

---

# OCR Word Search Solver

---

Alexandre Joaquim Lima Salgueiro    Hugo Guyennet  
alexandre-joaquim.lima-salgueiro    hugo.guyennet

Léa-Angélina Kolmerschlag    Tom Huynh  
lea-angelina.kolmerschlag    tom.huynh

4 novembre 2025

## Sommaire

<b>1 Rôle et mission de chaque membre</b>	<b>2</b>
<b>2 Objectifs, planification et état d'avancement</b>	<b>2</b>
2.1 Objectifs du projet . . . . .	2
2.2 Planification et état d'avancement . . . . .	3
2.3 Note sur la reconnaissance de caractères (OCR) . . . . .	3
<b>3 Extraction de la grille et des mots</b>	<b>4</b>
3.1 Chargement des images . . . . .	4
3.2 Extraction de la grille . . . . .	4
3.2.1 Transformation en niveaux de gris . . . . .	4
3.3 Débruitage adaptatif . . . . .	5
3.4 Seuil adaptatif . . . . .	5
3.4.1 Méthode de la moyenne . . . . .	6
3.4.2 Méthode Gaussienne . . . . .	6
3.4.3 Application du seuil . . . . .	6
3.5 Détection de la grille . . . . .	6
3.5.1 1. Recherche des contours . . . . .	6
3.5.2 2. Identification du plus grand contour . . . . .	7
3.5.3 3. Calcul de la boîte englobante (Bounding Box) . . . . .	7
3.5.4 4. Extraction de la grille . . . . .	7
3.6 Nettoyage morphologique de la grille . . . . .	7
3.7 Analyse de la Structure de la Grille . . . . .	7
3.7.1 Cas 1 : Grille avec lignes internes (Détection par Lignes) . . . . .	8
3.7.2 Cas 2 : Grille sans lignes internes (Détection par Lettres) . . . . .	9
3.7.3 Résultat : La grille segmentée . . . . .	10
<b>4 Extraction de la Liste de Mots</b>	<b>10</b>
4.1 Pré-traitement et Binarisation . . . . .	10
4.2 Analyse et Groupement des Mots . . . . .	12
<b>5 Réseau de Neurones pour la fonction XNOR</b>	<b>14</b>
5.1 Architecture du réseau . . . . .	14
5.2 Apprentissage par Rétropropagation . . . . .	14
5.3 Résultats et Performance . . . . .	15
<b>6 Solver</b>	<b>15</b>
6.1 Fichiers utilisés . . . . .	16
6.2 main.c . . . . .	16
6.3 solver.c . . . . .	16

# 1 Rôle et mission de chaque membre

- **Chargement des images et mise à l'échelle de gris :** Léa-Angélina Kolmerschlag
- **Rotation de l'image :** Léa-Angélina Kolmerschlag
- **Pré-traitement de l'image :** Tom Huynh
- **Extraction de la grille et des cellules :** Tom Huynh
- **Extraction des mots :** Tom Huynh et Alexandre Joaquim Lima Salgueiro
- **Réseau de neurones :** Tom Huynh et Hugo Guyennet
- **Solver :** Alexandre Joaquim Lima Salgueiro

# 2 Objectifs, planification et état d'avancement

## 2.1 Objectifs du projet

L'objectif principal du projet était de concevoir et d'implémenter un système complet capable de résoudre des grilles de mots mêlés à partir d'une simple image.

### Éléments pour la première soutenance

- ✓ Chargement d'une image et suppression des couleurs ;
- ✓ Rotation manuelle de l'image ;
- ✓ Détection de la position :
  - De la grille ;
  - De la liste de mots ;
  - Des lettres dans la grille ;
  - Des mots de la liste ;
  - Des lettres dans les mots de la liste.
- ✓ Découpage de l'image (sauvegarde de chaque lettre sous la forme d'une image) ;
- ✓ Implémentation de l'algorithme de résolution d'une grille de mots cachés dans le programme en ligne de commande `solver` ;
- ✓ Une preuve de concept du réseau de neurones, capable d'apprendre la fonction logique XNOR.

### Éléments pour la soutenance finale

- ⇒ Le prétraitement complet (avec la rotation automatique) ;
- ⇒ Le réseau de neurones complet et fonctionnel :
  - Apprentissage ;
  - Reconnaissance des lettres de la grille et de la liste de mots.
- ✓ La reconstruction de la grille ;
- ✓ La reconstruction de la liste de mots ;
- ✗ La résolution de la grille ;
- ✗ L'affichage de la grille ;
- ✗ La sauvegarde du résultat ;
- ⇒ Une interface graphique permettant d'utiliser tous ces éléments.

