement être identifiée.

6.3 Réseau de neurones

Différents obstacles ont pu être rencontrés lors de la création du réseau de neurone. En effet, trouver les valeurs pour atteindre un assez grand taux de précision ou encore ne pas avoir une complexité trop élevée n'a pas été une tâche facile. De plus, la gestion de mémoire durant l'exécution du programme a été l'obstacle le plus important à mon goût.

6.4 Solver

Peu de problèmes ont été rencontrés lors de la création du solver de par son implémentation assez simple, excepté la gestion de mémoire et les possibles segmentions fault dûs à la manipulation de tableau dynamique.

7 Avancée du projet

7.1 Pré-traitement

L'état du pré-traitement est très avancé, voire presque terminé.
Nous ne prévoyons plus de travailler activement, ou pas plus loin que de fix des bugs si nécessaire.

7.2 Détection de la grille et des caractères

En ce qui concerne la détection de caractères, l'algorithme générale est implémentée. En revanche, après son exécution, on peut constater que son efficacité n'est pas encore optimale. Il s'avère que des petits éléments de bruit ou encore des éléments non lettres sont détectés alors qu'il ne faudrait pas.

En ce qui concerne la détection de grille, elle a été implémentée de manière théorique mais non fonctionnelle. En effet, la détection de lettres n'étant elle-même pas pleinement fonctionnelle, il nous est impossible de tester efficacement la détection de grille et de mots associés.

7.3 Réseau de neurones

Le réseau de neurone peut actuellement reconnaître entre 70% et 80% des caractères correctement. En effet, il faudra l'exécuter avec un plus grand nombre de nombres de polices différentes pour améliorer sa précision. Actuellement, il est exécuté avec 21 polices majuscules et minuscules différentes. Résoudre les problèmes de l'optimisation et d'une meilleure gestion de mémoire aideront sans hésiter à obtenir une meilleure précision. De plus, les résultats du réseau de neurone sont stockés dans 4 fichiers différents et peuvent être sans problème réutilisés lorsque nous souhaitons connaître le caractère présent sur une image. Il reste à améliorer la précision, ainsi que créer la passerelles entre le réseau de neurones et le solver pour permettre de clôturer cette partie.

7.4 Solver

Actuellement, notre solver est capable de trouver un mot entré en paramètre dans une grille qui se caractérise par un fichier créé à la main. Il est efficace à 100% sur tous les essais actuels mais certains problèmes pourront survenir lorsque nous inclurons de nouvelles grilles. Le prochain objectif sera de construire la grille contenue dans

le fichier grâce à notre réseau de neurone pour ensuite résoudre la totalité des mots voulus. Nous devrons alors récupérer les coordonnées de ces mots pour les rendre visibles à l'utilisateur.

8 Conclusion

En conclusion, notre équipe progresse de manière organisée et efficace, avec une répartition des tâches bien définie. À ce jour, la majorité des tâches ont été réalisées dans les délais, témoignant de l'engagement et de la rigueur de chaque membre. Nous avons su maintenir une bonne communication et une forte cohésion au sein du groupe, ce qui a permis de résoudre les éventuels obstacles rapidement et de maintenir notre productivité. Chaque membre apporte sa contribution en respectant les objectifs fixés et l'avancée du projet se déroule comme prévu.

Ainsi, une fois que chacun aura terminé les tâches qui lui sont attribuées, il restera uniquement à consolider nos différentes parties et à harmoniser l'ensemble pour finaliser le projet. Cette phase d'intégration permettra de garantir la cohérence et la fluidité de notre travail collectif. Nous pouvons donc conclure que le projet avance selon les prévisions, avec une équipe impliquée et des résultats prometteurs.