## 2.2 Planification et état d'avancement

Le projet a été découpé en plusieurs phases successives, de la manipulation d'image basique à la résolution finale. Une planification initiale a été établie pour suivre la progression des différentes tâches.

Le projet respecte les délais prévus et présente même une avance sur la planification initiale. Le pipeline fonctionnel est capable de traiter avec succès les images de **niveau 3**, qui représentent les cas les plus complexes fournis (images bruitées, tournées, avec des conditions d'éclairage non-uniformes).

## 2.3 Note sur la reconnaissance de caractères (OCR)

Dans le cadre de ce projet, un réseau de neurones convolutionnel (CNN) a été développé en interne pour la reconnaissance optique des caractères (OCR). Ce réseau, bien que non présenté dans ce rapport, atteint une précision satisfaisante pour la reconnaissance des lettres dans les cellules extraites de la grille.

Le choix de ne pas détailler cette partie dans le présent rapport est motivé par la volonté de se concentrer sur les aspects de traitement d'image et d'analyse de structure, qui constituent le cœur de notre contribution. Le réseau XNOR présenté précédemment sert d'illustration des principes de base des réseaux de neurones, tandis que notre CNN d'OCR est une implémentation plus complexe et spécifique.

### 3 Extraction de la grille et des mots

Le but de l'extraction de la grille et des mots et de pouvoir donner les informations extraites au solver. Ce processus peut être vu comme une application d'algorithme de computer vision [8].

Nous prendrons comme exemples ces 3 images :



Figure 1: Grille avec du bruit et difficilement lisible



Figure 2: Grille avec des éléments autour et beaucoup de grain



Figure 3: Grille avec des éléments autour, froissée et avec des zones d'exposition différentes

#### 3.1 Chargement des images

Le chargement des images est réalisé à l'aide de GTK, qui gère le chargement et la manipulation des fichiers image.

La rotation est effectuée à l'aide de la bibliothèque graphique cairo, qui offre des transformations 2D de haute qualité.

#### 3.2 Extraction de la grille

Après avoir chargé l'image, on commence par la pré-traitement de l'image pour la rendre plus facile à traiter.

##### 3.2.1 Transformation en niveaux de gris



Figure 4



Figure 5



Figure 6

On commence par transformer l'image en niveaux de gris pour la rendre plus facile à traiter. En utilisant la formule de luminance standard :

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Où  $R$ ,  $G$  et  $B$  sont les valeurs des composantes rouge, vert et bleu de l'image.

### 3.3 Débruitage adaptatif



Figure 7



Figure 8



Figure 9

Le niveau de bruit est estimé par la variance locale ( $3 \times 3$  pixels) des intensités :

$$\sigma_{\text{bruit}}^2 = \frac{1}{|S|} \sum_{s \in S} \frac{1}{N} \sum_{p \in W_s} (I(p) - \mu_s)^2, \quad \mu_s = \frac{1}{N} \sum_{p \in W_s} I(p)$$

Cette étape vise à légèrement réduire le bruit pour faciliter les traitements ultérieurs, plutôt qu'à l'éliminer parfaitement.

Un niveau de bruit normalisé  $\eta = \min(1, \sigma_{\text{bruit}}^2 / 1000)$  ajuste les paramètres du filtre gaussien :

- $\eta < 0.1$  : noyau  $3 \times 3$ ,  $\sigma = 0.5$
- $0.1 \leq \eta < 0.3$  : noyau  $3 \times 3$ ,  $\sigma = 1.0$
- $\eta \geq 0.3$  : noyau  $5 \times 5$ ,  $\sigma = 1.5$

### 3.4 Seuil adaptatif



Figure 10



Figure 11

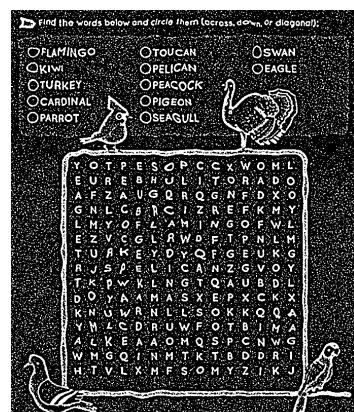


Figure 12

Le seuillage adaptatif binarise l'image en calculant un seuil local  $T(x, y)$  pour chaque pixel, basé sur un voisinage de taille  $\text{block\_size} \times \text{block\_size}$ . Deux méthodes sont utilisées :

### 3.4.1 Méthode de la moyenne

Le seuil  $T(x, y)$  est la moyenne des intensités des pixels dans le voisinage  $W_{xy}$ , moins une constante  $C$  :

$$T(x, y) = \frac{1}{N} \sum_{(i,j) \in W_{xy}} I(i, j) - C$$

Où  $N = \text{block\_size}^2$  est le nombre de pixels dans la fenêtre et  $I(i, j)$  est l'intensité du pixel.

### 3.4.2 Méthode Gaussienne

Le seuil est une somme pondérée des intensités du voisinage  $W_{xy}$ , où les poids  $w(i, j)$  sont dérivés d'un noyau gaussien et normalisés. Une constante  $C$  est soustraite :

$$T(x, y) = \sum_{(i,j) \in W_{xy}} w(i, j) \cdot I(i, j) - C$$

### 3.4.3 Application du seuil

L'image de sortie  $I_{\text{out}}$  est générée en appliquant le seuil  $T(x, y)$  :

- THRESH\_BINARY:

$$I_{\text{out}}(x, y) = \begin{cases} \text{max\_value} & \text{si } I(x, y) > T(x, y) \\ 0 & \text{sinon} \end{cases}$$

- THRESH\_BINARY\_INV:

$$I_{\text{out}}(x, y) = \begin{cases} 0 & \text{si } I(x, y) > T(x, y) \\ \text{max\_value} & \text{sinon} \end{cases}$$

## 3.5 Détection de la grille

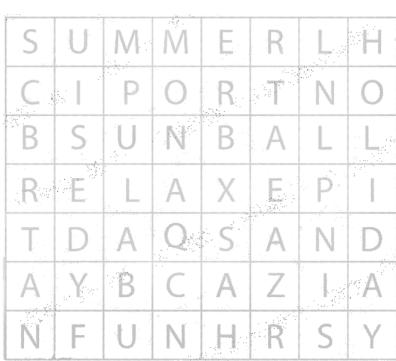


Figure 13



Figure 14



Figure 15

Une fois l'image binarisée, l'étape suivante consiste à localiser et à extraire la grille. L'approche est basée sur l'hypothèse que la grille est le plus grand objet de l'image.

### 3.5.1 1. Recherche des contours

Les contours de tous les objets sont identifiés avec un algorithme de suivi de bordure (inspiré de Suzuki [7]). L'algorithme parcourt l'image à la recherche d'un pixel de départ d'un objet non encore traité. Une fois trouvé, il suit la frontière de l'objet en examinant les 8 pixels voisins (voisinage de Moore) jusqu'à revenir au point de départ. Ce processus est répété pour tous les objets, générant une liste de contours définis par des listes de points.

### 3.5.2 2. Identification du plus grand contour

L'heuristique est que la grille est le plus grand objet. L'aire de chaque contour est calculée, et celui avec la plus grande superficie est sélectionné.

### 3.5.3 3. Calcul de la boîte englobante (Bounding Box)

Un rectangle droit (non incliné) englobant le contour de la grille est calculé pour obtenir ses coordonnées ( $x, y$ ), sa largeur et sa hauteur, simplifiant ainsi l'extraction.

### 3.5.4 4. Extraction de la grille

Avec la boîte englobante, la région de la grille est rognée de l'image en niveaux de gris.

Cette première extraction peut être imprécise et sera affinée par un second algorithme.

## 3.6 Nettoyage morphologique de la grille

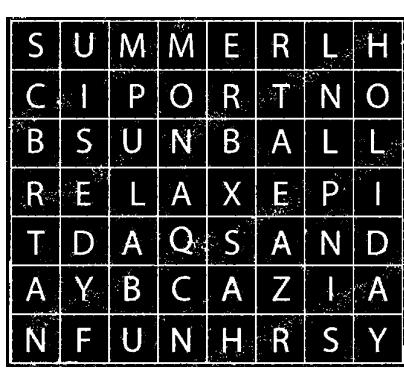


Figure 16

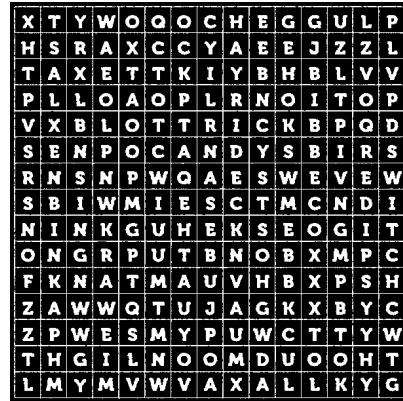


Figure 17

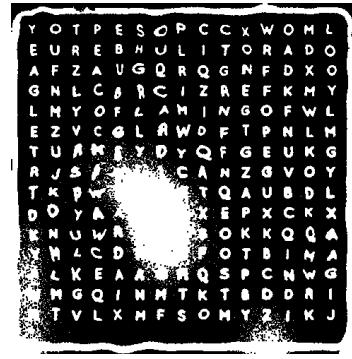


Figure 18

L'image de la grille extraite peut présenter des imperfections telles que des trous ou des coupures dans les lignes. Pour corriger cela, une opération de **fermeture morphologique** est appliquée. Elle combine deux processus pour nettoyer l'image [6]:

- **La dilatation** : Elle épaisse les lignes blanches pour combler les petits trous et reconnecter les segments brisés.
- **Au niveau du pixel** : Un pixel devient blanc si *au moins un* de ses voisins (défini par un élément structurant 2x2) est blanc. Cela étend les zones blanches.
- **L'érosion** : Opération inverse, elle amincit les lignes pour restaurer leur épaisseur d'origine et éliminer le bruit.
- **Au niveau du pixel** : Un pixel ne devient blanc que si *tous* ses voisins sont blancs. Cette règle plus stricte supprime les taches de bruit.

L'effet net de la fermeture est de nettoyer la grille en éliminant les défauts dans les lignes blanches, tout en préservant la structure globale de la grille.

## 3.7 Analyse de la Structure de la Grille

Après avoir isolé et nettoyé l'image de la grille, l'objectif est de la segmenter en cellules individuelles, chacune contenant une seule lettre. L'approche pour y parvenir dépend de la nature de la grille elle-même : certaines grilles sont dessinées avec des lignes internes séparant chaque cellule, tandis que d'autres ne contiennent que les lettres. Notre algorithme doit donc gérer ces deux cas distincts. Cette analyse de la structure interne vient ainsi affiner et compléter l'extraction globale de la grille vue précédemment, en nous permettant d'identifier précisément chaque cellule.

### 3.7.1 Cas 1 : Grille avec lignes internes (Détection par Lignes)

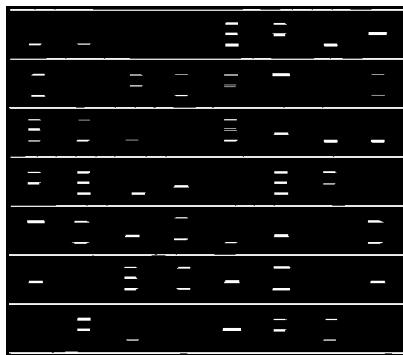


Figure 19

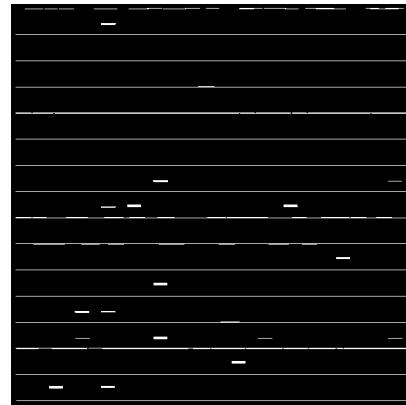


Figure 20

Figure 21: Détection des lignes horizontales.

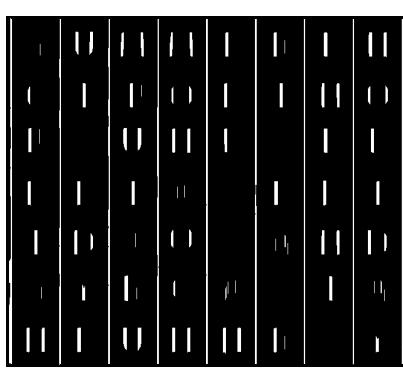


Figure 22

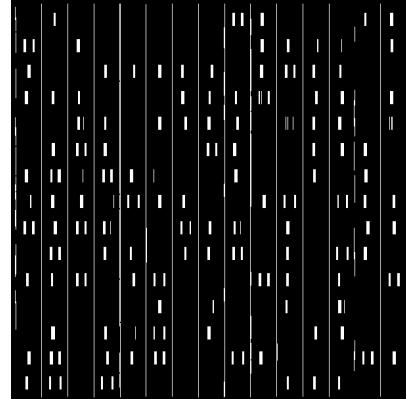


Figure 23

Figure 24: Détection des lignes verticales.

Cette méthode est utilisée lorsque l'image de la grille contient des lignes horizontales et verticales clairement visibles qui délimitent les cellules.

**1. Détection des lignes** Pour extraire les lignes, nous utilisons une opération spécialisée appelée **ouverture morphologique**. Celle-ci est particulièrement efficace pour supprimer les petits objets d'une image tout en préservant la forme et la taille des objets plus grands. L'opération se déroule en deux temps, en utilisant un noyau (ou "élément structurant") adapté à la forme que l'on souhaite conserver. La détection de ligne peut également être fait avec un algorithme de Hough [3].

- **Pour les lignes horizontales**, on utilise un élément structurant très large et fin (par exemple, de 40x1 pixels).
  1. **Étape d'érosion** : L'image est d'abord érodée avec ce noyau horizontal. Rappelons que l'érosion amincit les formes. Comme le noyau est très large, tout objet blanc qui est plus fin que 40 pixels horizontalement (comme les lettres, le bruit, ou les lignes verticales) sera complètement "rongé" et disparaîtra. Seules les longues lignes horizontales survivront à cette étape, bien qu'elles soient amincies.
  2. **Étape de dilatation** : L'image résultante (ne contenant que des lignes horizontales amincies) est ensuite dilatée avec le même noyau. La dilatation épaisse les formes. Cette étape a pour effet de restaurer les lignes horizontales à leur épaisseur d'origine.

Le résultat net de cette ouverture est une image ne contenant que les lignes horizontales de la grille.

- **Pour les lignes verticales**, le principe est identique, mais on utilise un élément structurant très haut et étroit (par exemple, 1x40 pixels).

1. **Étape d'érosion** : L'érosion avec ce noyau vertical supprime tous les objets qui ne sont pas de longues lignes verticales.

2. **Étape de dilatation** : La dilatation qui suit restaure l'épaisseur des lignes verticales qui ont survécu.

**2. Fusion et Extension des Lignes** Les lignes détectées par l'ouverture morphologique peuvent être fragmentées. Pour reconstruire des lignes complètes, un traitement supplémentaire est appliqué. Les segments de ligne qui sont colinéaires (sur le même axe) sont identifiés et fusionnés en un seul segment continu. Ensuite, ces segments fusionnés sont étendus sur toute la largeur (pour les lignes horizontales) ou toute la hauteur (pour les lignes verticales) de la grille. Cette étape garantit que la grille est entièrement délimitée, même si certaines parties des lignes étaient manquantes dans l'image nettoyée.

**3. Détermination des frontières** Une fois les images des lignes obtenues, les frontières des cellules sont déterminées en analysant les projections de ces images. On scanne chaque ligne (pour les frontières horizontales) et chaque colonne (pour les frontières verticales) pour identifier les positions exactes des lignes détectées. Ces positions deviennent les délimitations des cellules.

### 3.7.2 Cas 2 : Grille sans lignes internes (Détection par Lettres)



Figure 25

Lorsque la grille ne contient pas de lignes internes, nous devons déduire sa structure en nous basant sur la disposition des lettres elles-mêmes.

**1. Localisation de la région de texte** D'abord, nous détectons tous les contours dans l'image de la grille et nous les filtrons pour ne garder que ceux qui ressemblent à des lettres (en se basant sur leur taille, leur forme, etc.). Ensuite, nous calculons la boîte englobante de l'ensemble de ces contours de lettres. Cette boîte définit la "région de texte", qui est une version plus précise de la grille, débarrassée des marges vides.

**2. Inférence des dimensions de la grille** Pour déduire le nombre de lignes et de colonnes à partir des positions des lettres, un algorithme de clustering est utilisé. L'idée est de regrouper les lettres qui sont alignées. L'algorithme traite les coordonnées Y et X des centres des lettres de manière indépendante. D'abord, il groupe les coordonnées Y qui sont très proches pour déterminer le nombre de lignes distinctes. Ensuite, il répète le processus sur les coordonnées X pour trouver le nombre de colonnes. Le nombre de groupes finaux pour chaque axe nous donne les dimensions de la grille.

**3. Création d'une grille virtuelle** Connaissant les dimensions de la région de texte et le nombre de lignes et de colonnes, nous pouvons générer une grille virtuelle en divisant simplement la région en cellules de taille égale.

### 3.7.3 Résultat : La grille segmentée

Quelle que soit la méthode utilisée, le résultat final est une série de coordonnées qui définissent les frontières de chaque cellule de la grille. Cela nous permet de "découper" chaque lettre individuellement pour la reconnaissance de caractères.

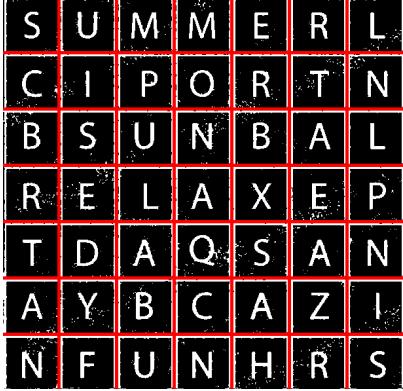


Figure 26

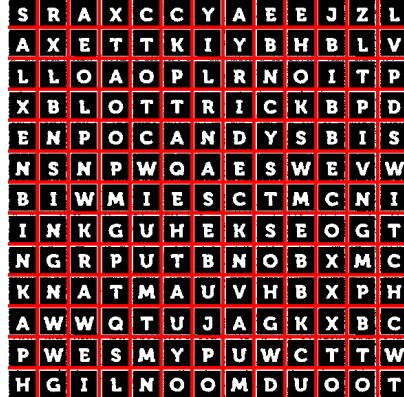


Figure 27



Figure 28

Figure 29: Grille segmentée avec les frontières des cellules détectées.

## 4 Extraction de la Liste de Mots

L'étape finale consiste à extraire la liste des mots à rechercher à partir d'une région de l'image. Ce processus est une chaîne de traitement d'image à part entière, conçue pour isoler des lignes de texte.

### 4.1 Pré-traitement et Binarisation

**1. Préparation de l'image** Comme pour la grille, l'image est d'abord convertie en niveaux de gris et un léger flou gaussien est appliqué pour réduire le bruit et lisser les caractères.

**2. Binarisation par Combinaison** Le texte peut être difficile à segmenter à cause des variations d'éclairage, des ombres ou de la qualité de l'impression. Pour surmonter cela, au lieu d'une seule méthode de seuillage, nous en combinons deux pour maximiser nos chances de capturer tout le texte :

- **Méthode d'Otsu [5] :** La méthode d'Otsu est un algorithme de seuillage automatique qui analyse l'histogramme des intensités de l'image pour trouver le seuil optimal. Ce seuil est la valeur qui maximise la variance entre les deux classes de pixels (premier-plan et arrière-plan) qu'elle sépare. Elle est particulièrement efficace pour les images à contraste élevé, comme du texte noir sur fond blanc.



Figure 30

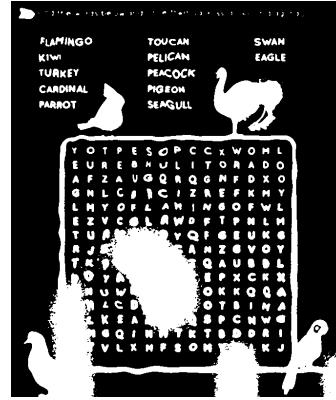


Figure 31

Figure 32: Seuillage par Otsu.

- **Seuillage par la Moyenne** : Une version plus simple du seuillage global, où le seuil est calculé comme un pourcentage (par exemple, 80%) de l'intensité moyenne de tous les pixels de l'image.



Figure 33

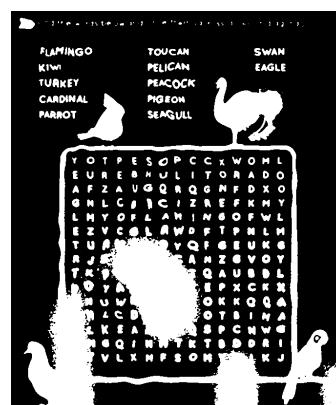


Figure 34

Figure 35: Seuillage par la moyenne.

3. **Fusion des Résultats** Les deux images binaires résultantes sont ensuite fusionnées en une seule image en utilisant une opération OU logique pixel à pixel. Un pixel est considéré comme faisant partie du texte dans l'image finale s'il a été classé comme tel par *au moins une* des deux méthodes.



Figure 36

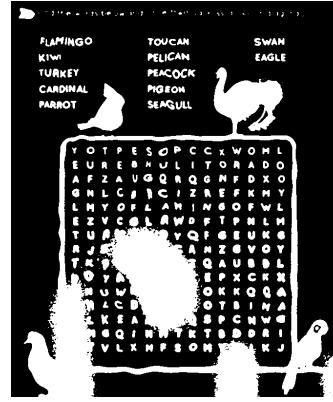


Figure 37

Figure 38: Fusion des résultats.

## 4.2 Analyse et Groupement des Mots

Une fois l'image binarisée, des opérations morphologiques sont appliquées pour transformer les lettres individuelles en des "blobs" solides correspondant à des mots entiers. Une séquence de **fermeture**, de **dilatation** et d'**érosion** permet de combler les trous dans les lettres et de connecter les caractères adjacents.



Figure 39

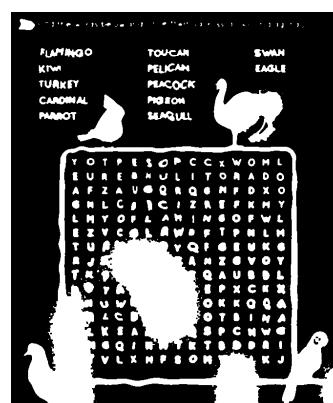


Figure 40

Figure 41: Fermeture morphologique.

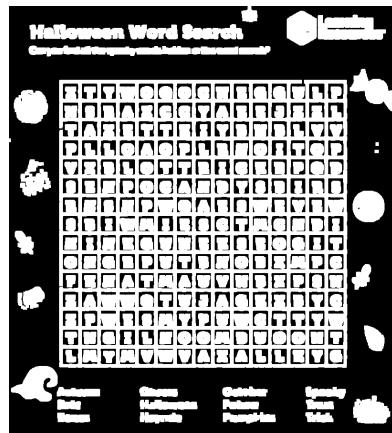


Figure 42

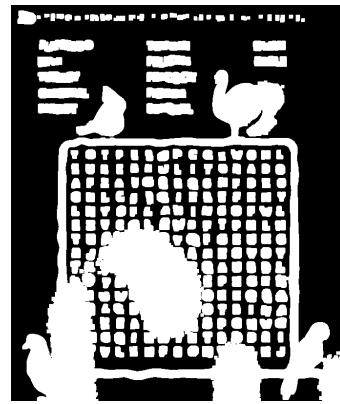


Figure 43

Figure 44: Dilatation.



Figure 45

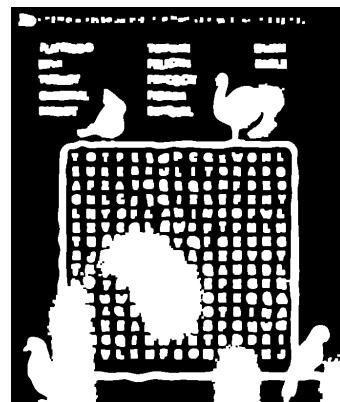


Figure 46

Figure 47: Érosion.

**1. Détection et Filtrage :** Comme précédemment, les contours de tous les "blobs" sont détectés, et ceux qui ne correspondent pas à des critères géométriques de mots (taille, ratio, etc.) sont éliminés.

**2. Fusion Horizontale des Mots :** Pour regrouper les fragments de mots, un algorithme de fusion est appliqué. Après avoir trié les boîtes englobantes des contours par leur coordonnée X, l'algorithme les parcourt. Si une boîte est suffisamment proche horizontalement et alignée verticalement avec la précédente, elles sont fusionnées. Sinon, la boîte précédente est considérée comme un mot complet. Ce processus garantit que seuls les fragments de mots adjacents et colinéaires sont assemblés.

**3. Identification de la Liste de Mots :** Pour identifier la liste de mots principale, les mots sont d'abord regroupés en lignes par clustering vertical de leurs centres. Ensuite, l'algorithme analyse l'espacement vertical entre les lignes pour identifier l'interligne le plus courant. En se basant sur cet interligne, il regroupe les lignes en blocs. Finalement, le bloc contenant le plus grand nombre de mots est sélectionné comme étant la liste de mots à rechercher. Le résultat est un ensemble final de boîtes englobantes, chacune correspondant à un mot à trouver.



Figure 48



Figure 49

Figure 50: Détection des mots.

## 5 Réseau de Neurones pour la fonction XNOR

Pour démontrer les principes des réseaux de neurones, un simple réseau a été implémenté en C pour apprendre la fonction logique XNOR.

### 5.1 Architecture du réseau

Le réseau possède une architecture de type *feed-forward* composée de trois couches :

- **Couche d'entrée** : Elle est composée de 2 neurones, qui correspondent aux deux entrées binaires de la fonction XNOR.
- **Couche cachée** : Elle contient 2 neurones. Chaque neurone est entièrement connecté aux neurones de la couche d'entrée.
- **Couche de sortie** : Elle est constituée d'un seul neurone, qui fournit le résultat final de l'opération XNOR.

Tous les neurones du réseau (ceux de la couche cachée et de la couche de sortie) utilisent la fonction d'activation sigmoïde :

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

Cette fonction a pour caractéristique de borner les sorties des neurones entre 0 et 1, ce qui est adapté pour des problèmes de classification binaire. Sa dérivée, utilisée lors de la rétropropagation de l'erreur, est simple à calculer :

$$f'(x) = f(x) \cdot (1 - f(x))$$

### 5.2 Apprentissage par Rétropropagation

L'entraînement du réseau est réalisé grâce à l'algorithme de rétropropagation du gradient (*backpropagation*).

- **Initialisation** : Les poids des connexions et les biais des neurones sont initialisés avec des valeurs aléatoires.
- **Propagation avant (*Forward Propagation*)** : Pour chaque exemple d'entraînement, les entrées sont propagées à travers le réseau, de la couche d'entrée à la couche de sortie, pour calculer une prédiction.
- **Calcul de l'erreur** : L'erreur entre la sortie prédictive et la sortie attendue est calculée (par exemple, avec l'erreur quadratique moyenne).
- **Rétropropagation de l'erreur** : L'erreur est ensuite propagée en sens inverse, de la couche de sortie vers la couche d'entrée. À chaque couche, on calcule le gradient de l'erreur par rapport aux poids et aux biais.

- **Mise à jour des poids** : Les poids et les biais du réseau sont ajustés dans la direction opposée du gradient, afin de minimiser l'erreur. Cette mise à jour est pondérée par un *taux d'apprentissage* (learning rate).

Les hyperparamètres utilisés pour l'entraînement étaient les suivants :

- Taux d'apprentissage : 0.5
- Nombre d'époques : 100 000

### 5.3 Résultats et Performance

Après 100 000 époques d'entraînement, le réseau est capable de prédire la sortie de la fonction XNOR avec une précision parfaite pour les quatre combinaisons d'entrées possibles. Le temps d'exécution total pour l'entraînement était de **22.727 ms**.

Input	Output Attendu	Output Prédit	Sortie brute du réseau
0 0	1	1	0.992813
0 1	0	0	0.006186
1 0	0	0	0.006188
1 1	1	1	0.992620

Table 1: Résultats de prédiction pour la fonction XNOR après entraînement.

La performance du modèle est la suivante :

- **Précision (Accuracy)** : 4/4 (100.0%)
- **Erreur Quadratique Moyenne (MSE)** : 0.000046

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Où  $n$  est le nombre d'exemples (4 dans ce cas),  $y_i$  est la sortie attendue et  $\hat{y}_i$  est la prédiction du réseau.

## 6 Solver

Le programme solver a pour but de renvoyer les coordonnées auxquelles se situe le mot recherché dans une grille.

Il prend deux arguments en entrée :

- Le nom d'un fichier texte : ce fichier doit contenir une grille de caractères en majuscules et comporter au minimum cinq lignes et cinq colonnes.
- Le mot à chercher dans la grille (en majuscules ou en minuscules).

Le programme affiche sur la sortie standard :

- “Not Found” si le mot n'a pas été trouvé ;
- $(x_0, y_0)(x_1, y_1)$  si le mot a été trouvé, avec :
  - $(x_0, y_0)$  = abscisse et ordonnée (en caractères) de la première lettre du mot dans la grille ;
  - $(x_1, y_1)$  = abscisse et ordonnée (en caractères) de la dernière lettre du mot dans la grille.

Les coordonnées  $(0,0)$  correspondent au premier caractère de la grille (en haut à gauche).

## 6.1 Fichiers utilisés

Le programme solver nécessite l'utilisation de cinq fichiers :

- `main.c`
- `solver.c`
- `solver.h`
- `search.c`
- `search.h`

## 6.2 `main.c`

Le fichier `main.c` constitue la base du programme. Il prend les deux arguments cités précédemment en entrée, lit le fichier texte, puis stocke la grille sous forme d'un tableau de tableaux de caractères.

Ensuite, il vérifie la validité du mot à chercher et convertit toutes ses lettres en majuscules. Enfin, il appelle la fonction `solver()`.

## 6.3 `solver.c`

Le fichier `solver.c` parcourt la grille à la recherche de la première lettre du mot recherché. Lorsqu'elle est trouvée, il appelle les fonctions de recherche présentes dans `search.c` :

- `north_search` : recherche le mot dans la direction Nord ;
- `south_search` : recherche le mot dans la direction Sud ;
- `east_search` : recherche le mot dans la direction Est ;
- `west_search` : recherche le mot dans la direction Ouest ;
- `northeast_search` : recherche le mot dans la direction Nord-Est ;
- `northwest_search` : recherche le mot dans la direction Nord-Ouest ;
- `southeast_search` : recherche le mot dans la direction Sud-Est ;
- `southwest_search` : recherche le mot dans la direction Sud-Ouest.

Si le mot est trouvé, la fonction renvoie ses coordonnées. Sinon, elle renvoie "Not Found".

## Bibliographie

### References

- [1] G. Bradski. The opencv library. Dr. Dobb's Journal of Software Tools, 2000.
- [2] John S Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in neural information processing systems*, volume 2, 1990.
- [3] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.
- [4] Andrej Karpathy. micrograd: A tiny scalar-valued autograd engine and a neural net library on top of it with pytorch-like api. <https://github.com/karpathy/micrograd>, 2022.
- [5] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.
- [6] Jean Serra. *Image analysis and mathematical morphology*. Academic press, 1982.
- [7] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.
- [8] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.